Algoritma dan Struktur Data

Muh. Izzuddin Mahali, M.Cs.

PT. Elektronika FT UNY

Program

- Program: sederetan perintah-perintah yang harus dikerjakan oleh komputer untuk menyelesaikan masalah.
- 3 level bahasa pemrograman:

  1. Bahasa tingkat rendah
  2. Bahasa tingkat menengah
  3. Bahasa tingkat tinggi

Bahasa Tingkat Rendah

Bahasa mesin

Berisi: kode-kode mesin yg hanya dapat diinterpretasikan langsung oleh mesin komputer.

Berupa kode numerik 0 dan 1

Microcode: sekumpulan instruksi dalam bahasa mesin

(+) : Eksekusi cepat

(-) : Sulit dipelajari manusia

Bahasa Tingkat Menengah

Bahasa Assembly

- Bahasa simbol dari bahasa mesin
- Contoh: ADD, MUL, SUB, dll

Macro instruksi: sekumpulan kode dalam bahasa assembly

- (+) : Eksekusi cepat, masih dapat dipelajari daripada bahasa mesin, file kecil
- (-) : Tetap sulit dipelajari, program sangat panjang

Bahasa Tingkat Tinggi

The 3rd Generation Programming Language

- Lebih dekat dengan bahasa manusia
- Memberi banyak fasilitas kemudahan dalam pembuatan program, mis.: variabel, tipe data, konstanta, struktur kontrol, loop, fungsi, prosedur, dll
- Contoh: Pascal, Basic, C++, Java

(+) : Mudah dipelajari, mendekati permasalahan yang akan dipecahkan, kode program pendek

(-) : Eksekusi lambat

Translator

- Translator: penerjemah dari bahasa tingkat tinggi ke bahasa tingkat rendah.
- Assembler, Interprenter, dan Compiler
- Assembler merupakan penerjemah bahasa Assembly ke bahasa mesin.

Interpreter

Perintah diterjemahkan baris demi baris jadi program tidak dianalisis seluruhnya dulu tapi bersamaan dengan jalannya program.

(+) : mudah bagi user, debugging cepat

(-) : eksekusi program lambat, tidak langsung menjadi program executable

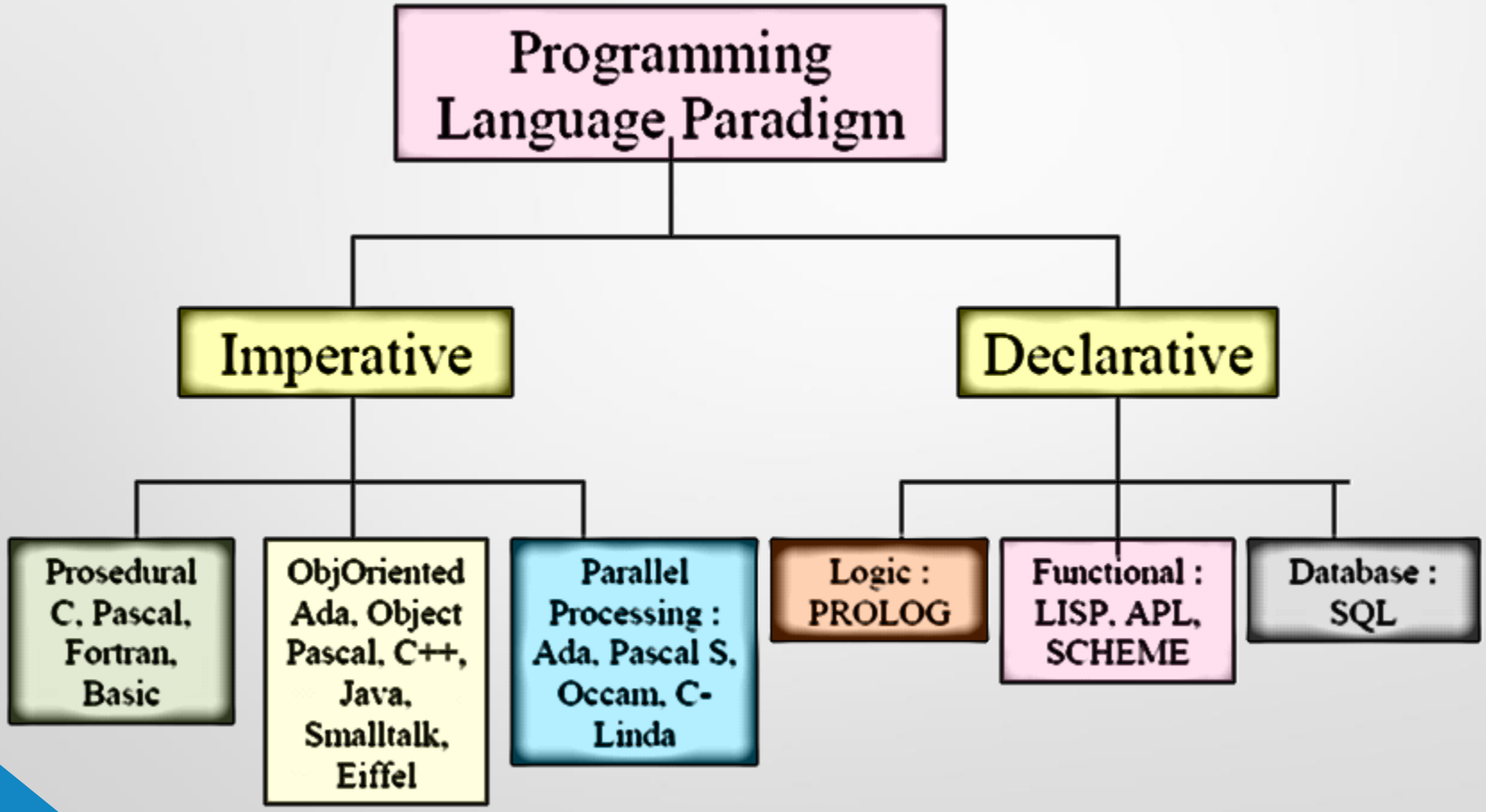
Contoh: Basic, List

Compiler

Seluruh program diterjemahkan.

Semua perintah (pascal, C++) dirubah dalam bentuk exe atau bahasa asembly.

Paradigma Bahasa Pemrograman



PT. Elektronika FT UNY

Muh. Izzuddin Mahali, M.Cs.

Sejarah Algoritma

Ditinjau dari asal usul katanya kata Algoritma sendiri mempunyai sejarah yang aneh. Orang hanya menemukan kata Algorism yang berarti proses menghitung dengan angka arab. Anda dikatakan Algorist jika anda menghitung menggunakan Angka Arab. Para ahli bahasa berusaha menemukan asal kata ini namun hasilnya kurang memuaskan.

Akhirnya para ahli sejarah matematika menemukan asal kata tersebut yang berasal dari nama seorang ahli matematika dari Uzbekistan Abu Abdullah Muhammad Ibnu Musa Al-Khuwarizmi (770- 840). Al-Khuwarizmi dibaca orang barat menjadi Algorism. Al-Khuwarizmi menulis buku yang berjudul Kitab Al Jabar Wal-Muqabala yang artinya "Buku pemugaran dan pengurangan" (The book of restoration and reduction). Dari judul buku itu kita juga memperoleh akar kata "Aljabar" (Algebra).

Sejarah Algoritma

Perubahan kata dari Algorism menjadi Algorithm muncul karena kata Algorism sering dikelirukan dengan Arithmetic, sehingga akhiran –sm berubah menjadi –thm. Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa. Maka lambat laun kata Algorithm berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna kata aslinya. Dalam Bahasa Indonesia, kata Algorithm diserap menjadi Algoritma.

Definisi Algoritma

Kita bisa mendefinisikan algoritma sebagai berikut:

" Algoritma adalah logika, metode dan tahapan (urutan) sistematis yang digunakan untuk memecahkan suatu permasalahan."

Dan kamus besar bahasa Indonesia (Balai Pustaka 1988) secara formal mendefinisikan algoritma sebagai berikut:

"Algoritma adalah urutan logis pengambilan putusan untuk pemecahan masalah."

Definisi Algoritma

Algoritma: sederetan langkah-langkah logis yang disusun secara sistematis untuk memecahkan suatu masalah.

Disebut Logis karena setiap langkah bisa diketahui dengan pasti.

Algoritma lebih merupakan alur pemikiran untuk menyelesaikan suatu pekerjaan atau suatu masalah.

Ciri Ciri Algoritma

1. Algoritma harus berhenti setelah mengerjakan sejumlah langkah terbatas.
2. Setiap langkah harus didefinisikan dengan tepat dan tidak berarti-dua (Ambiguitas).
3. Algoritma memiliki nol atau lebih masukkan.
4. Algoritma memiliki satu atau lebih keluaran.
5. Algoritma harus efektif (setiap langkah harus sederhana sehingga dapat dikerjakan dalam waktu yang masuk akal).

Pengertian Struktur Data

Struktur data adalah cara menyimpan atau merepresentasikan data di dalam komputer agar bisa dipakai secara efisien Sedangkan data adalah representasi dari fakta dunia nyata.

Fakta atau keterangan tentang kenyataan yang disimpan, direkam atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal atau simbol

Type Data

1. Type data sederhana
    a. Type data sederhana tunggal, misalnya
        Integer, real, boolean dan karakter
    b. Type data sederhana majemuk, misalnya
        String
2. Struktur Data, meliputi
    a. Struktur data sederhana, misalnya array dan record
    b. Struktur data majemuk, yang terdiri dari
        Linier : Stack, Queue, serta List dan Multilist
        Non Linier : Pohon Biner dan Graph

Pemakaian struktur data yang tepat di dalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

Belajar Memprogram dan Belajar Bahasa Pemrograman

Belajar memprogram adalah belajar tentang metodologi pemecahan masalah, kemudian menuangkannya dalam suatu notasi tertentu yang mudah dibaca dan dipahami.

Belajar bahasa pemrograman adalah belajar memakai suatu bahasa, aturan tata bahasanya, instruksi-instruksinya, tata cara pengoperasian compiler-nya untuk membuat program yang ditulis dalam bahasa itu saja.

Notasi Algoritma

- Penulisan algoritma tidak tergantung dari spesifikasi bahasa pemrograman dan komputer yang mengeksekusinya.
- Notasi algoritma bukan notasi bahasa pemrograman tetapi dapat diterjemahkan ke dalam berbagai bahasa pemrograman.

Notasi Algoritma

1. Uraian kalimat deskriptif (narasi)

   Contoh:

   Algoritma Kelulusan_mhs

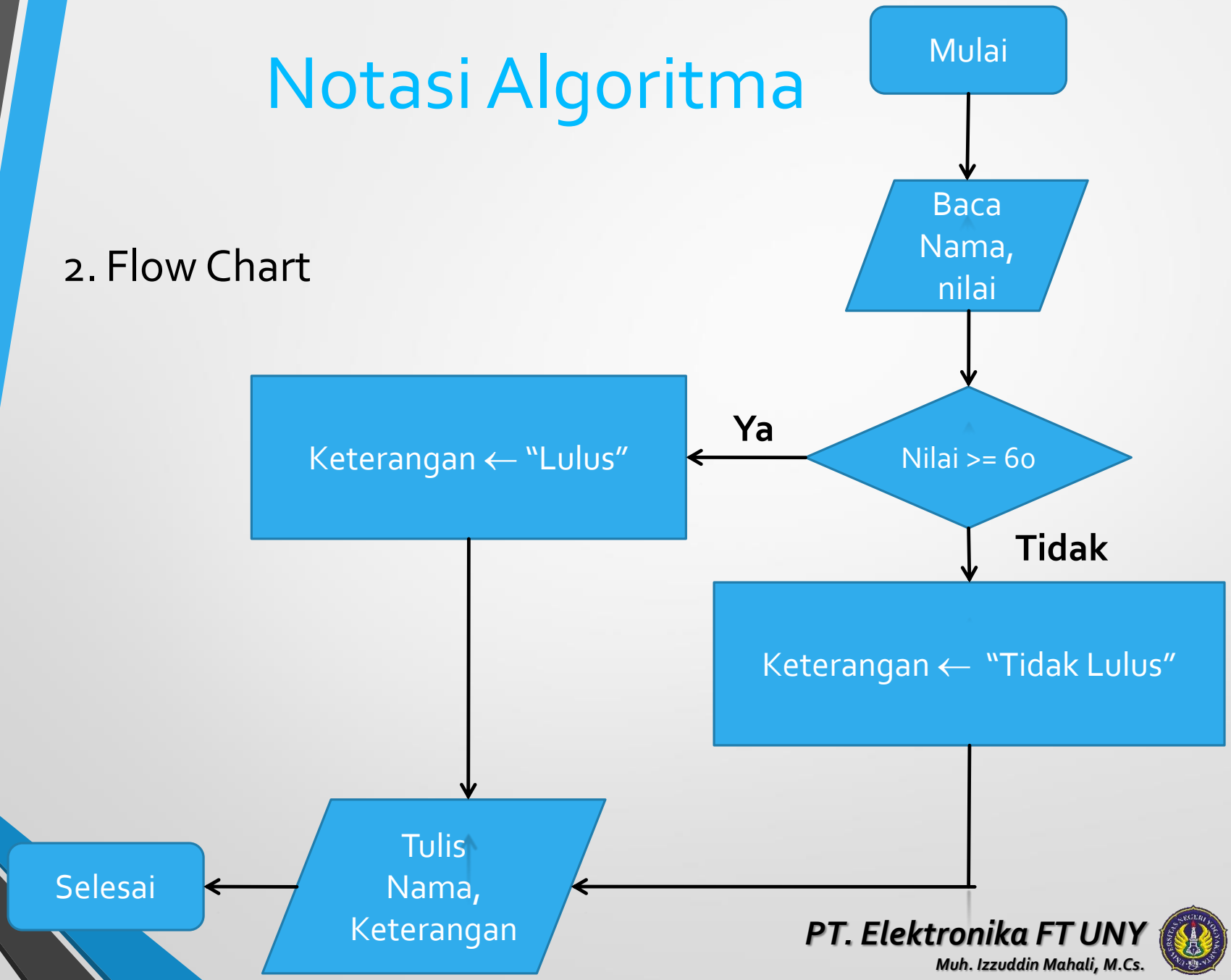
   Diberikan nama dan nilai mahasiswa, jika nilai tersebut lebih besar atau sama dengan 60 maka mahasiswa tersebut dinyatakan lulus jika nilai lebih kecil dari 60 maka dinyatakan tidak lulus.

   DESKRIPSI :

   1. baca nama dan nilai mahasiswa.
   2. jika nilai >= 60 maka
   3. Berikan keterangan ← "lulus"
   4. tetapi jika tidak
   5. Berikan keterangan ← "tidak lulus"
   6. tulis nama dan keterangan

Notasi Algoritma

2. Flow Chart



Notasi Algoritma

3. Pseudo Code

   Ada 3 bagian: Judul, Deklarasi, Deskripsi.

   Algoritma kelulusan

   Deklarasi

```
nama, keterangan : string
nilai : integer
```

   Deskripsi

```
read (nama, nilai);
if nilai >= 60 then
   keterangan := "lulus";
else
   keterangan := "tidak lulus";
write(nama, keterangan);
```

Aturan Pseudo Code

- **Judul algoritma**
  Bagian yang terdiri atas nama algoritma dan penjelasan (spesifikasi) tentang algoritma tersebut. Nama sebaiknya singkat dan menggambarkan apa yang dilakukan oleh algoritma tersebut.
- **Deklarasi**
  Bagian untuk mendefinisikan atau mendeklarasikan semua apa yang digunakan atau dibutuhkan dalam pemrograman.
- **Deskripsi**
  Bagian ini berisi uraian langkah-langkah penyelesaian masalah.

Operator Aritmatik

- /, *, div, mod → level tinggi
- +, - → level rendah
- Mod dan div hanya untuk bilangan bulat!!!

- Contoh :
- 5 – 2 + 1 = ?
- 5 – 2 * 3 = ?
- (6 + 3 * 2) / 6 – 2 * 3 = ?

Type Data

- Bilangan bulat (Shortint , Integer, Longint, Byte, Word)
- Boolean (Boolean, ByteBool , WordBool, LongBool)
- Bilangan real (Real, Single, Double, Extended, Comp)
- Karakter
- String
- Larik (Array)
- Pointer

Outline Algoritma dan Struktur Data

- Logika Pemrograman Pengertian dasar algoritma
- Algoritma dan Flowchart
- Array, Record dan pointer
- Fungsi dan Prosedur
- Searching dan sorting
- Stack dan queue
- Linked list
- Rekursif
- Tree
- Hashing

S E L E S A I

Percabangan

Muh. Izzuddin Mahali, M.Cs.

Pertemuan 2. Algoritma dan Struktur Data

Percabangan dan Seleksi

- Struktur seleksi untuk melakukan proses pengujian dalam mengambil suatu keputusan guna mengeksekusi suatu blok instruksi, yang menilai dua atau beberapa keadaan sekaligus.
- Kondisi yang dinilai berupa ekspresi dengan nilai True atau False
- Jika True maka proses akan mengeksekusi statemen pertama , Sebaliknya jika bernilai False maka Proses akan mengeksekusi statemen ke dua.

Operator Relasi

Operator relasi yang digunakan membandingkan hubungna Antara dua buah operand akan didapatkan hasil tipe Boolean. True (benar) dan False (salah).

Operator	Operasi
=	Sama Dengan
<>	Tidak Sama Dengan
>	Lebih besar dari
>=	Lebih Besar dari Sama Dengan
<	Lebih Kecil dari
<=	Lebih Kecil dari Sama Dengan

Pernyataan If (If Statement)

1. Statement IF .. THEN

Bentuk umum dari IF .. THEN dengan suatu pernyataan adalah

```
IF <kondisi> THEN
Begin
   ……
   ……
End.
```

Pernyataan If (If Statement)

Contoh :

```
Program BilPositif;
Uses WINCRT;
Var
   Bil : Integer;
Begin
   Write('Masukkan sebuah bilangan : ');
   Readln(Bil);
   IF Bil>0 THEN
       Writeln ('Bilangan Positif');
   Readln;
End.
```

```
Hasil eksekusi :
   Masukkan sebuah bilangan : 3
   Bilangan Positif
```

Pernyataan If (If Statement)

2. Statement IF .. THEN .. ELSE

   Bentuk umum dari IF .. THEN .. ELSE adalah

```
IF <kondisi> THEN
   Begin
      ……
      ……
   End;
ELSE
   Begin
      ……
      ……
   End;
```

Pernyataan If (If Statement)

Contoh :

```
Program BilPositif;
Uses WINCRT;
Var
    Bil : Integer;
Begin
    Write('Masukkan sebuah bilangan : ');
    Readln(Bil);
    IF Bil>0 THEN
         Writeln ('Bilangan Positif');
    ELSE
         Writeln ('Bilangan Negatif');
    Readln;
End.
```

Hasil eksekusi :

```
    Masukkan sebuah bilangan : -13
    Bilangan Negatif
```

Pernyataan If (If Statement)

3. Statement IF .. THEN .. ELSE IF

Statement IF .. THEN .. ELSE IF digunakan untuk menyelesaikan permasalahan dengan jumlah kondisi lebih dari 2 buah :

Bentuk umum dari statement ini adalah :

```
IF <kondisi 1> THEN
   Begin
      ……
   End;
ELSE IF <kondisi 2> THEN
   Begin
      ……
   End;
```

Pernyataan If (If Statement)

Contoh :

```
Program Deret;
Uses WINCRT;
Var
    Bil : Integer;
Begin
    Write('Masukkan sebuah bilangan : ');
    Readln(Bil);
    IF Bil Mod 2 = 1 THEN
         Writeln ('Bilangan Ganjil');
    ELSE IF Bil Mod 2 = 0 THEN
         Writeln ('Bilangan Genap');
    Readln;
End.
```

Hasil eksekusi :

```
    Masukkan sebuah bilangan : 13
    Bilangan Ganjil
```

CASE .. OF Statement

Pernyataan Case merupakan alternative dari statement IF dengan pilihan ganda, biasanya pada masalah tertentu. Case akan lebih memberi kejelasan dibandingkan dengan IF dan semua permasalahan yang dibuat dengan IF dapat diselesaikan dengan Case.

CASE .. OF merupakan suatu ungkapan logika yang disebut dengna selector dan sejumlah statementyang diawali dengan label permasalahan (case label)

CASE .. OF Statement

Bentuk umum CASE .. OF adalah :

```
CASE ungkapan OF
   Daftar label 1 : Statement 1;
   Daftar label 2 : Statement 2;
   Daftar label 3 : Statement 3;
   ……
End;
```

CASE .. OF Statement

Contoh :

```
Uses WINCRT;
Var
    alas, tinggi, jejari, luas: Real;
    pilih: Integer;
Begin
     Writeln('1. Segitiga 2. Lingkaran');
     Readln(pilih);
CASE pilih OF
     1: Begin
           Readln(alas, tinggi);
           luas := 1 / 2 * alas * tinggi;
           Writeln('Luas Segitiga : ', luas);
        End;
     2: Begin
           Readln(jejari);
           luas := pi * Sqr(jejari);
           Writeln('Luas Lingkarang : ', luas);
        end;
     End;
     Readln;
End.
```

S E L E S A I

Perulangan

Muh. Izzuddin Mahali, M.Cs.

Pertemuan 3. Algoritma dan Struktur Data

Pendahuluan

Digunakan untuk program yang pernyataannya akan dieksekusi berulang-ulang. Instruksi dikerjakan selama memenuhi suatu kondisi tertentu. Jika syarat (kondisi) masih terpenuhi maka pernyataan (aksi) akan terus dilakukan secara berulang.

Struktur Perulangan

1. Struktur For
   a) Perulangan Positif
   b) Perulangan Negatif
   c) Perulangan Bersarang
2. Struktur While .. Do
3. Struktur Repeat .. Until

Struktur For

Digunakan untuk mengulang statemen berulang kali sejumlah yang ditentukan.

- Perulangan Positif

```
FOR variable control := nilai awal To Nilai akhir DO Statemen
Ket.
    Nilai awal < Nilai akhir
```

- Perulangan Negatif

```
FOR variable control := nilai awal To Nilai akhir DO Statemen
    Ket.
        Nilai awal > Nilai akhir
```

Struktur For Perulangan Positif

FOR variable control : = nilai awal TO nilai akhir DO statemen

Contoh program (1) :

```
Var
I : integer ;
Begin
    For I : = 1 to 5 do
        Write ( I ) ;
        Writeln ('Pascal');
End.
```

Output program (1) :

```
    12345Pascal
```

Struktur For Perulangan Positif

Contoh program (2) :

```
Var
I : integer ;
Begin
    For I : = 1 to 5 do
        Begin
            Write ( I ) ;
            Writeln ('Pascal');
        End;
End.
```

Output program (1) :

1Pascal

2Pascal

3Pascal

4Pascal

5Pascal

Struktur For Perulangan Negatif

Dengan penghitung / counter dari besar ke kecil (pertambahannya negatif).

Bentuk umum :

FOR variable control : = nilai awal DOWN TO nilai akhir DO statemen

Contoh program :

```
Var
I : integer ;
Begin
     For I : = 5 down to 1 do
          Begin
               Write ( I ) ;
               Writeln ('Pascal');
          End ;
     End.
```

Output program :

```
5Pascal
4Pascal
3Pascal
2Pascal
1Pascal
```

Struktur For Perulangan Bersarang

Perulangan yang berada didalam perulangan yang lainnya. Perulangan yang lebih dalam akan diproses lebih dulu sampai habis, kemudian perulangan yang lebih luar baru akan bertambah, mengerjakan perulangan yang lebih dalam lagi mulai dari nilai awalnya dan seterusnya.

Struktur For Perulangan Bersarang

Contoh program :

```
Var
I, J : integer ;
Begin
    For I : = 1 to 5 do
    Begin
        For J : = 1 to 3 do
            Write ( I : 8, J : 3);
        Writeln ;
    End ;
End.
```

Struktur While .. Do

Digunakan untuk melakukan proses perulangan suatu statemen terus menerus selama kondisi ungkapan logika pada while masih bernilai logika benar.

Bentuk umum :

WHILE ungkapan logika DO statemen

Contoh program :

```
Var
I : integer ;
Begin
     I : = 0 ;
     While I < 5 do
          Begin
               Writeln (I);
               I : = I + 1 ;
          End ;
End.
```

Output program : o

1

2

3

4

Struktur Repeat .. Until

Digunakan untuk mengulang statemen sampai kondisi yang diseleksi di *Until* tidak terpenuhi.

Bentuk umum :

REPEAT statemen UNTIL ungkapan

Contoh program :

```
Var
I : integer ;
Begin
    I : = 0 ;
    Repeat
        I : = I + 1 ;
        Writeln (I) ;
    Until I = 5;
End.
```

Output program : 1
2
3
4
5

Perbedaan While..Do dengan Repeat .. Until

Perbedaan antara struktur " repeat until " dengan " while do " adalah :

- - Paling sedikit statemen-statemen dalam repeat until diproses sekali, karena seleksi kondisi ada pada statemen until yang terletak dibawah.
- - Pada while do paling sedikit dikerjakan nol kali, karena seleksi kondisi ada pada statemen while yang terletak diatas, sehingga apabila kondisi tidak terpenuhi maka tidak akan masuk ke dalam lingkungan perulangannya.
- - Pada repeat until dapat tidak menggunakan blok statemen (BEGIN dan END) untuk menunjukan batas perulangannya, karena batas perulangannya sudah ditunjukkan oleh repeat sampai dengan until.

Latihan

1

12

123

1234

12345

Latihan

- 1 2 4 7 11 16 22 .....
- 1 -2 3 -4 5 -6 7 -8 .....

Diskusikan!

- 1 2 3 4 5

  6 7 8 9 10

  11 12 13 14 15

  16 17 18 19 20

- Buatlah program menghitung Faktor Persekutuan Terbesar (FPB) dari dua bilangan yang diinputkan!

PROSEDUR DAN FUNCTION

PROSEDUR DAN FUNCTION

PROSEDUR

Prosedur adalah suatu program yang terpisah dalam blok sendiri yang berfungsi sebagai seubprogram (program bagian). Prosedur diawali dengan kata cadangan PROCEDURE di dalam bagian deklarasi prosedur. Prosedur dipanggil dan digunakan di dalam blok program lainnya dengan menyebutkan judul prosedurnya.

```
Bentuk penulisan :
PROGRAM judul-program;
  PROCEDURE judul-prosedur;
  BEGIN
    ...
  END;

BEGIN
   ....
END.
```

Contoh :

```
PROGRAM prosedur;
USES CRT;
PROCEDURE garis;
BEGIN
  WRITELN('----------------');
END;

BEGIN
  CLRSCR;
  garis;
  WRITELN('BELAJAR PROSEDUR');
  garis;
  READLN;
END.
```

Hasil :

```
--------------------------------
BELAJAR PROSEDUR
--------------------------------
```

PARAMETER DALAM PROSEDUR

Nilai di dalam suatu modul program Pascal sifatnya adalah lokal, artinya hanya dapat digunakan pada modul atau unit program yang bersangkutan saja, tidak dapat digunakan pada modul atau unit program yang lain.

PENGIRIMAN PARAMETER SECARA NILAI

Bila parameter dikirim secara nilai (by value), parameter formal di prosedur akan berisi nilai yang dikirimkan yang kemudian bersifat lokal di prosedur. Bila nilai parameter formal di dalam prosedur tersebut berubah, tidak akan mempengaruhi nilai parameter nyata.

Pengiriman nilai ini merupakan pengiriman searah yang tidak dikirimkan balik dari parameter formal ke parameter nyata. Parameter yang digunakan dengan pengiriman secara nilai ini disebut dengan parameter nilai (value parameter)

Contoh :

```
PROGRAM parameter_value;
USES CRT;
PROCEDURE hitung(A,B,C : INTEGER);
BEGIN
   C:=A+B;
   WRITELN('Nilai di Prosedur');
   WRITELN('NILAI A= ',A,' NILAI B= ',B,' NILAI C= ',C);
END;
VAR
  X,Y,Z : INTEGER;
BEGIN
  CLRSCR;
  WRITE('NILAI X= '); READLN(X);
  WRITE('NILAI Y= '); READLN(Y);
  WRITE('NILAI Z= '); READLN(Z);
  hitung(X,Y,Z);
  WRITELN('Nilai setelah Prosedur');
```

```
WRITELN('NILAI X= ',X,' NILAI Y= ',Y,' NILAI Z= ',Z);
  READLN;
END.
```

Hasil :

```
NILAI X= 3
NILAI Y= 4
Nilai parameter Z sebelum dan sesudah prosedur
  sama
NILAI Z= 5 Nilai di Prosedur
NILAI A= 3 NILAI B= 4 NILAI C= 7
Nilai setelah Prosedur
NILAI X= 3 NILAI Y= 4 NILAI Z= 5
```

PENGIRIMAN PARAMETER SECARA ACUAN

Bila pengiriman parameter secara acuan (by reference), maka perubahan-perubahan yang terjadi pada nilai parameter formal di prosedur akan mempengaruhi nilai parameter nyata. Parameter-parameter ini disebut dengan *variabel parameter* serta dideklarasikan di deklarasi prosedur dengan menggunakan kata cadangan VAR.

```
Contoh :
PROGRAM parameter_reference;
USES CRT;
PROCEDURE hitung(VAR A,B,C : INTEGER);
BEGIN
   C:=A+B;
   WRITELN('Nilai di Prosedur');
   WRITELN('NILAI A= ',A,' NILAI B= ',B,' NILAI C= ',C);
END;
```

```
VAR
  X,Y,Z : INTEGER;
BEGIN
  CLRSCR;
  WRITE('NILAI X= '); READLN(X);
  WRITE('NILAI Y= '); READLN(Y);
  WRITE('NILAI Z= '); READLN(Z);
  hitung(X,Y,Z);
  WRITELN('Nilai setelah Prosedur');
  WRITELN('NILAI X= ',X,' NILAI Y= ',Y,' NILAI Z= ',Z);
  READLN;
END.
```

Hasil :

```
NILAI X= 3
NILAI Y= 4
NILAI Z= 0
Nilai parameter Z sebelum dan sesudah prosedur berbeda
Nilai di Prosedur NILAI A= 3 NILAI B= 4 NILAI C= 7
Nilai setelah Prosedur
NILAI X= 3 NILAI Y= 4 NILAI Z= 7
```

PROSEDUR MEMANGGIL PROSEDUR YANG LAIN

Di dalam prosedur dapat memanggil prosedur yang lainnya.

```
Contoh :
PROGRAM panggil_prosedur;
USES CRT;
PROCEDURE pro1(x1 : INTEGER);
BEGIN
   x1:=x1*x1;
   WRITELN('Nilai X di Prosedur1 = ',x1);
END;


PROCEDURE pro2(x2 : INTEGER);
BEGIN
   WRITELN('Nilai X di Prosedur2 = ',x2);
   pro1(x2);
END;
```

```
VAR
   X : INTEGER;
BEGIN
   CLRSCR;
   WRITE('Masukkan nilai X= '); READLN(x);
   pro2(x);
   READLN;
END.
```

Hasil :

```
Masukkan nilai X= 12
Nilai X di Prosedur2 = 12
Nilai X di Prosedur1 = 144
```

PROSEDUR TERSARANG

Prosedur tersarang (nested procedure) adalah prosedur yang berada di dalam prosedur yang lainnya.

```
Contoh :
PROGRAM nested_prosedur;
USES CRT;

PROCEDURE kesatu;
  PROCEDURE kedua;
  BEGIN
     WRITELN('Prosedur KEDUA ini ada di dalam prosedur KESATU');
  END;

 PROCEDURE ketiga;
  BEGIN
     WRITELN('Prosedur  KETIGA  ini  juga  ada  di  dalam  prosedur
   KESATU');
  END;
```

```
BEGIN
   WRITELN('Ini Prosedur KESATU akan memanggil Prosedur
  KEDUA & KETIGA');
   kedua;
   ketiga;
END;

BEGIN
  CLRSCR;
  WRITELN('Ini program di modul utama akan memanggil
   Prosedur KESATU');
  kesatu;
  READLN;
END.
```

Hasil :

Ini program di modul utama akan memanggil Prosedur KESATU
Ini Prosedur KESATU akan memanggil Prosedur KEDUA & KETIGA
Prosedur KEDUA ini ada di dalam prosedur KESATU
Prosedur KETIGA ini juga ada di dalam prosedur KESATU

PROSEDUR MEMANGGIL DIRINYA SENDIRI

Prosedur memanggil dirinya sendiri merupakan suatu prosedur yang memanggil atau menggunakan prosedur itu juga. Proses dari suatu program yang memanggil dirinya sendiri dikenal dengan nama *recursion*. Proses ini membutuhkan lebih banyak memori dalam komputer.

```
Contoh :
PROGRAM rekursi_prosedur;
USES CRT;
VAR
  I : INTEGER;

PROCEDURE rekursi;
BEGIN
   I:=I+1;
   WRITELN('PASCAL');
   IF I < 10 THEN rekursi;
END;
```

```
BEGIN
  CLRSCR;
  I:=0; rekursi;
  READLN;
END.
```

Hasil :

```
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
```

LATIHAN

Buat program untuk menghitung gaji harian PT. XYZ dengan ketentuan:

- Gaji pokok karyawan Rp. 20000/jam
- Bila karyawan bekerja lebih dari 7 jam/hari maka kelebihannya dihitung lembur yang besarnya 1.5 dari gaji pokok
- Untuk karyawan yang bekerja 8 jam/hari atau lebih akan mendapat tambahan uang makan sebesar Rp. 5000
- Karyawan yang bekerja 9 jam/hari atau lebih akan mendapat uang transport lembur sebesar Rp. 4000

Program ini akan terdiri dari 4 buah prosedur : pokok, lembur, makan, jasa

Input : NIP, Nama, Jumlah jam kerja

Output :

LAPORAN GAJI HARIAN KARYAWAN
PT. XYZ

Bulan : April 2011

NIP	Nama	Gaji Pokok	Lembur	Uang makan	Transport Lembur

Sorting (1)

Muh Izzuddin Mahali, M.Cs.

PT. Elektronika FT UNY

PENDAHULUAN

- Pengurutan data dalam struktur data sangat penting, baik untuk data yang bertipe data numerik maupun karakter.
- Pengurutan dapat dilakukan secara ascending (naik) maupun descending (turun).
- Pengurutan adalah proses menyusun kembali data yang acak menjadi susunan yang teratur menurut aturan tertentu.

PENGERTIAN

- *Sorting* = pengurutan
- *Sorted* = terurut menurut kaidah/aturan tertentu
- Data pada umumnya disajikan dalam bentuk *sorted*.
- Contoh:
  - Data Mahasiswa
  - Kata-kata dalam kamus
  - File-file di dalam sebuah directory
  - Indeks sebuah buku
  - Data mutasi rekening tabungan
  - dll
- Bayangkan jika data di atas tidak terurut!

TUJUAN

- Mudah dalam Membaca data
- Mudah dalam menemukan data
- Penyajian data lebih teratur
- dll

ALGORITMA SORTING

Berbagai macam algoritma sorting

- Exchange Sort
- Selection Sort
- Insertion Sort
- Bubble Sort
- Quick Sort
- Shell Sort
- Binary Insertion Sort
- Dan masih banyak lagi

ALGORITMA SORTING

- Algoritma berdasarkan Priority Queue→Selection Sort & Heap Sort
- Algoritma penyisipan dalam keterurutan→Insertion Sort & Tree Sort
- Algoritma transposisi →Bubble Sort
- Algoritma increment →Shell Sort
- Algoritma dengan Divide & Conquer→Quick Sort & Merge Sort
- Algoritma-algoritma penghitungan alamat →Radix Sort & Proximity Map Sort

Bubble Sort

- Metode sorting termudah
- Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.
- Ascending :

  if elemen sekarang **>** dari elemen berikutnya then kedua elemen ditukar
- Descending:

  if elemen sekarang **<** dari elemen berikutnya then kedua elemen **ditukar**

Bubble Sort

- Membandingkan data ke-1 dengan data ke-2, jika data ke-1 lebih besar, maka kedua data ditukar.
- Kemudian membandingkan data ke-2 dengan data ke-3, jika data ke-2 lebih besar, kedua data ditukar lagi.
- Demikian seterusnya sampai data terakhir, sehingga data kedudukannya akan bergeser-geser.
- Untuk proses 2, pembandingan (pergeseran data) hanya sampai pada data terakhir dikurangi satu.

Bubble Sort

- Algoritma:
  banyaknya data: n
  Data diurutkan/disorting dari yang bernilai besar

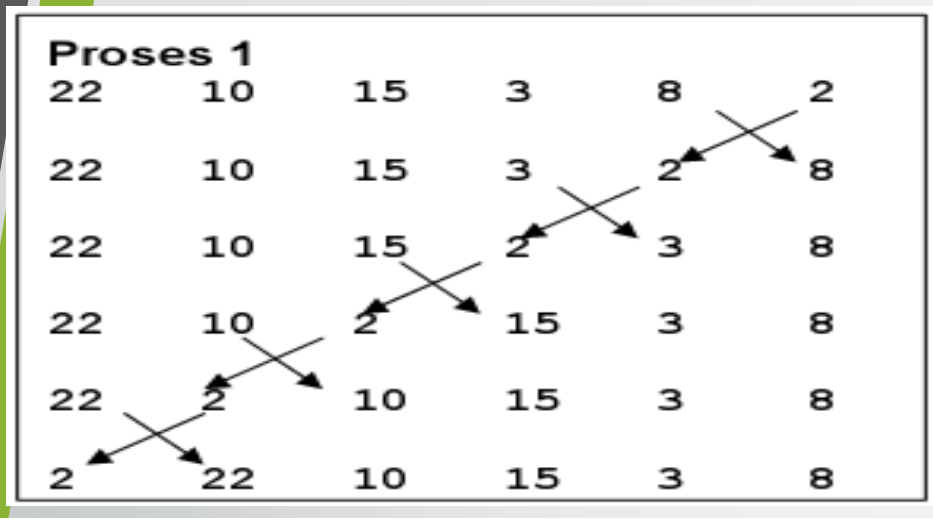
Proses

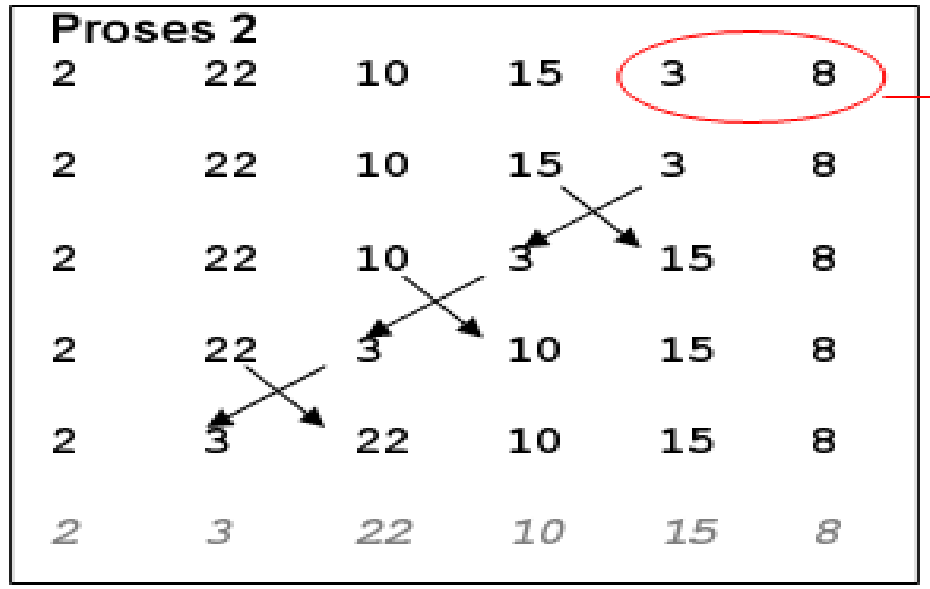
step 1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort





Tidak ada penukaran, karena 3 < 8

Pegurutan berhenti di sini!

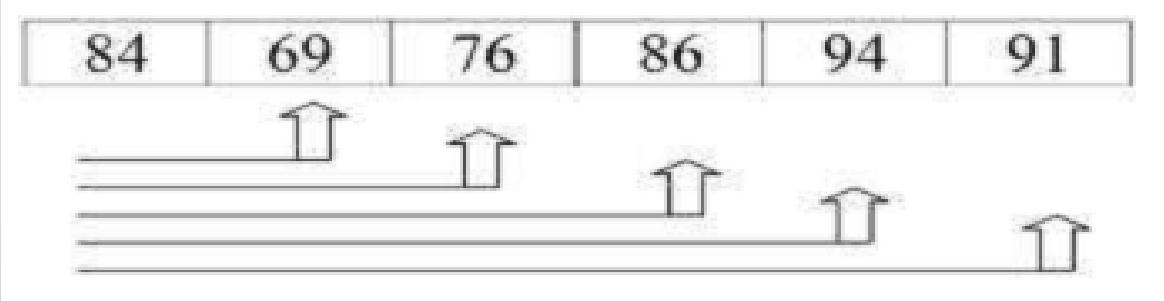
Exchange Sort

- Sangat mirip dengan Bubble Sort
- Banyak yang mengatakan Bubble Sort sama dengan Exchange Sort
- Pebedaan : dalam hal bagaimana membandingkan antar elemen-elemennya.
  - Exchange sort membandingkan **suatu elemen** dengan **elemen-elemen lainnya** dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen **pusat (pivot).**
  - Sedangkan Bubble sort akan membandingkan **elemen pertama/terakhir** dengan **elemen sebelumnya/sesudahnya**, kemudian elemen tersebut itu akan menjadi **pusat (pivot)** untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi, begitu seterusnya.

Exchange Sort

- Semua data dibandingkan dengan data pembanding, dimana pada akhir proses data terbesar akan berada pada akhir urutan.
- Pada proses 1: data ke-1 dibandingkan dengan data ke-2 **jika data ke-1 lebih besar** maka kedua data **langsung ditukar**. Kemudian data ke-1 dibandingkan lagi dengan data ke-3, lebih besar? Tukar! Demikian seterusnya.
- Pada proses 2: data ke-2 dibandingkan dengan data ke-3 jika data ke-2 lebih besar maka kedua data langsung ditukar. Kemudian data ke-2 dibandingkan lagi dengan data ke-4, lebih besar? Tukar! Demikian seterusnya.
- Dan seterusnya.....

Exchange Sort



Proses 1

Pivot (Pusat)

84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	**86**	94	91
86	69	76	**84**	**94**	91
94	69	76	84	**86**	91
94	69	76	84	86	91

Exchange Sort

Proses 2

Pivot (Pusat)

94	**69**	**76**	84	86	91
94	**76**	**69**	**84**	86	91
94	**84**	69	**76**	**86**	91
94	**86**	69	76	**84**	**91**
94	**91**	69	76	84	86

Proses 3

Pivot (Pusat)

94	91	**69**	**76**	84	86
94	91	**76**	**69**	**84**	86
94	91	**84**	69	**76**	**86**
94	91	**86**	69	76	84

Exchange Sort

Proses 4

Pivot (Pusat)

94	91	86	**69**	**76**	**84**
94	91	86	**76**	**69**	**84**
94	91	86	**84**	69	**76**

Proses 5

Pivot (Pusat)

94	91	86	84	**69**	**76**
94	91	86	84	**76**	**69**

SELECTION SORT

- Hampir sama dengan Exchange Sort, bedanya yang ditukar adalah indeknya. **Penukaran data dilakukan di akhir proses.**
- Pada proses 1: **variabel indek diberi nilai 1** (data ke-1) kemudian **data indek** dibandingkan dengan data ke-2 **jika data indek lebih besar** maka **nilai indek adalah 2** (data ke-2). Kemudian **data indek** dibandingkan lagi dengan data ke-3, lebih besar? Nilai indek ditukar! Demikian seterusnya.

  Setelah selesai, **nilai indek diperiksa apakah nilai indek berubah atau tidak**. **Jika nilai indek mengalami perubahan maka data ke-1 ditukar dengan data indek**.
- Pada proses 2: **variabel indek diberi nilai 2** (data ke-2) kemudian **data indek** dibandingkan dengan data ke-3 **jika data indek lebih besar** maka **nilai indek adalah 3** (data ke-3). Kemudian **data indek** dibandingkan lagi dengan data ke-4, lebih besar? Nilai indek ditukar! Demikian seterusnya.

  Setelah selesai, **nilai indek diperiksa apakah nilai indek berubah atau tidak**. **Jika nilai indek mengalami perubahan maka data ke-2 ditukar dengan data indek**.
- Dan seterusnya.....

SELECTION SORT

Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	Posisi
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (**tukar idx**)	4
21 < 40	4

Tukar data ke-0 (**32**) dengan data ke-4 (**21**)

0	1	2	3	4	5
21	75	69	58	32	40

Proses 2

0	1	2	3	4	5
21	**75**	69	58	32	40

Pembanding	Posisi
75 > 69 (**tukar idx**)	2
69 > 58 (**tukar idx**)	3
58 > 32 (**tukar idx**)	4
32 < 40	4

Tukar data ke-1 (**75**) dengan data ke-4 (**32**)

0	1	2	3	4	5
21	32	69	58	75	40

Proses 3

0	1	2	3	4	5
21	32	**69**	58	75	40

Pembanding	Posisi
69 > 58 (**tukar idx**)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (**69**) dengan data ke-5 (**40**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 4

0	1	2	3	4	5
21	32	40	**58**	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (**58**) dengan data ke-3 (**58**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 5

0	1	2	3	4	5
21	32	40	58	**75**	69

Pembanding	Posisi
75 > 69	5

Tukar data ke-4 (**75**) dengan data ke-5 (**69**)

0	1	2	3	4	5
21	32	40	58	69	75

Insertion Sort

- Mirip dengan cara orang **mengurutkan** kartu, selembar demi selembar kartu diambil dan **disisipkan** (insert) ke tempat yang seharusnya.
- Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang **lebih kecil**, maka akan ditempatkan (**diinsert**) diposisi yang seharusnya.
- Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang

Insertion Sort

Proses 1

0	1	2	3	4	5
22	**10**	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2

Proses 2

0	1	2	3	4	5
10	22	**15**	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	**15**	22	3	8	2

Proses 3

0	1	2	3	4	5
10	15	22	**3**	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2

Proses 4

0	1	2	3	4	5
3	10	15	22	**8**	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	**8**	10	15	22	2

Insertion Sort

Proses 5

0	1	2	3	4	5
3	8	10	15	22	**2**

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22

S E L E S A I

PT. Elektronika FT UNY

Muh. Izzuddin Mahali, M.Cs.

Sorting (2)

Muh Izzuddin Mahali, M.Cs.

PENDAHULUAN

- Pengurutan data dalam struktur data sangat penting, baik untuk data yang bertipe data numerik maupun karakter.
- Pengurutan dapat dilakukan secara ascending (naik) maupun descending (turun).
- Pengurutan adalah proses menyusun kembali data yang acak menjadi susunan yang teratur menurut aturan tertentu.

PENGERTIAN

- *Sorting* = pengurutan
- *Sorted* = terurut menurut kaidah/aturan tertentu
- Data pada umumnya disajikan dalam bentuk *sorted*.
- Contoh:
  - Data Mahasiswa
  - Kata-kata dalam kamus
  - File-file di dalam sebuah directory
  - Indeks sebuah buku
  - Data mutasi rekening tabungan
  - dll
- Bayangkan jika data di atas tidak terurut!

TUJUAN

- Mudah dalam Membaca data
- Mudah dalam menemukan data
- Penyajian data lebih teratur
- dll

ALGORITMA SORTING

Berbagai macam algoritma sorting

- Exchange Sort
- Selection Sort
- Insertion Sort
- Bubble Sort
- Quick Sort
- Shell Sort
- Binary Insertion Sort
- Dan masih banyak lagi

ALGORITMA SORTING

- Algoritma berdasarkan Priority Queue→Selection Sort & Heap Sort
- Algoritma penyisipan dalam keterurutan→Insertion Sort & Tree Sort
- Algoritma transposisi →Bubble Sort
- Algoritma increment →Shell Sort
- Algoritma dengan Divide & Conquer→Quick Sort & Merge Sort
- Algoritma-algoritma penghitungan alamat →Radix Sort & Proximity Map Sort

Bubble Sort

- Metode sorting termudah
- Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.
- Ascending :

  if elemen sekarang **>** dari elemen berikutnya then kedua elemen ditukar
- Descending:

  if elemen sekarang **<** dari elemen berikutnya then kedua elemen **ditukar**

Bubble Sort

- Membandingkan data ke-1 dengan data ke-2, jika data ke-1 lebih besar, maka kedua data ditukar.
- Kemudian membandingkan data ke-2 dengan data ke-3, jika data ke-2 lebih besar, kedua data ditukar lagi.
- Demikian seterusnya sampai data terakhir, sehingga data kedudukannya akan bergeser-geser.
- Untuk proses 2, pembandingan (pergeseran data) hanya sampai pada data terakhir dikurangi satu.

Bubble Sort

- Algoritma:
  banyaknya data: n
  Data diurutkan/disorting dari yang bernilai besar

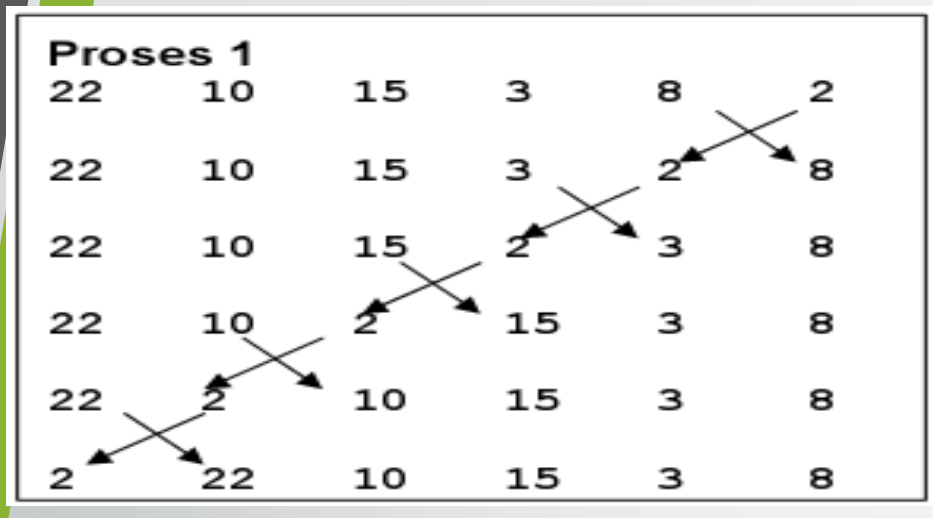
Proses

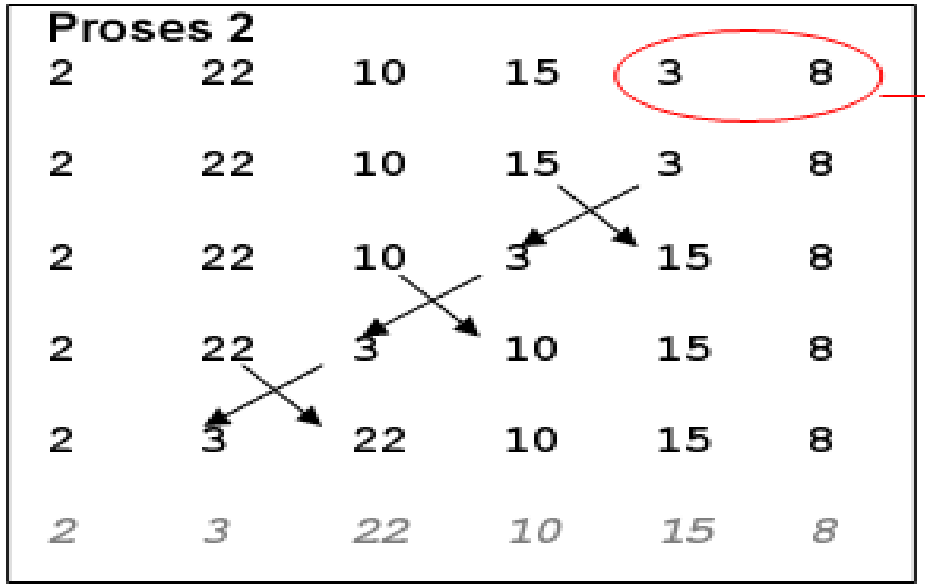
step 1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort





Tidak ada penukaran, karena 3 < 8

Pegurutan berhenti di sini!

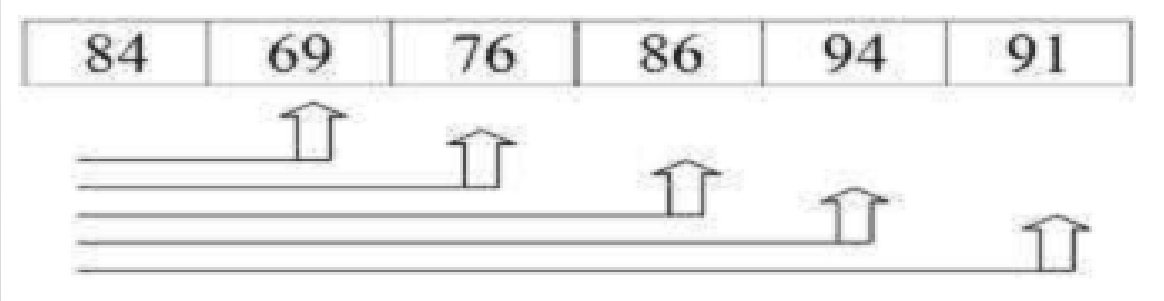
Exchange Sort

- Sangat mirip dengan Bubble Sort
- Banyak yang mengatakan Bubble Sort sama dengan Exchange Sort
- Pebedaan : dalam hal bagaimana membandingkan antar elemen-elemennya.
  - Exchange sort membandingkan **suatu elemen** dengan **elemen-elemen lainnya** dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen **pusat (pivot).**
  - Sedangkan Bubble sort akan membandingkan **elemen pertama/terakhir** dengan **elemen sebelumnya/sesudahnya**, kemudian elemen tersebut itu akan menjadi **pusat (pivot)** untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi, begitu seterusnya.

Exchange Sort

- Semua data dibandingkan dengan data pembanding, dimana pada akhir proses data terbesar akan berada pada akhir urutan.
- Pada proses 1: data ke-1 dibandingkan dengan data ke-2 **jika data ke-1 lebih besar** maka kedua data **langsung ditukar**. Kemudian data ke-1 dibandingkan lagi dengan data ke-3, lebih besar? Tukar! Demikian seterusnya.
- Pada proses 2: data ke-2 dibandingkan dengan data ke-3 jika data ke-2 lebih besar maka kedua data langsung ditukar. Kemudian data ke-2 dibandingkan lagi dengan data ke-4, lebih besar? Tukar! Demikian seterusnya.
- Dan seterusnya.....

Exchange Sort



Proses 1

Pivot (Pusat)

84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	**86**	94	91
86	69	76	**84**	**94**	91
94	69	76	84	**86**	91
94	69	76	84	86	91

Exchange Sort

Proses 2

Pivot (Pusat)

94	**69**	**76**	84	86	91
94	**76**	**69**	**84**	86	91
94	**84**	69	**76**	**86**	91
94	**86**	69	76	**84**	**91**
94	**91**	69	76	84	86

Proses 3

Pivot (Pusat)

94	91	**69**	**76**	84	86
94	91	**76**	**69**	**84**	86
94	91	**84**	69	**76**	**86**
94	91	**86**	69	76	84

Exchange Sort

Proses 4

Pivot (Pusat)

94	91	86	**69**	**76**	**84**
94	91	86	**76**	**69**	**84**
94	91	86	**84**	69	**76**

Proses 5

Pivot (Pusat)

94	91	86	84	**69**	**76**
94	91	86	84	**76**	**69**

SELECTION SORT

- Hampir sama dengan Exchange Sort, bedanya yang ditukar adalah indeknya. **Penukaran data dilakukan di akhir proses.**
- Pada proses 1: **variabel indek diberi nilai 1** (data ke-1) kemudian **data indek** dibandingkan dengan data ke-2 **jika data indek lebih besar** maka **nilai indek adalah 2** (data ke-2). Kemudian **data indek** dibandingkan lagi dengan data ke-3, lebih besar? Nilai indek ditukar! Demikian seterusnya.

  Setelah selesai, **nilai indek diperiksa apakah nilai indek berubah atau tidak**. **Jika nilai indek mengalami perubahan maka data ke-1 ditukar dengan data indek**.
- Pada proses 2: **variabel indek diberi nilai 2** (data ke-2) kemudian **data indek** dibandingkan dengan data ke-3 **jika data indek lebih besar** maka **nilai indek adalah 3** (data ke-3). Kemudian **data indek** dibandingkan lagi dengan data ke-4, lebih besar? Nilai indek ditukar! Demikian seterusnya.

  Setelah selesai, **nilai indek diperiksa apakah nilai indek berubah atau tidak**. **Jika nilai indek mengalami perubahan maka data ke-2 ditukar dengan data indek**.
- Dan seterusnya.....

SELECTION SORT

Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	Posisi
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (**tukar idx**)	4
21 < 40	4

Tukar data ke-0 (**32**) dengan data ke-4 (**21**)

0	1	2	3	4	5
21	75	69	58	32	40

Proses 2

0	1	2	3	4	5
21	**75**	69	58	32	40

Pembanding	Posisi
75 > 69 (**tukar idx**)	2
69 > 58 (**tukar idx**)	3
58 > 32 (**tukar idx**)	4
32 < 40	4

Tukar data ke-1 (**75**) dengan data ke-4 (**32**)

0	1	2	3	4	5
21	32	69	58	75	40

Proses 3

0	1	2	3	4	5
21	32	**69**	58	75	40

Pembanding	Posisi
69 > 58 (**tukar idx**)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (**69**) dengan data ke-5 (**40**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 4

0	1	2	3	4	5
21	32	40	**58**	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (**58**) dengan data ke-3 (**58**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 5

0	1	2	3	4	5
21	32	40	58	**75**	69

Pembanding	Posisi
75 > 69	5

Tukar data ke-4 (**75**) dengan data ke-5 (**69**)

0	1	2	3	4	5
21	32	40	58	69	75

Insertion Sort

- Mirip dengan cara orang **mengurutkan** kartu, selembar demi selembar kartu diambil dan **disisipkan** (insert) ke tempat yang seharusnya.
- Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang **lebih kecil**, maka akan ditempatkan (**diinsert**) diposisi yang seharusnya.
- Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang

Insertion Sort

Proses 1

0	1	2	3	4	5
22	**10**	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2

Proses 2

0	1	2	3	4	5
10	22	**15**	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	**15**	22	3	8	2

Proses 3

0	1	2	3	4	5
10	15	22	**3**	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2

Proses 4

0	1	2	3	4	5
3	10	15	22	**8**	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	**8**	10	15	22	2

Insertion Sort

Proses 5

0	1	2	3	4	5
3	8	10	15	22	**2**

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22

Shell Sort

- Penemu : Donald Shell
- Metode perbandingan dan pertukaran
- Perbandingan dimulai dari separuh array yang akan disortir dengan separuh bagian yang lain.
- Contoh :
  - Jika terdapat 100 elemen, diperbandingkan elemen 1 dan elemen 51, elemen 2 dan elemen 52 dst. Selanjutnya algoritma akan membandingkan elemen 1 dan elemen 26, elemen 2 dan elemen 27 dst.

Penjelasan algoritma

- Program akan dijalankan jika `range<>0` terpenuhi
- Sebelum masuk putaran ditentukan range dan target
- Pada putaran ke-`1`, `range = banyak/2`
- Tiap putaran dimulai dari `counter=1` sampai dengan `counter=target`

Penjelasan algoritma

- Pada tiap counter dilakukan proses : `kiri = counter` dan selanjutnya,
- `item(kiri)` dibandingkan dengan `item(kanan)` dimana : `kanan = kiri + range`
- Jika `item(kiri) >= item(kanan)` maka proses selesai dan dilanjutkan counter atau mungkin putaran berikutnya

Penjelasan algoritma

- Jika `item(kiri) < item(kanan)` maka terjadi pertukaran, selanjutnya :
- jika `item kiri < range` maka proses selesai dan dilanjutkan counter berikutnya
- jika `kiri > range` maka `kiri = kiri - range` dan proses dimulai dari awal pembandingan `item(kiri)` dan `item(kanan)` lagi
- Jika semua `counter` pada suatu putaran telah selesai maka range akan dihitung kembali yaitu : `range = range/2`.
- Jika `range <> 0` maka program akan dijalankan sampai `range = 0` berarti data telah terurut

Shell Sort

Elemen ke	1	2	3	4	5	6	7	8	9	10
Nilai awal	14	6	23	18	7	47	2	83	16	38
Untuk Range 5										
Setelah putaran ke-1	47	6	23	18	7	14	2	83	16	38
Setelah putaran ke-2	47	6	23	18	7	14	2	83	16	38
Setelah putaran ke-3	47	6	83	18	7	14	2	23	16	38
Setelah putaran ke-4	47	6	83	18	7	14	2	23	16	38
Setelah putaran ke-5	47	6	83	18	38	14	2	23	16	7
Untuk Range 2										
Setelah putaran ke-6	83	6	47	18	38	14	2	23	16	7
Setelah putaran ke-7	83	18	47	6	38	14	2	23	16	7
Setelah putaran ke-8	83	18	47	6	38	14	2	23	16	7
Setelah putaran ke-9	83	18	47	14	38	6	2	23	16	7
Setelah putaran ke-10	83	18	47	14	38	6	2	23	16	7
Setelah putaran ke-11	83	23	47	18	38	14	2	6	16	7
Setelah putaran ke-12	83	23	47	18	38	14	16	6	2	7
Setelah putaran ke-13	83	23	47	18	38	14	16	7	2	6
Untuk Range 1										
Setelah putaran ke-14	83	23	47	18	38	14	16	7	2	6
Setelah putaran ke-15	83	47	23	18	38	14	16	7	2	6
Setelah putaran ke-16	83	47	23	18	38	14	16	7	2	6
Setelah putaran ke-17	83	47	38	23	18	14	16	7	2	6
Setelah putaran ke-18	83	47	38	23	18	14	16	7	2	6
Setelah putaran ke-19	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-20	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-21	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-22	83	47	38	23	18	16	14	7	6	2

S E L E S A I

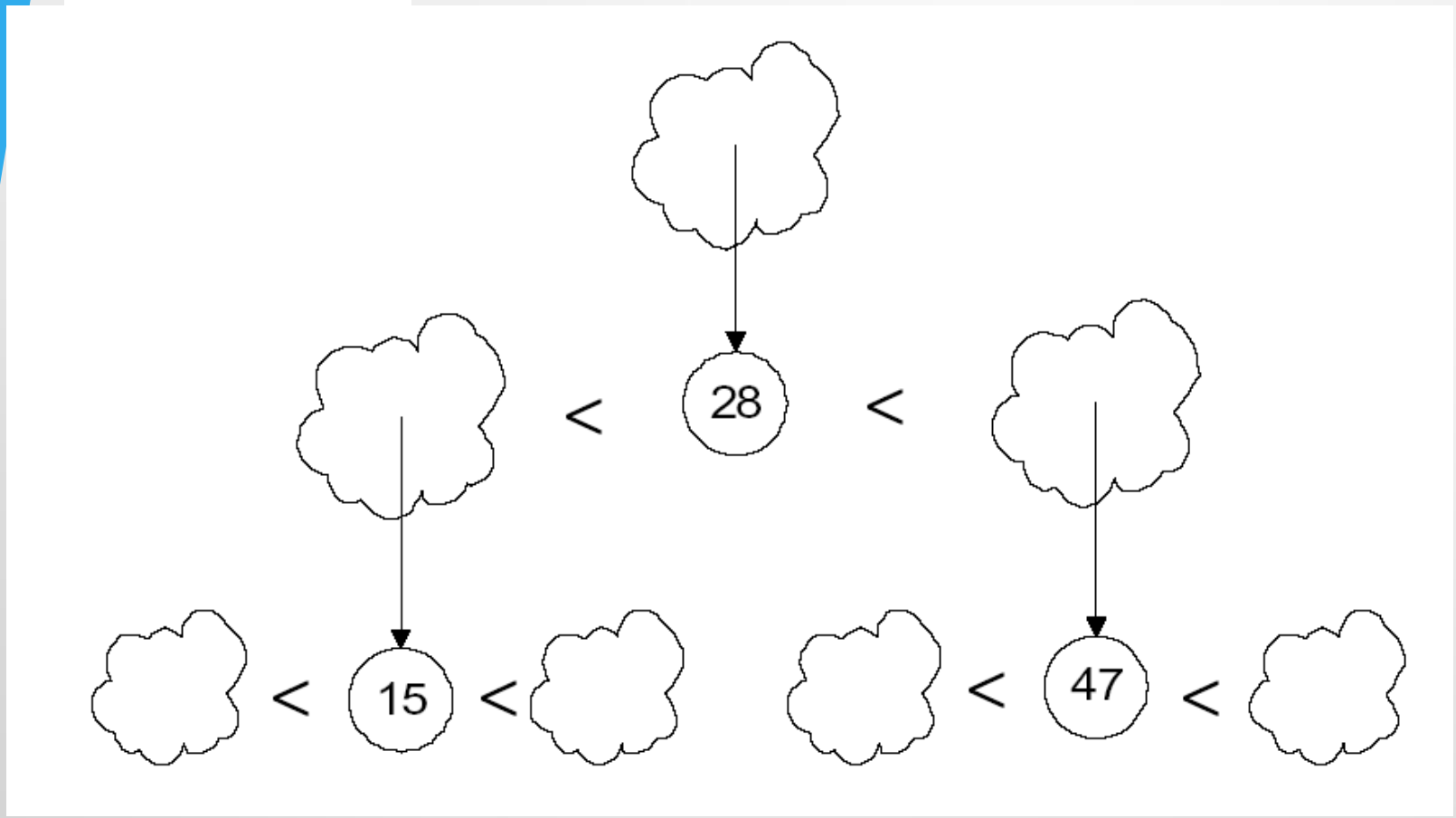
PT. Elektronika FT UNY

Muh. Izzuddin Mahali, M.Cs.

Quick Sort dan Merge Sort

Muh Izzuddin Mahali, M.Cs.

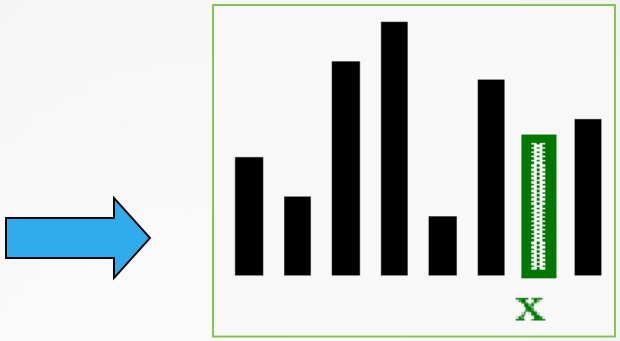
Ide Quicksort



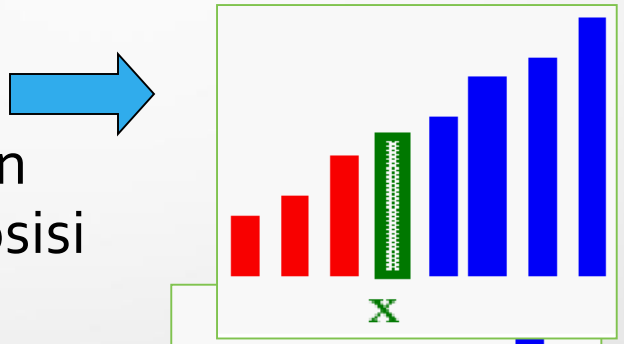
Tentukan “pivot”. Bagi Data menjadi 2 Bagian yaitu Data kurang dari dan Data lebih besar dari pivot. Urutkan tiap bagian tersebut secara rekursif.

Ide Quicksort

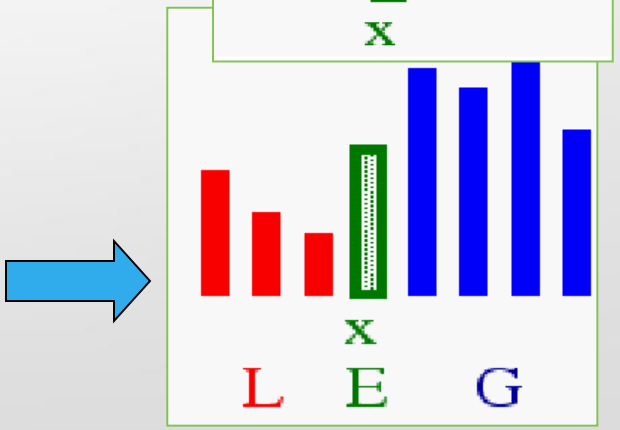
1. **Tentukan pivotnya**



2. **Divide (Bagi):** Data disusun sehingga x berada pada posisi akhir E



3. **Recur and Conquer**: Diurutkan secara rekursif



Quicksort

- Algoritma divide-and-conquer
  - array A[p..r] is *dipartisi* menjadi dua subarray yang tidak empty A[p..q] and A[q+1..r]
    - Invariant: Semua elemen pada A[p..q] lebih kecil dari semua elemen pada A[q+1..r]
  - Subarray diurutkan secara rekursif dengan memanggil quicksort

Partisi

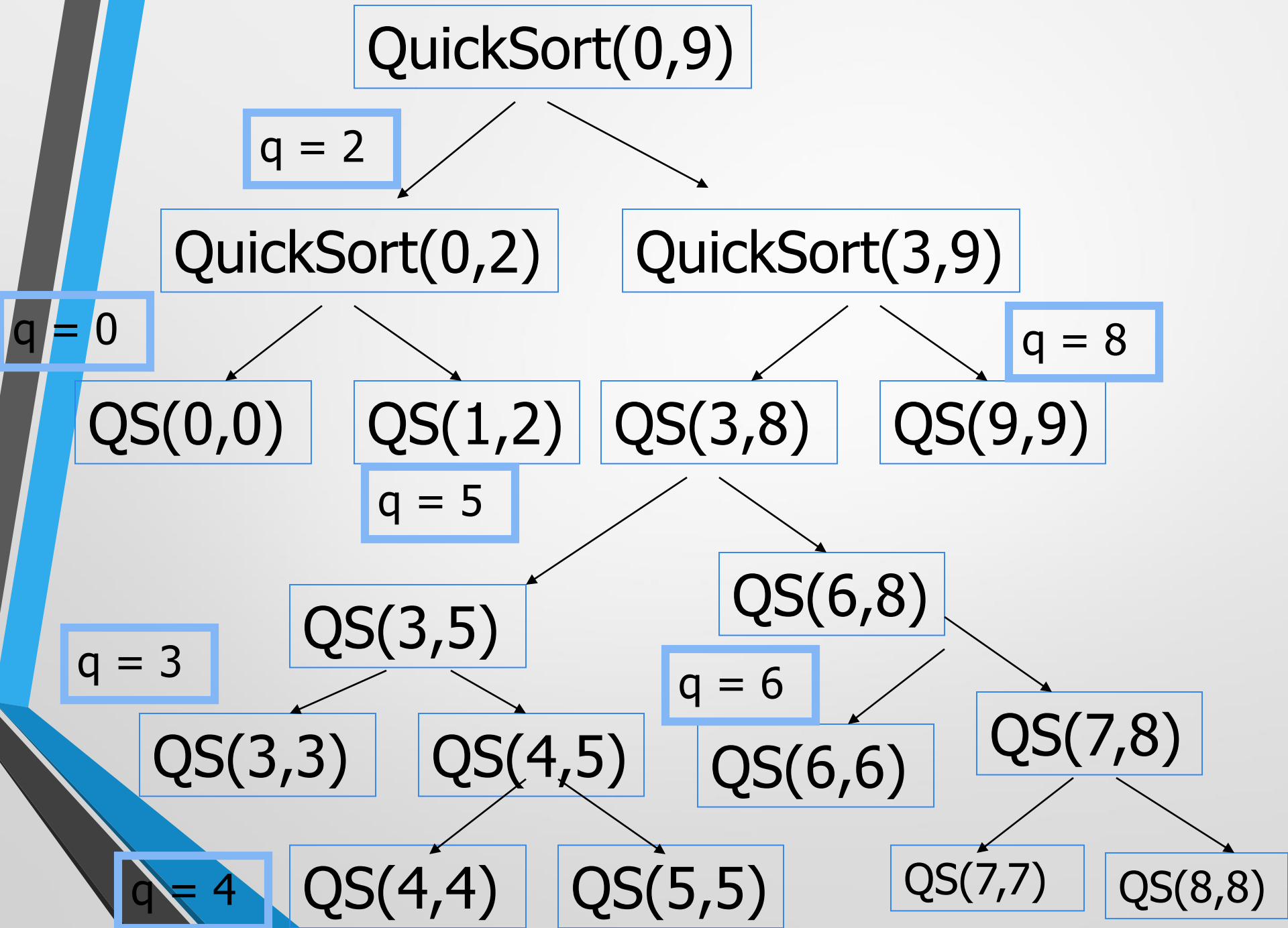
- Jelas, semua kegiatan penting berada pada fungsi **`partition()`**
  - Menyusun kembali subarray
  - Hasil akhir :
    - Dua subarray
    - Semua elemen pada subarray pertama ≤ semua nilai pada subarray kedua
  - Mengembalikan indeks pivot yang membagi subarray

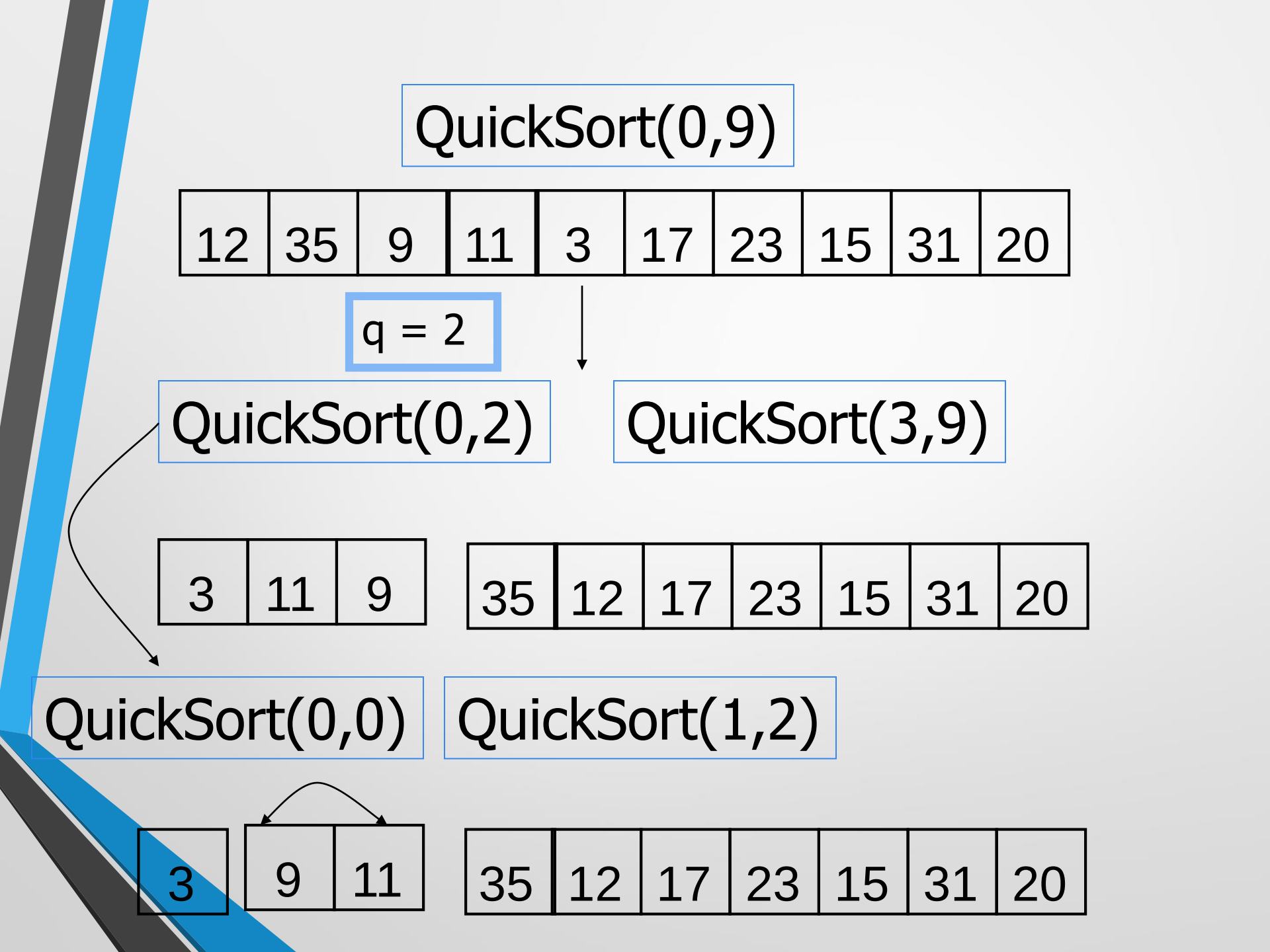
Partisi

- Partition(A, p, r):
  - Pilih elemen sebagai "pivot" (*which?*)
  - Dua bagian A[p..i] and A[j..r]
    - Semua element pada A[p..i] <= pivot
    - Semua element pada A[j..r] >= pivot
  - Increment i sampai A[i] >= pivot
  - Decrement j sampai A[j] <= pivot
  - Swap A[i] dan A[j]
  - Repeat Until i >= j
  - Return j

Quicksort

12	35	9	11	3	17	23	15	31	20


QuickSort(0,9)
q = 2
QuickSort(0,2)
QuickSort(3,9)
q = 0
q = 8
QS(0,0)
QS(1,2)
QS(3,8)
QS(9,9)
q = 5
QS(3,5)
QS(6,8)
q = 3
q = 6
QS(3,3)
QS(4,5)
QS(6,6)
QS(7,8)
q = 4
QS(4,4)
QS(5,5)
QS(7,7)
QS(8,8)


QuickSort(0,9)
12 35 9 11 3 17 23 15 31 20
q = 2
QuickSort(0,2)
QuickSort(3,9)
3 11 9
35 12 17 23 15 31 20
QuickSort(0,0)
QuickSort(1,2)
3
9 11
35 12 17 23 15 31 20

12	35	9	11	3	17	23	15	31	20

QuickSort(0,9)

- X = PIVOT merupakan indeks ke –0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih besar dari PIVOT. Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari PIVOT. Cara kerjanya : selama Data[j] > PIVOT maka nilai j dikurangi

q = Partition(0,9)

12	35	9	11	3	17	23	15	31	20

PIVOT = 12

i = 0 j = 4

i < j maka SWAP

SWAP

3	35	9	11	12	17	23	15	31	20

PIVOT = 12

i = 1 j = 3

i < j maka SWAP

SWAP

3	11	9	35	12	17	23	15	31	20

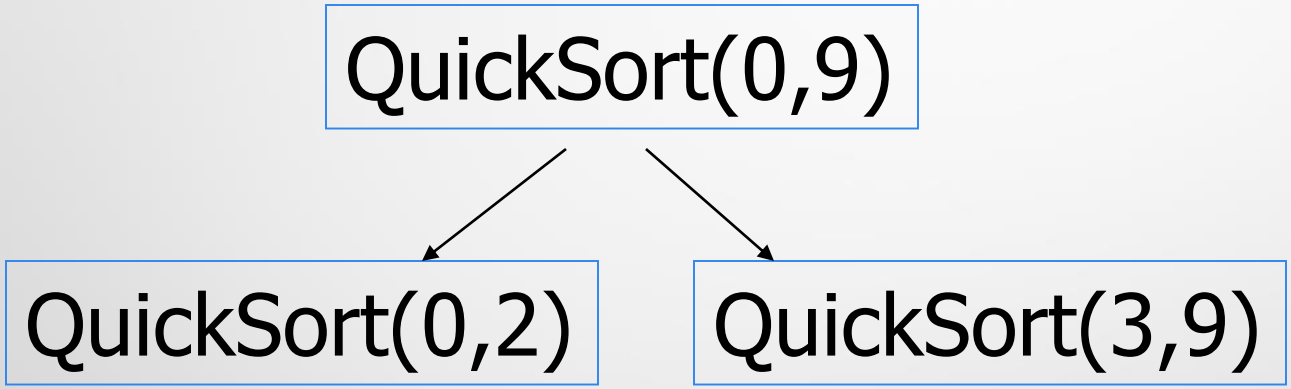
PIVOT = 12

i = 3 j = 2

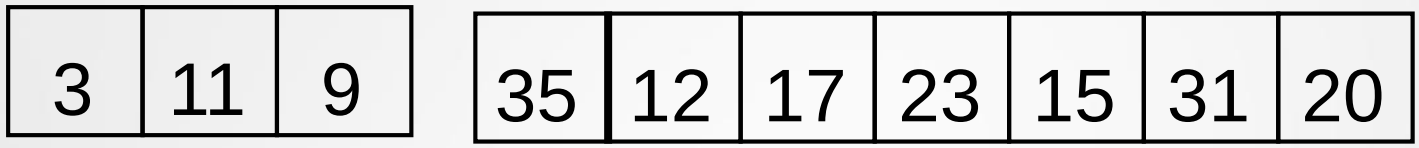
i < j (False) NO SWAP

Return j = 2

Q = Partisi = 2



QuickSort(0,2)



PIVOT = 3

i = 0 j = 0

i < j (False) NO SWAP

Return j = 0

Q = Partisi = 0

QuickSort(0,0)

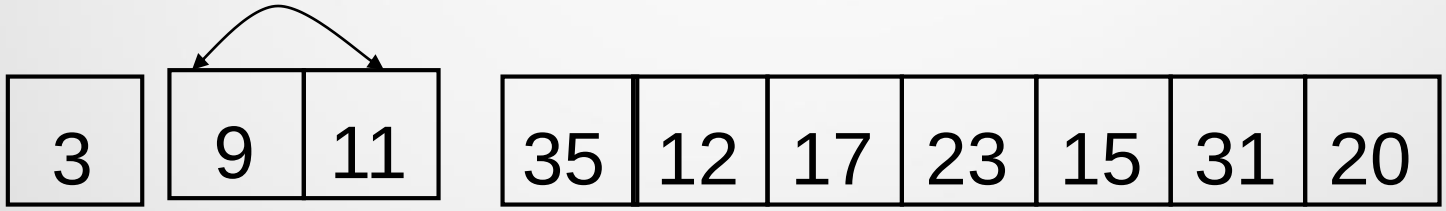
QuickSort(1,2)

QuickSort(1,2)

PIVOT = 11

i = 1 j = 2

i < j SWAP



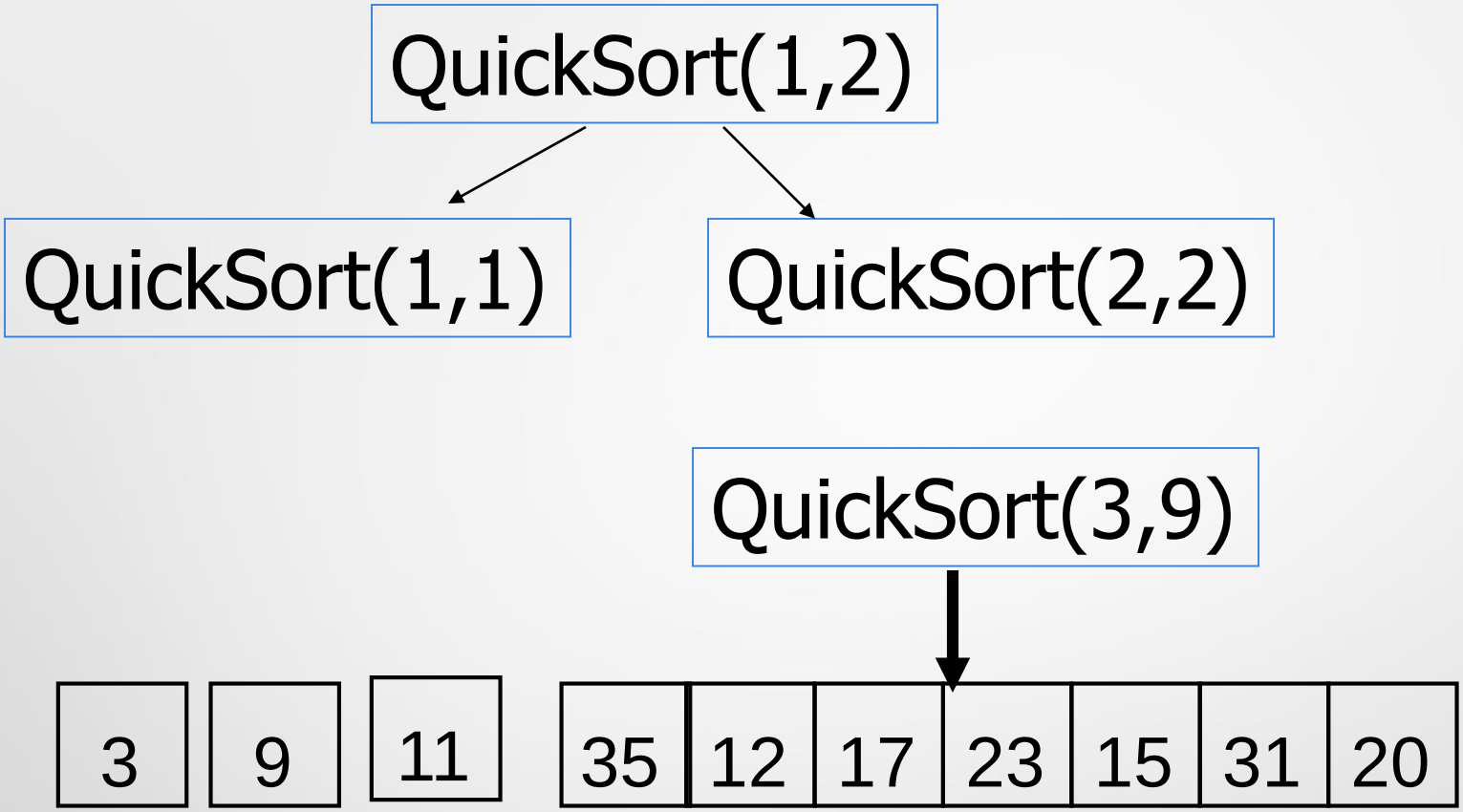
PIVOT = 11

i = 2 j = 1

i < j NO SWAP

Return j = 1

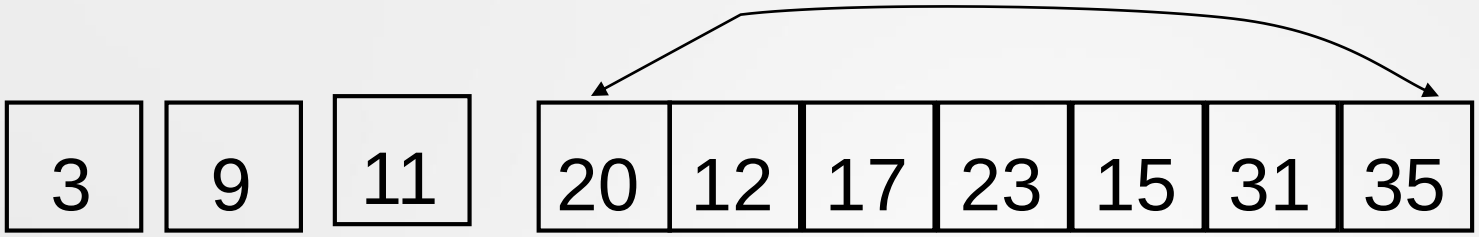
Q = Partisi = 1



PIVOT = 35

i = 3 j = 9

i < j SWAP



PIVOT = 35

i = 9 j = 8

i < j NO SWAP

Return j = 8

Q = Partisi = 8

QuickSort(3,9)

QuickSort(3,8)

QuickSort(9,9)

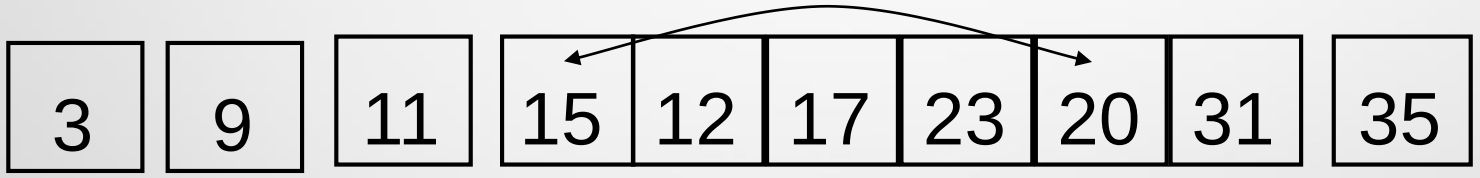
3	9	11	20	12	17	23	15	31	35

QuickSort(3,8)

PIVOT = 20

i = 3 j = 7

i < j SWAP



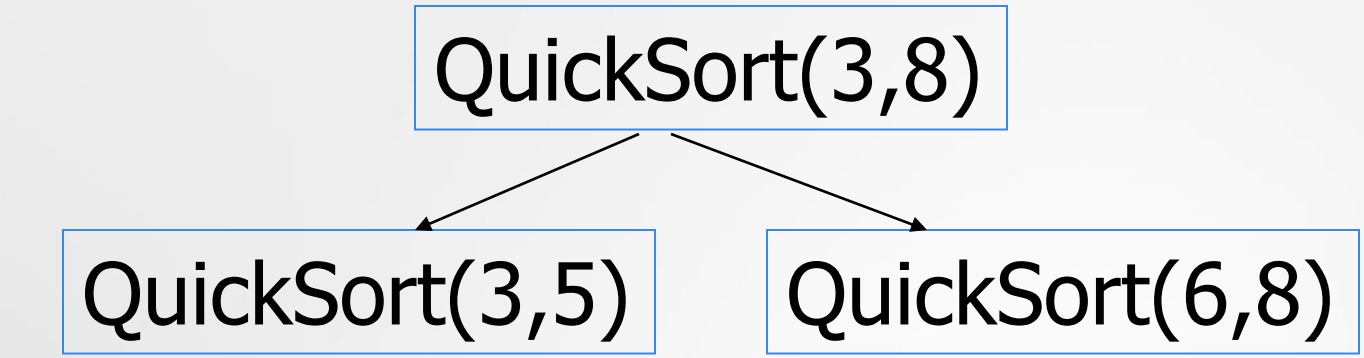
PIVOT = 20

i = 6 j = 5

i < j NO SWAP

Return j = 5

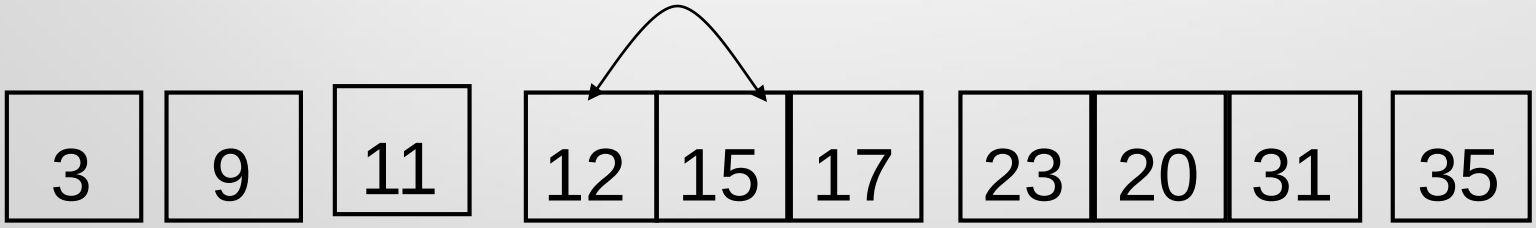
Q = Partisi = 5



PIVOT = 15

i = 3 j = 4

i < j SWAP



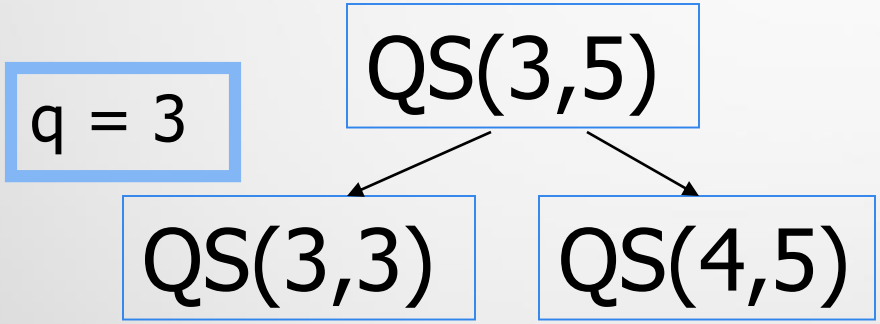
PIVOT = 15

i = 4 j = 3

i < j NO SWAP

Return j = 3

Q = Partisi = 3



3	9	11	12	15	17	23	20	31	35

QS(4,5)

PIVOT = 15

i = 4 j = 4

i < j NO SWAP

Return j = 4

Q = Partisi = 4

q = 4

QS(4,5)

QS(4,4) QS(5,5)

3	9	11	12	15	17	23	20	31	35

QuickSort(6,8)

PIVOT = 23

i = 6 j = 7

i < j SWAP

3	9	11	12	15	17	20	23	31	35

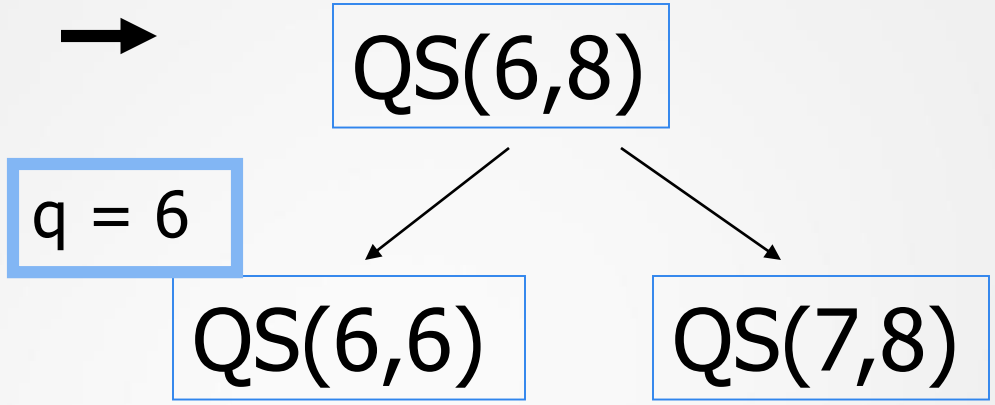
PIVOT = 23

i = 7 j = 6

i < j NO SWAP

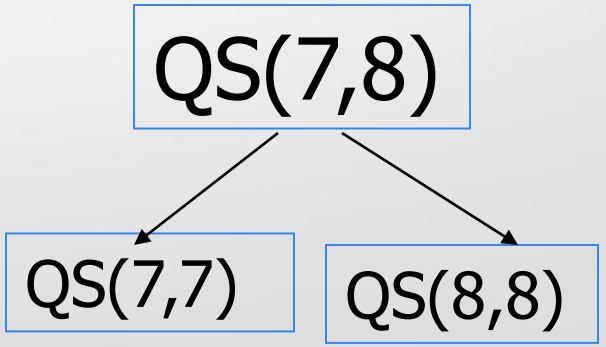
Return j = 6

Q = Partisi = 6



3	9	11	12	15	17	20	23	31	35

QS(7,8)



PIVOT = 23

i = 7 j = 7

i < j NO SWAP

Return j = 7

Q = Partisi = 7

Analisa Quicksort

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?

Analisa Quicksort

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ?

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ?$O(\log_2 n)$

Quicksort Analysis

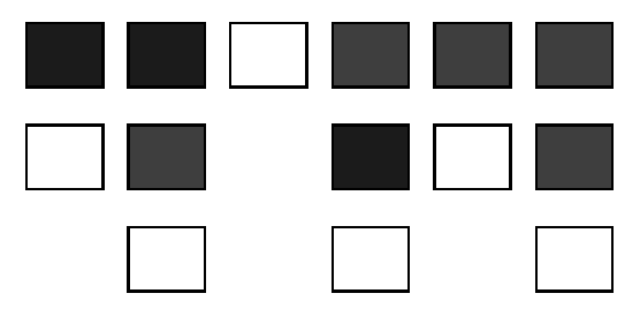
- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ? $O(\log_2 n)$
  - Jumlah pengaksesan partisi ?

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ? $O(\log_2 n)$
  - Jumlah pengaksesan partisi ? O(n)

Quicksort Analysis

- Diasumsikan bahwa pivot dipilih secara random
- **Running Time untuk Best case : $O(n \log_2 n)$**
  - Pivot selalu berada ditengah elemen
  - Tiap pemanggilan rekursif array dibagi menjadi dua subaarray dengan ukuran yang sama, Bagian sebelah kiri pivot : elemennya lebih kecil dari pivot, Bagian sebelah kanan pivot : elemennya lebih besar dari pivot



Quicksort Analysis

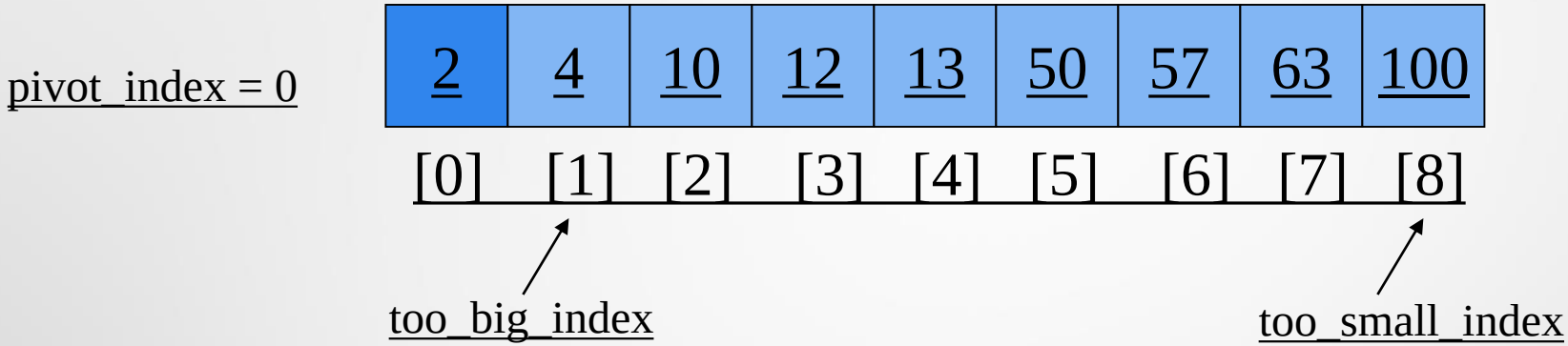
- Diasumsikan bahwa pivot dipilih secara random
- **Running Time untuk Best case : $O(n \log_2 n)$**
- Berapa running time untuk Worst case?

Quicksort Analysis

Worst case: $O(N^2)$

- Pivot merupakan elemen terbesar atau terkecil pada tiap pemanggilan rekursif, sehingga menjadi suatu bagian yang lebih kecil pivot, pivot dan bagian yang kosong

Quicksort: Worst Case



- Dimisalkan elemen pertama dipilih sebagai pivot
- Assume first element is chosen as pivot.
- Misalkan terdapat array yang sudah urut

Kesimpulan Algoritma Sorting

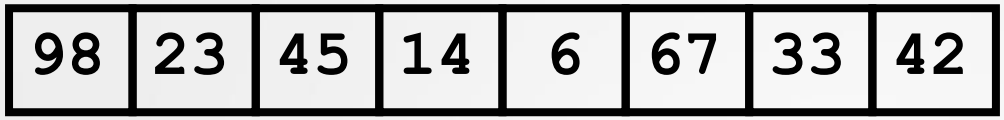
Algorithm	Time	Notes
selection-sort	$O(n^2)$	▪ in-place ▪ lambat (baik untuk input kecil)
insertion-sort	$O(n^2)$	▪ in-place ▪ lambat (baik untuk input kecil)
quick-sort	$O(n \log_2 n)$ expected	▪ in-place, randomized ▪ paling cepat (Bagus untuk data besar)
merge-sort	$O(n \log_2 n)$	▪ sequential data access ▪ cepat (Bagus untuk data besar)

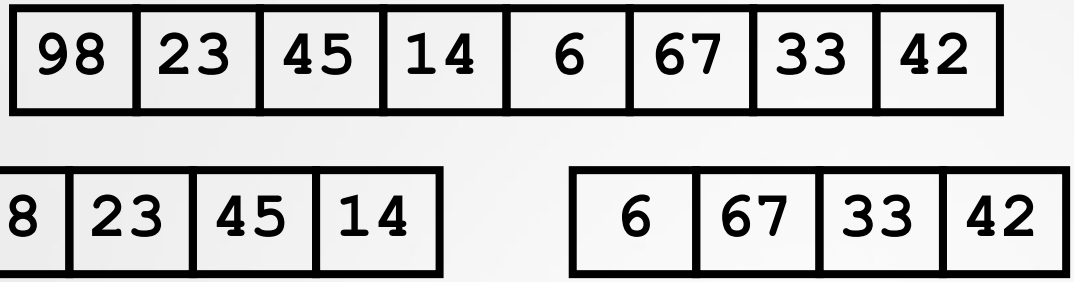
Sorting Algorithms – Merge Sort

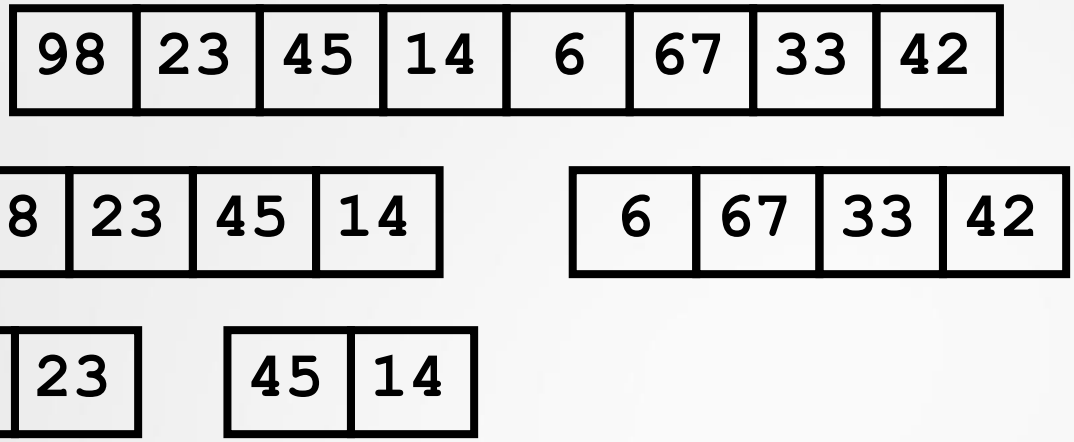
Mergesort

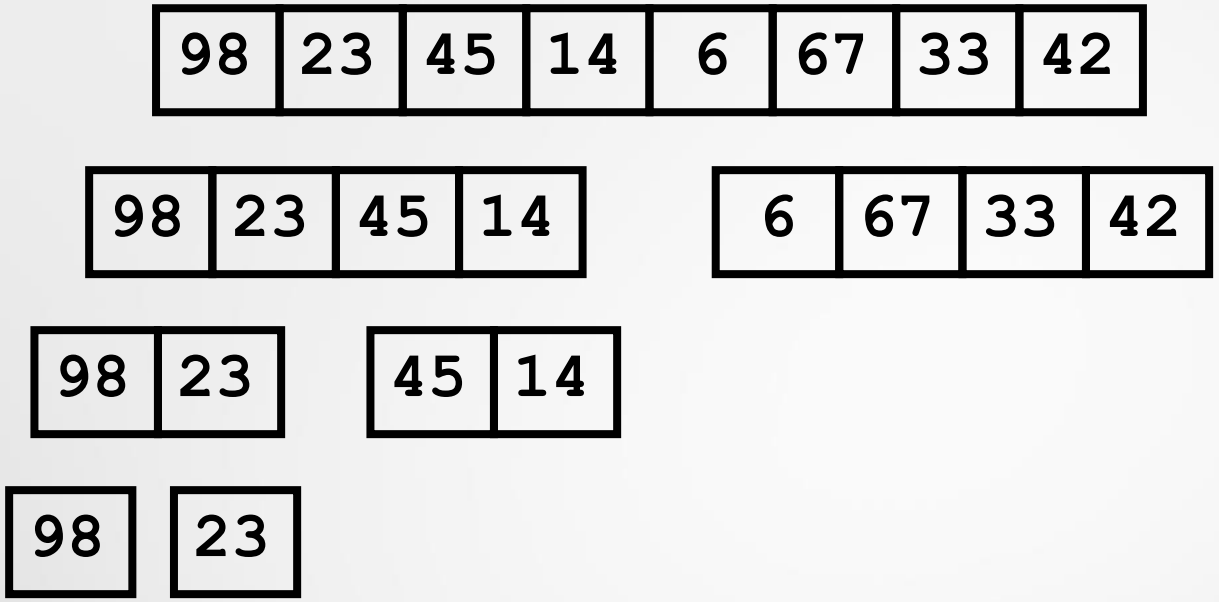
- **Merupakan algoritma divide-and-conquer (membagi dan menyelesaikan)**
- **Membagi array menjadi dua bagian sampai subarray hanya berisi satu elemen**
- **Mengabungkan solusi sub-problem :**
  - **Membandingkan elemen pertama subarray**
  - **Memindahkan elemen terkecil dan meletakkannya ke array hasil**
  - **Lanjutkan Proses sampai semua elemen berada pada array hasil**

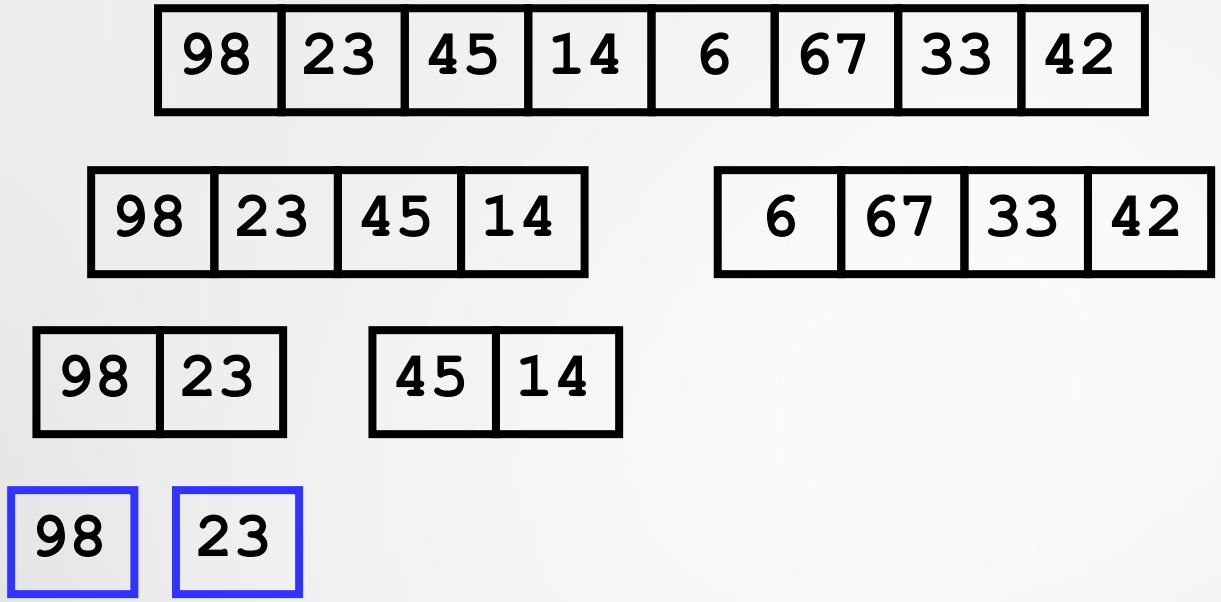
37	23	6	89	15	12	2	19

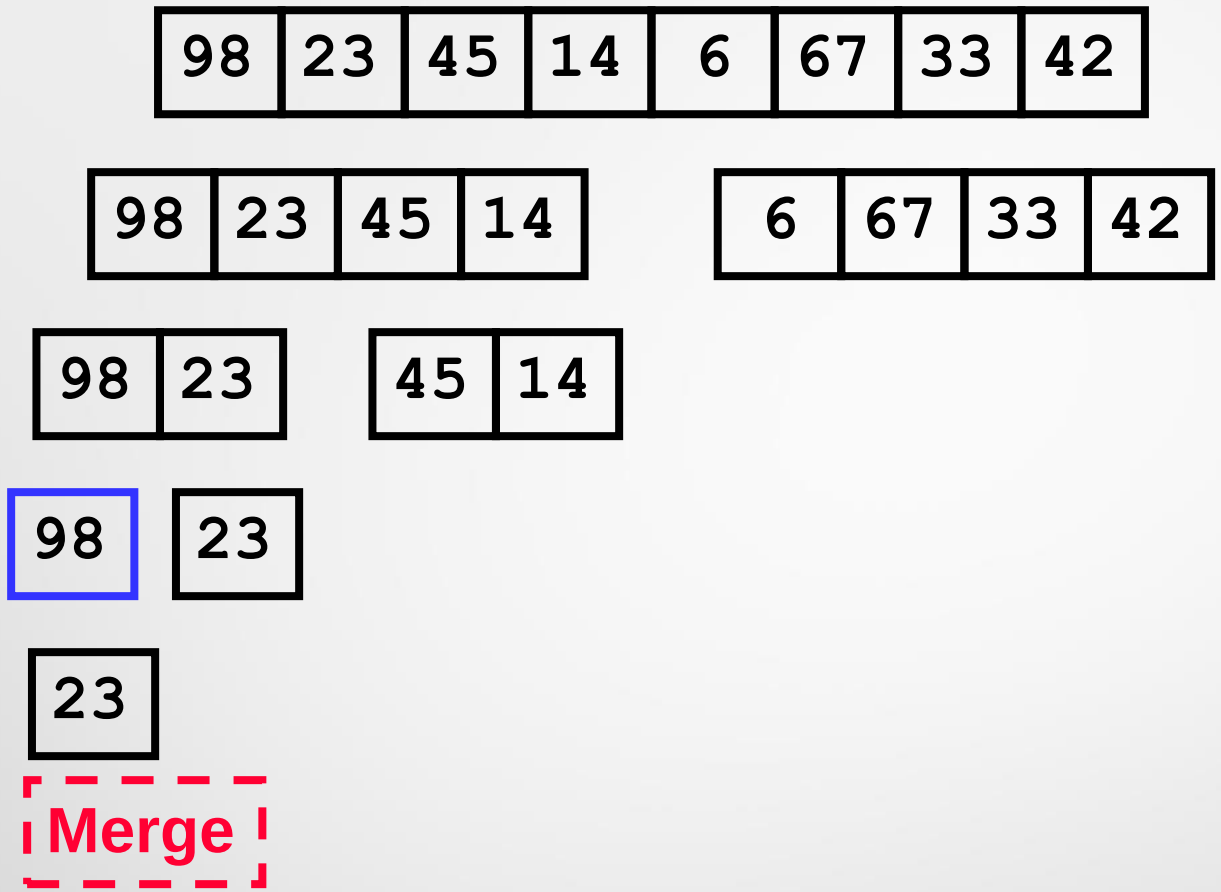

98
23
45
14
6
67
33
42

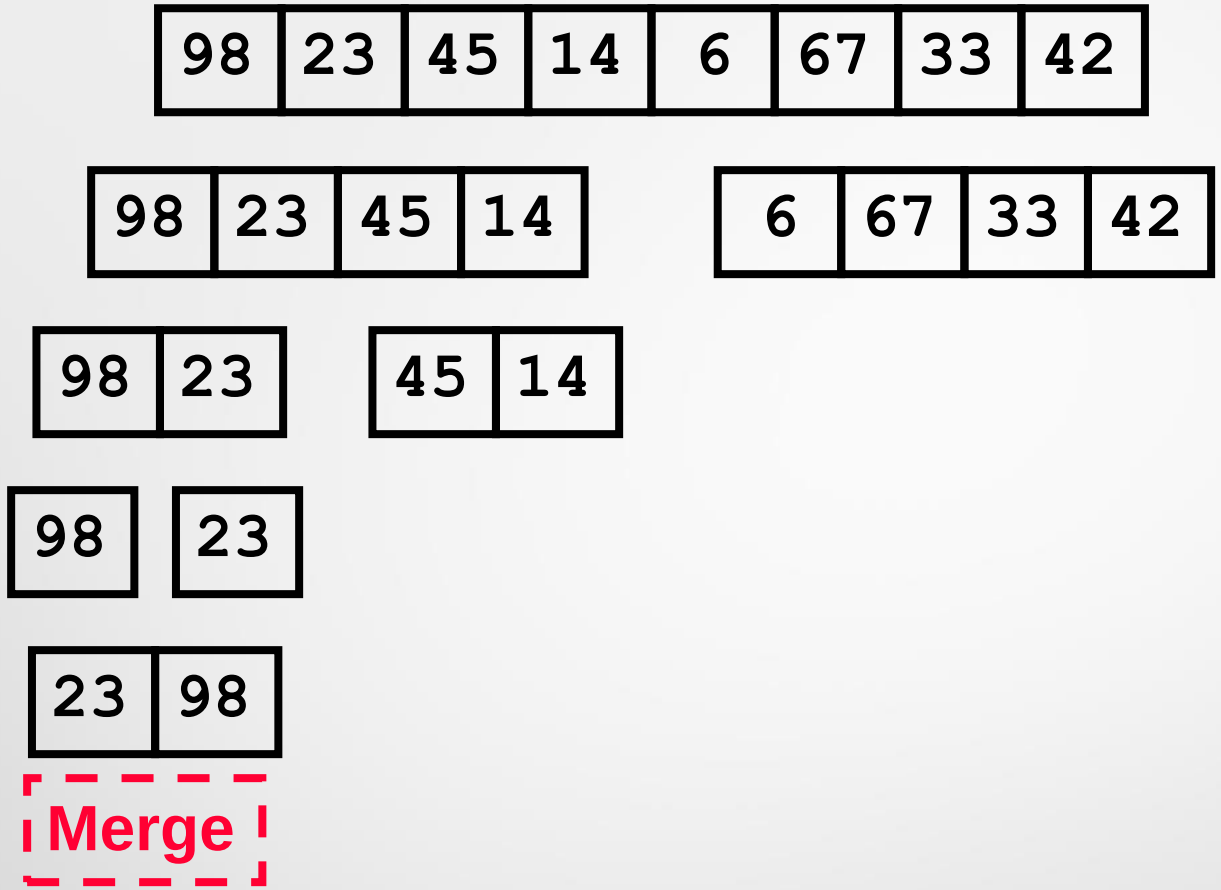

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42

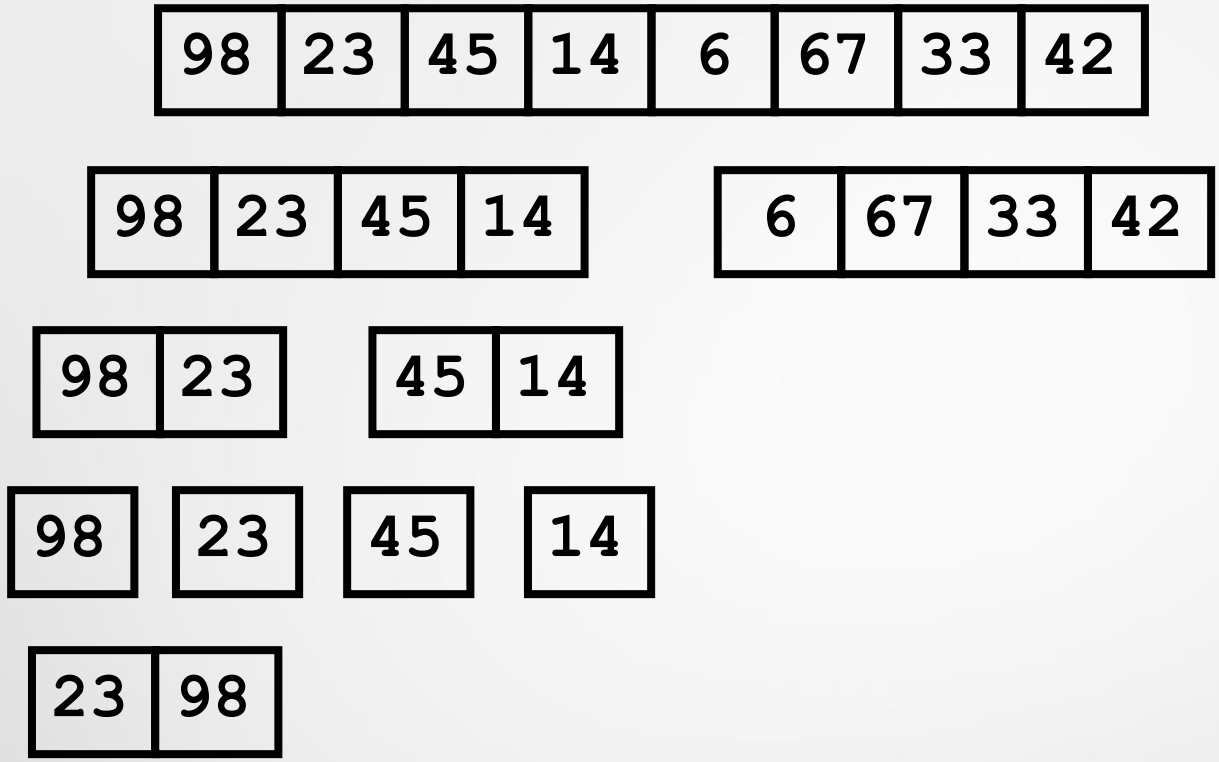

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14

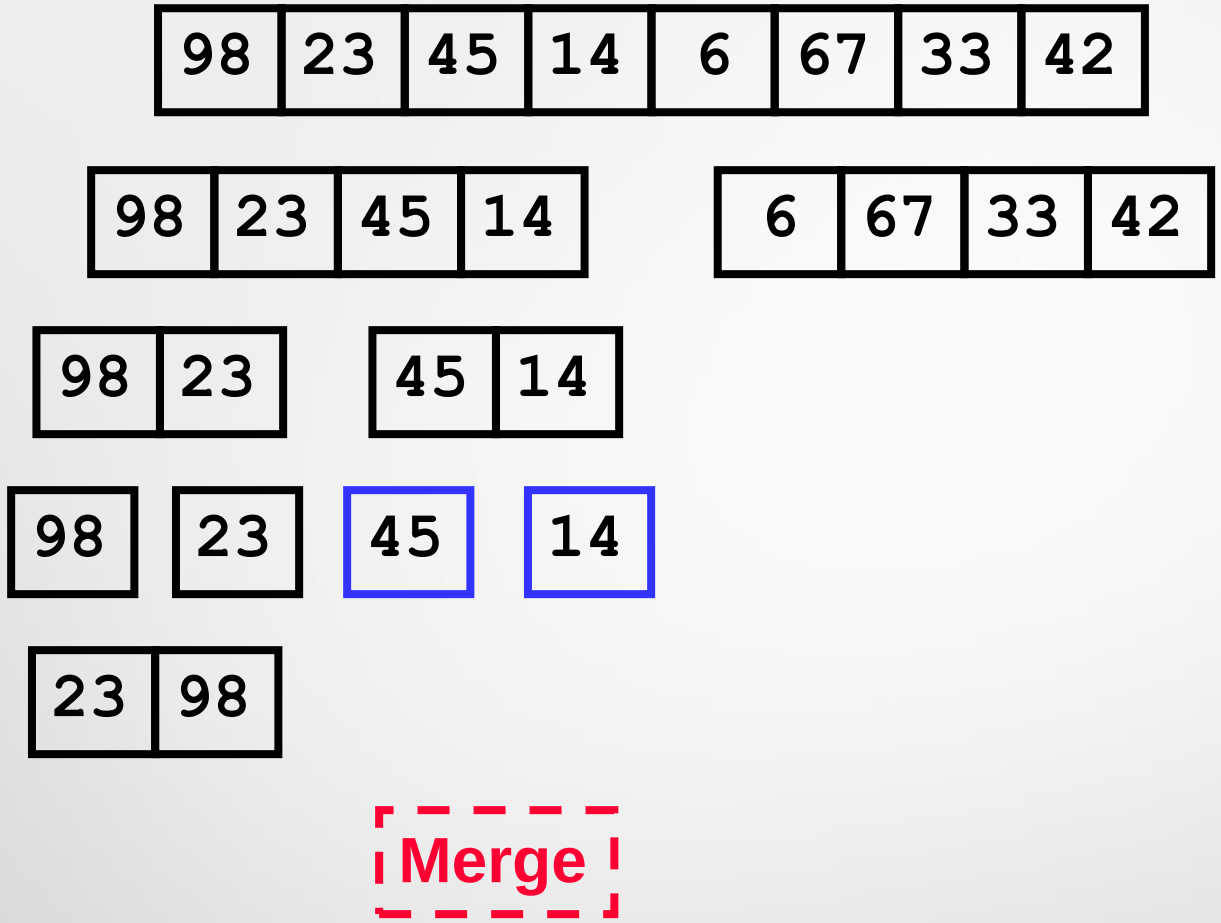

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23

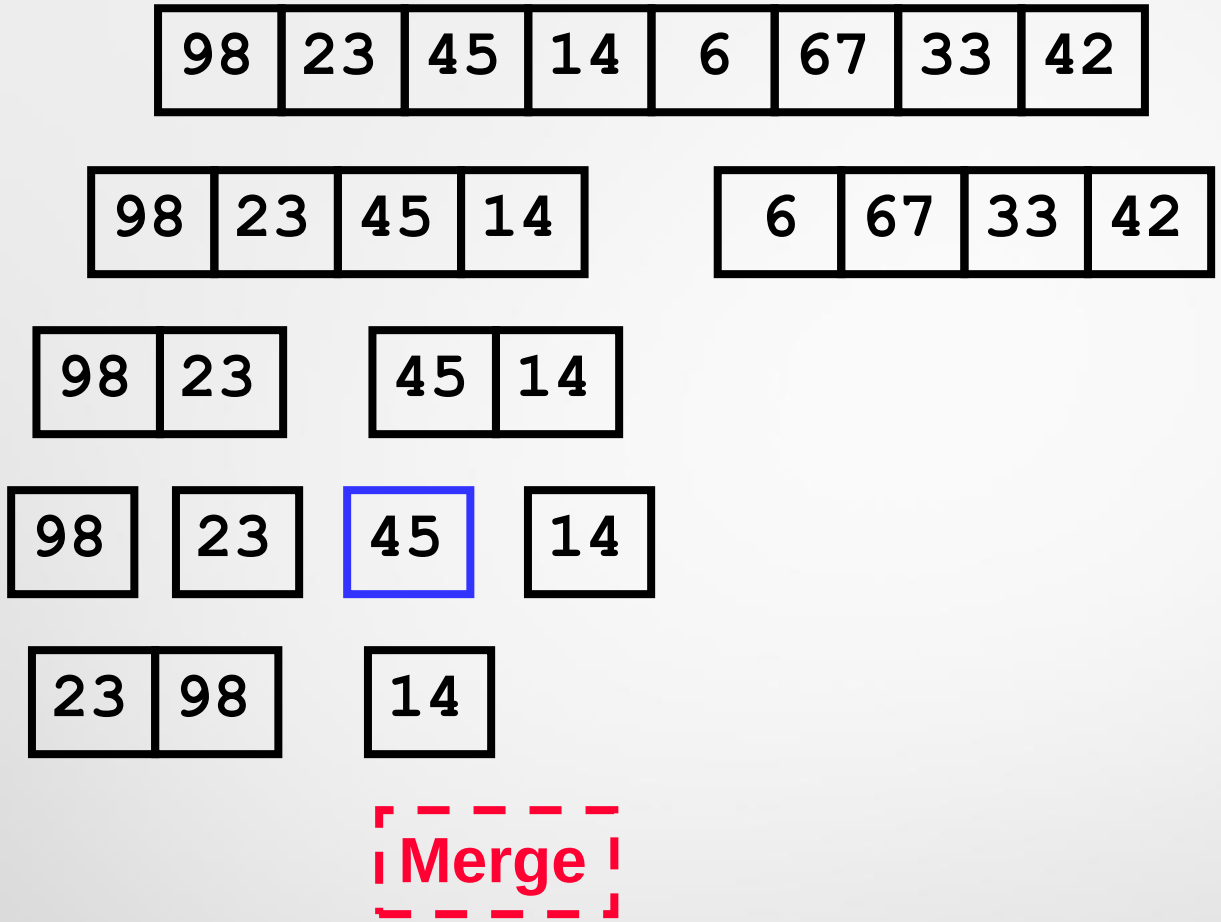

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
Merge

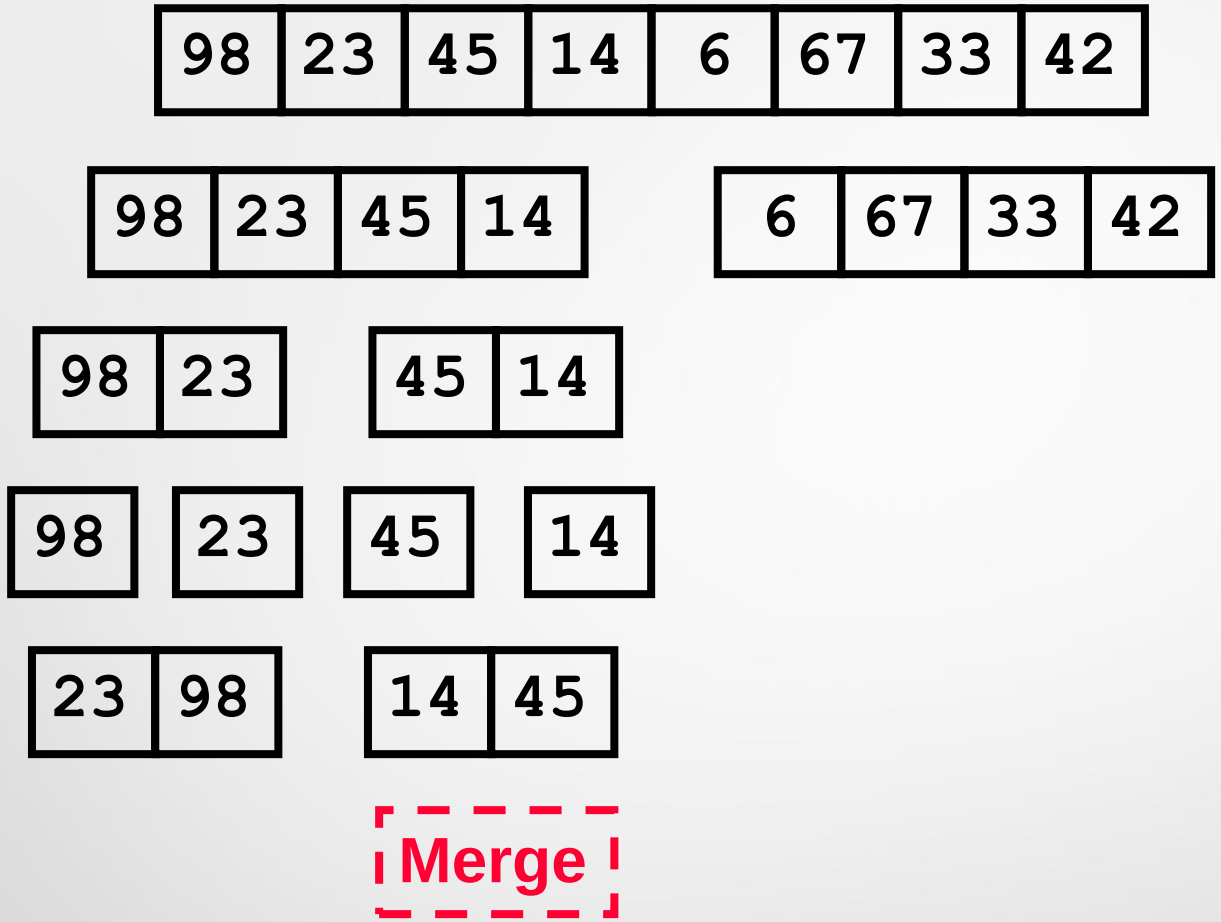

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23
Merge

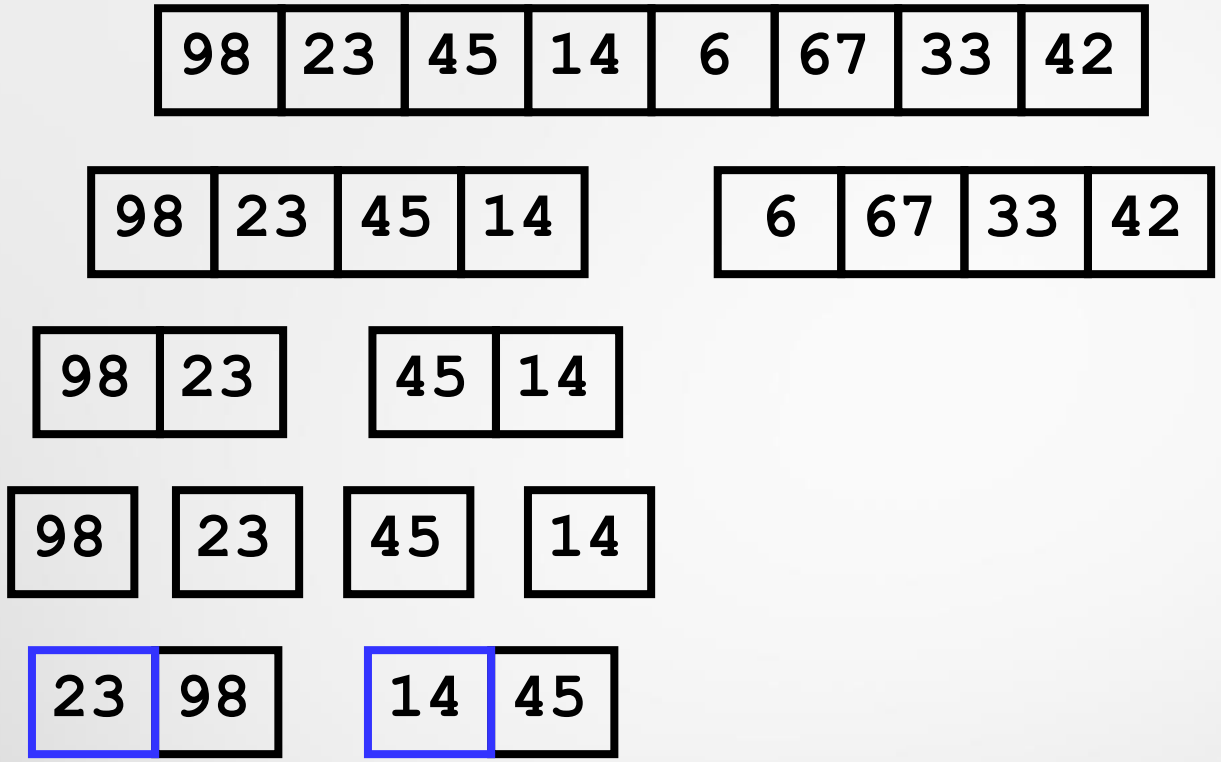

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23 98
Merge

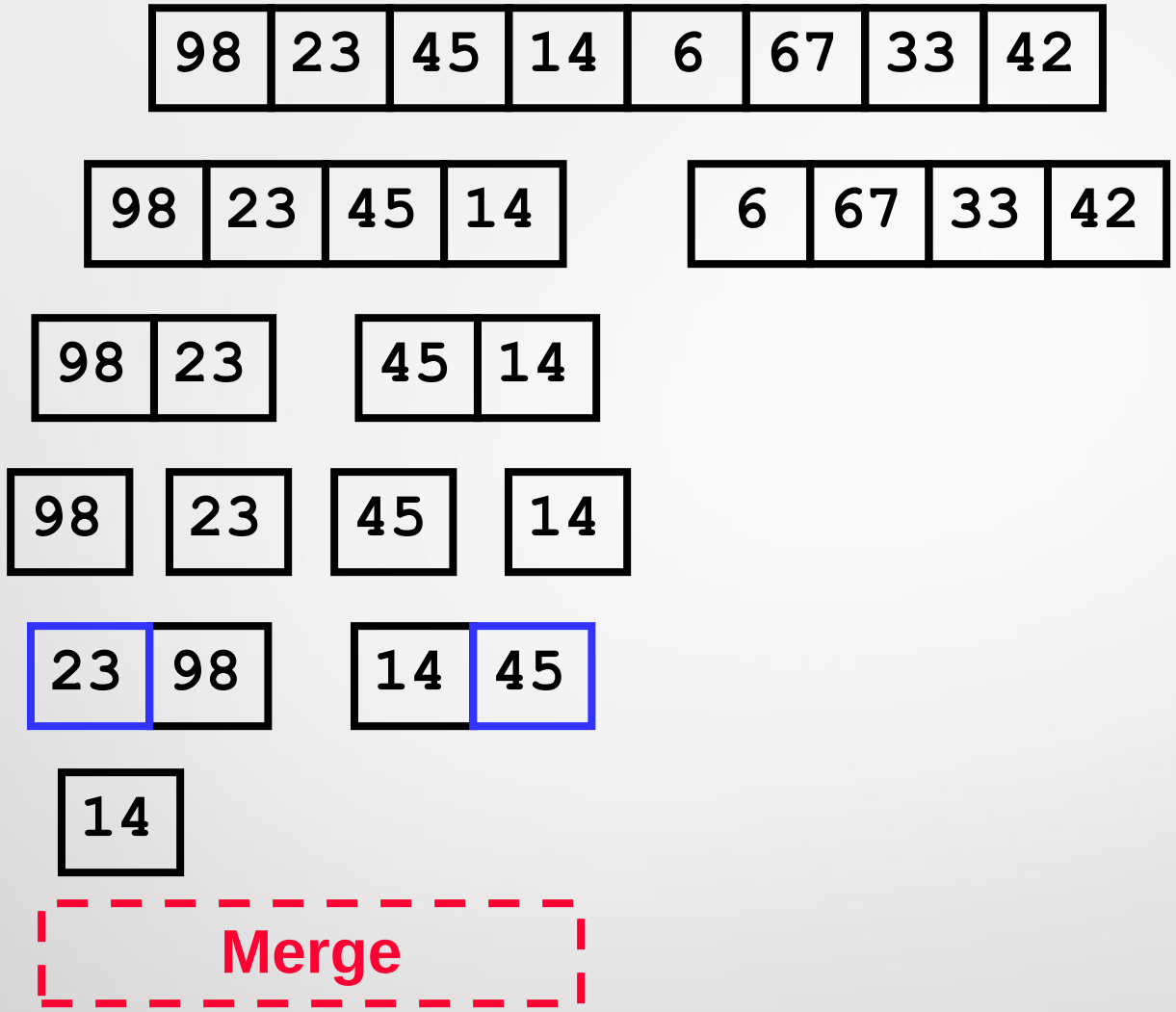

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98

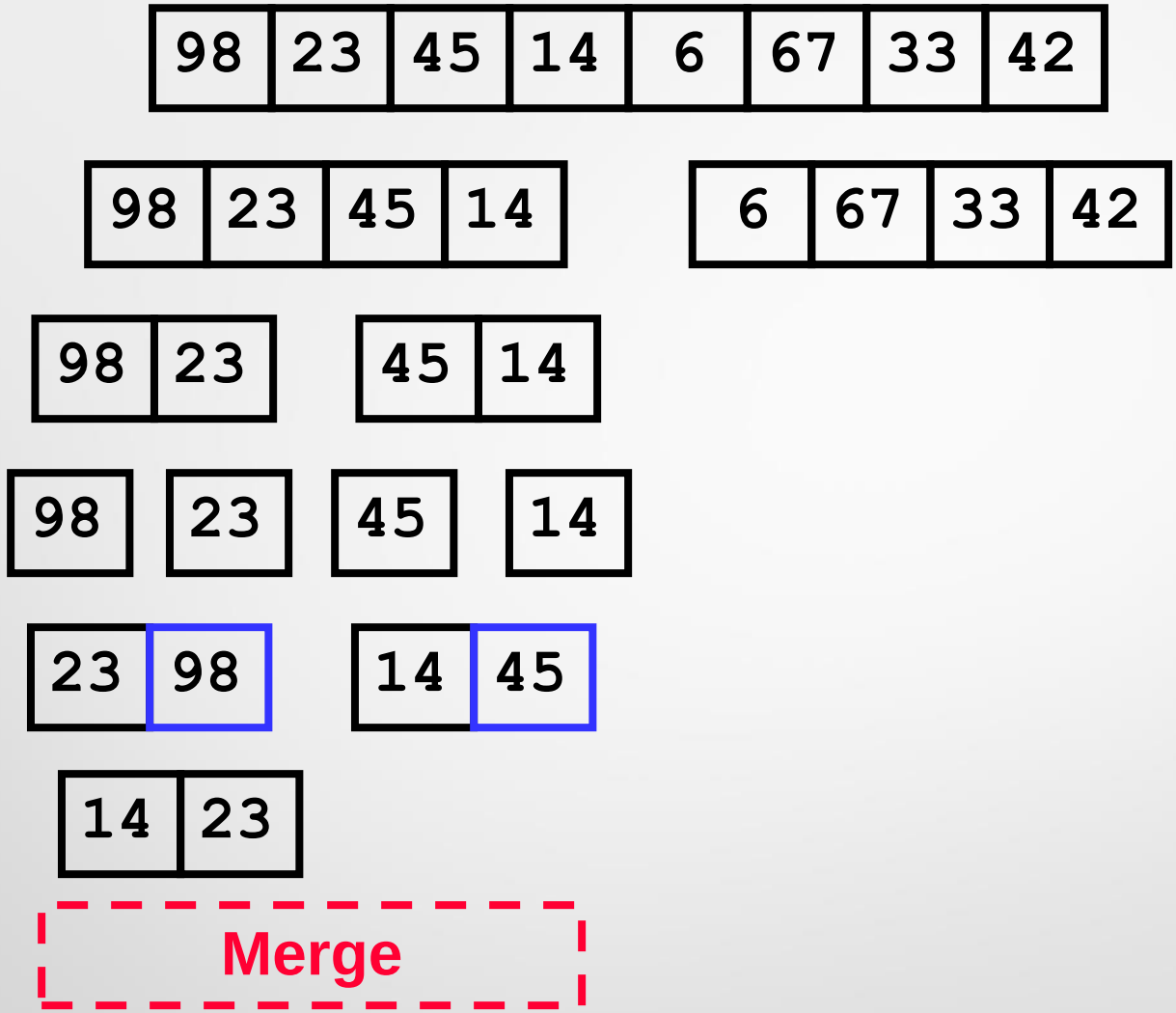

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
Merge

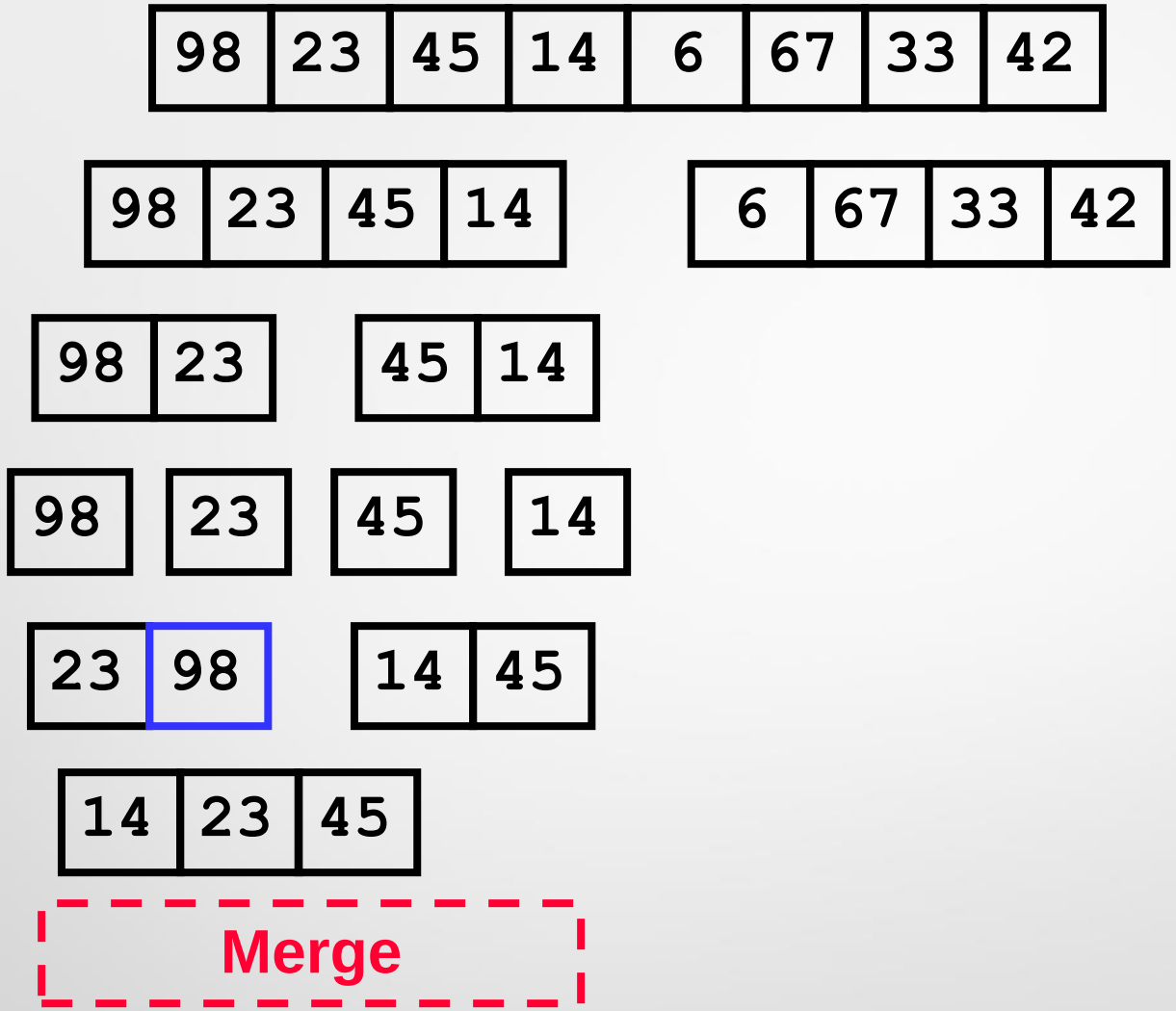

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14
Merge

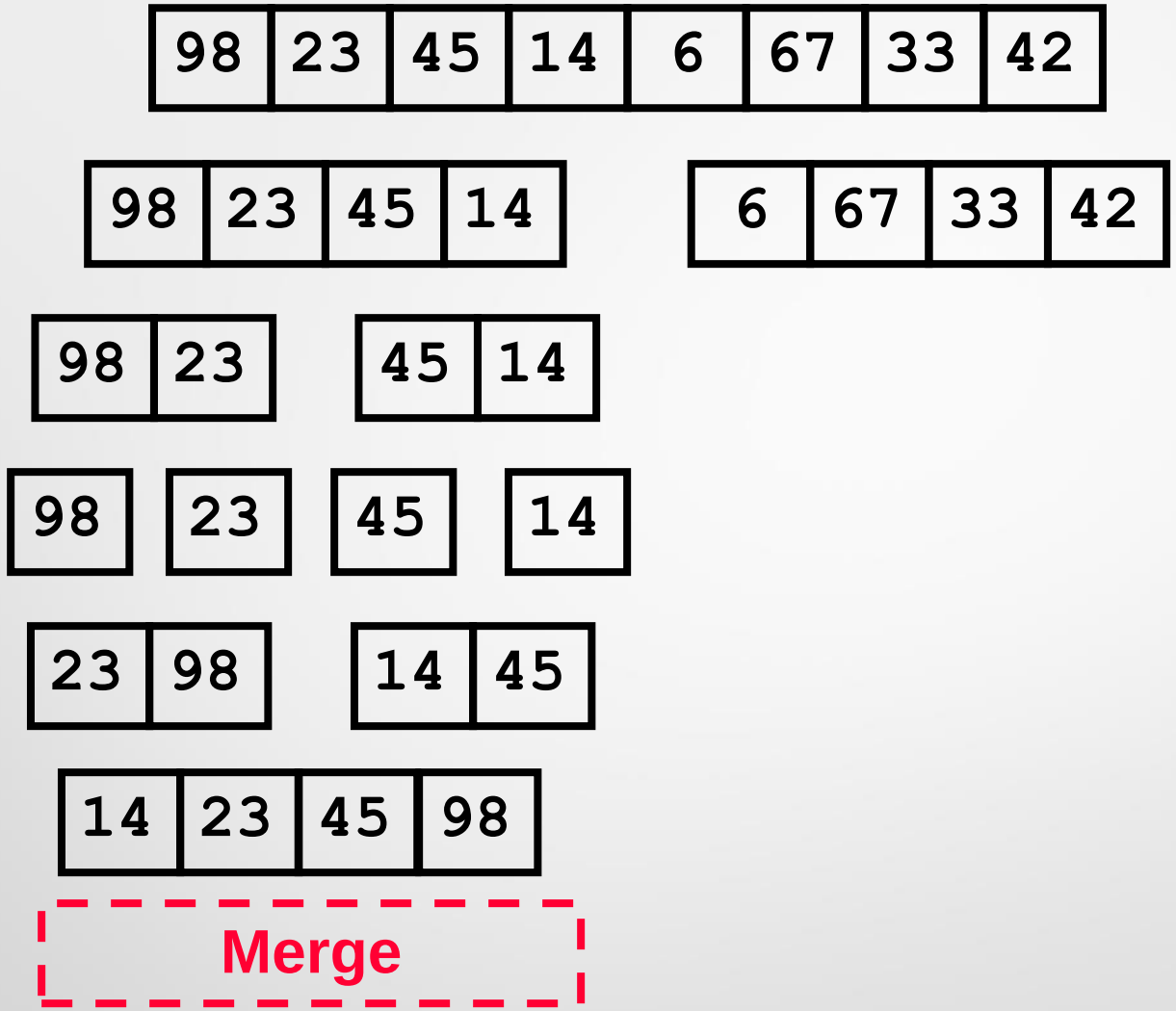

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

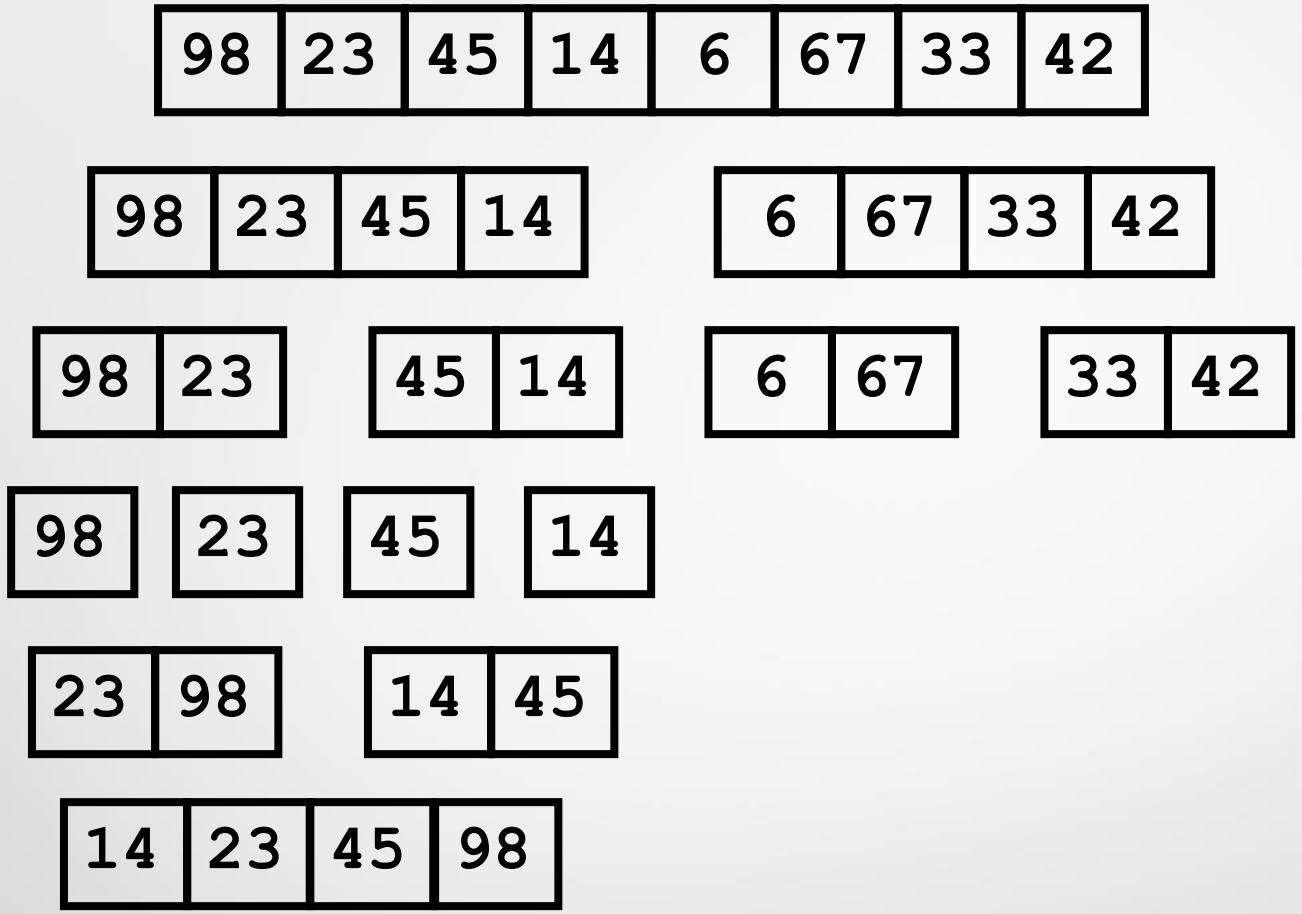

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

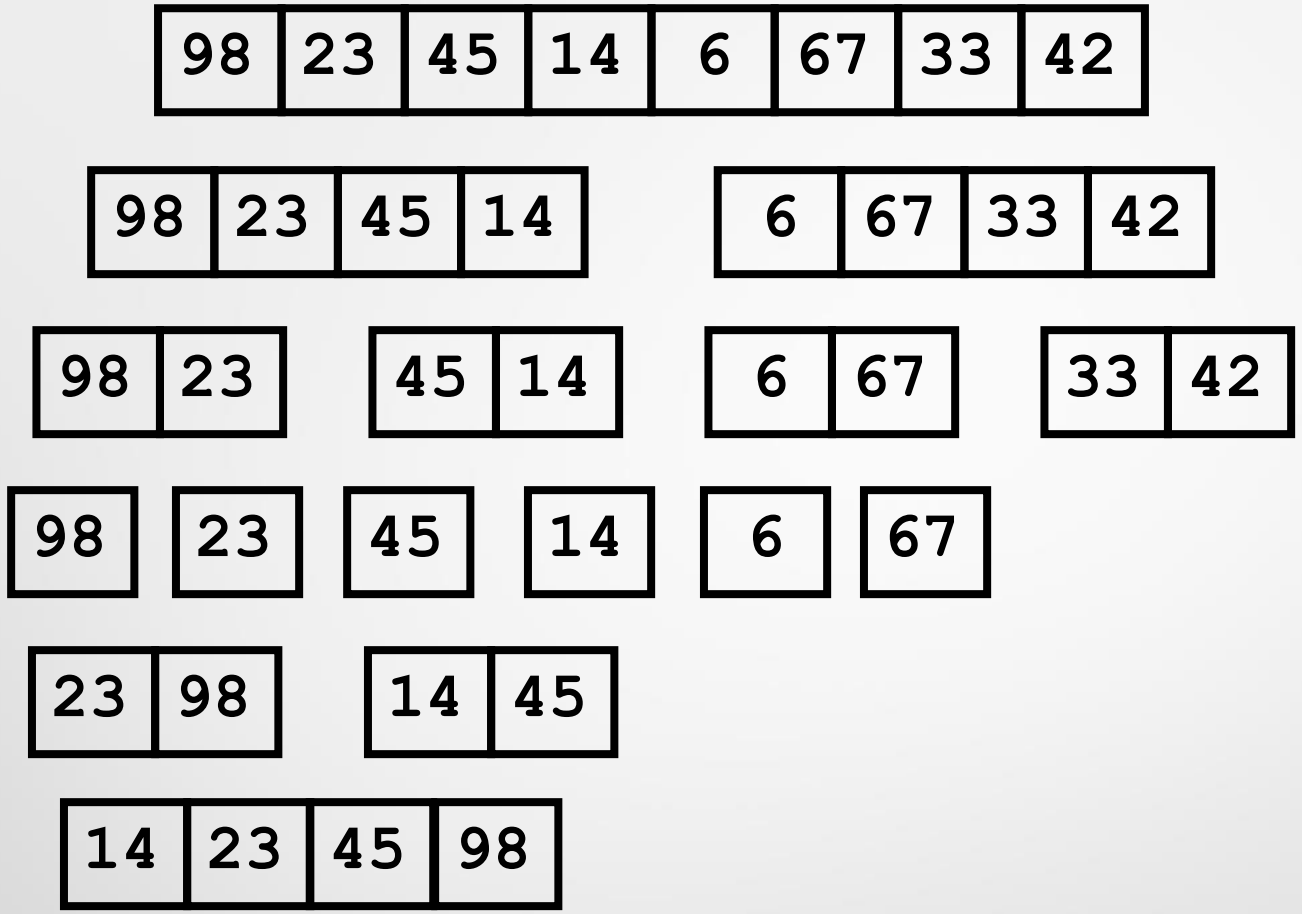

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14
Merge

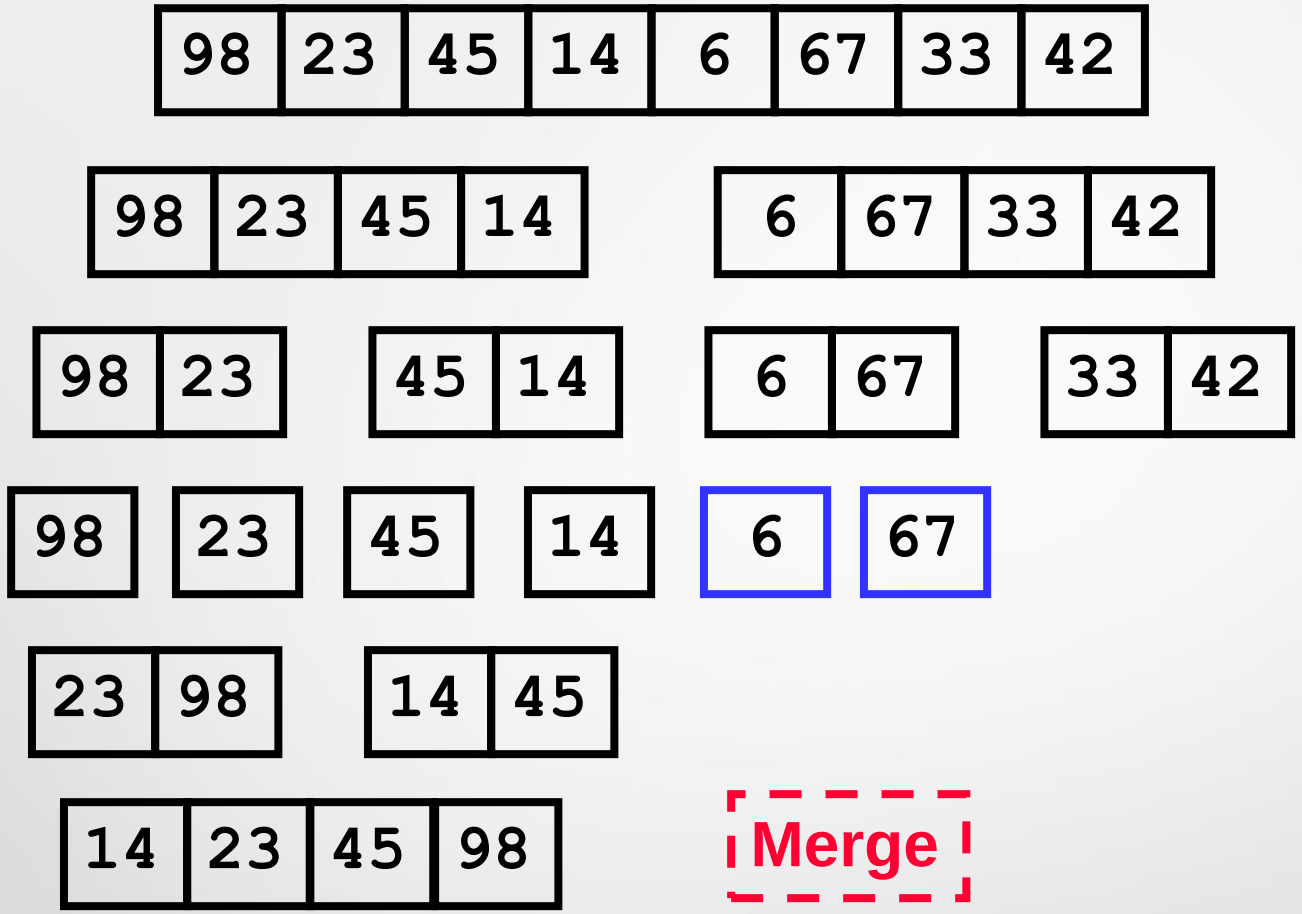

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23
Merge

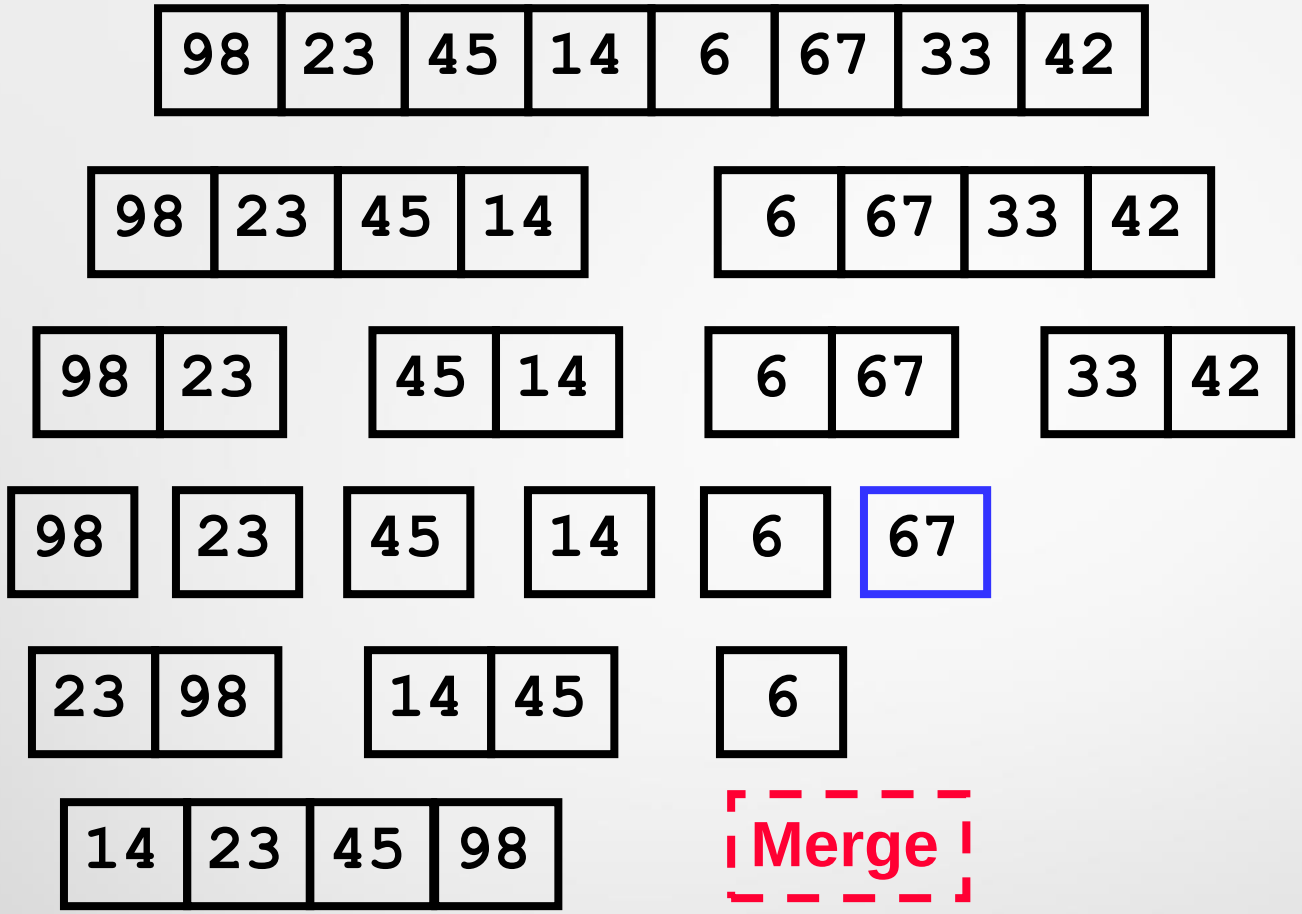

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45
Merge

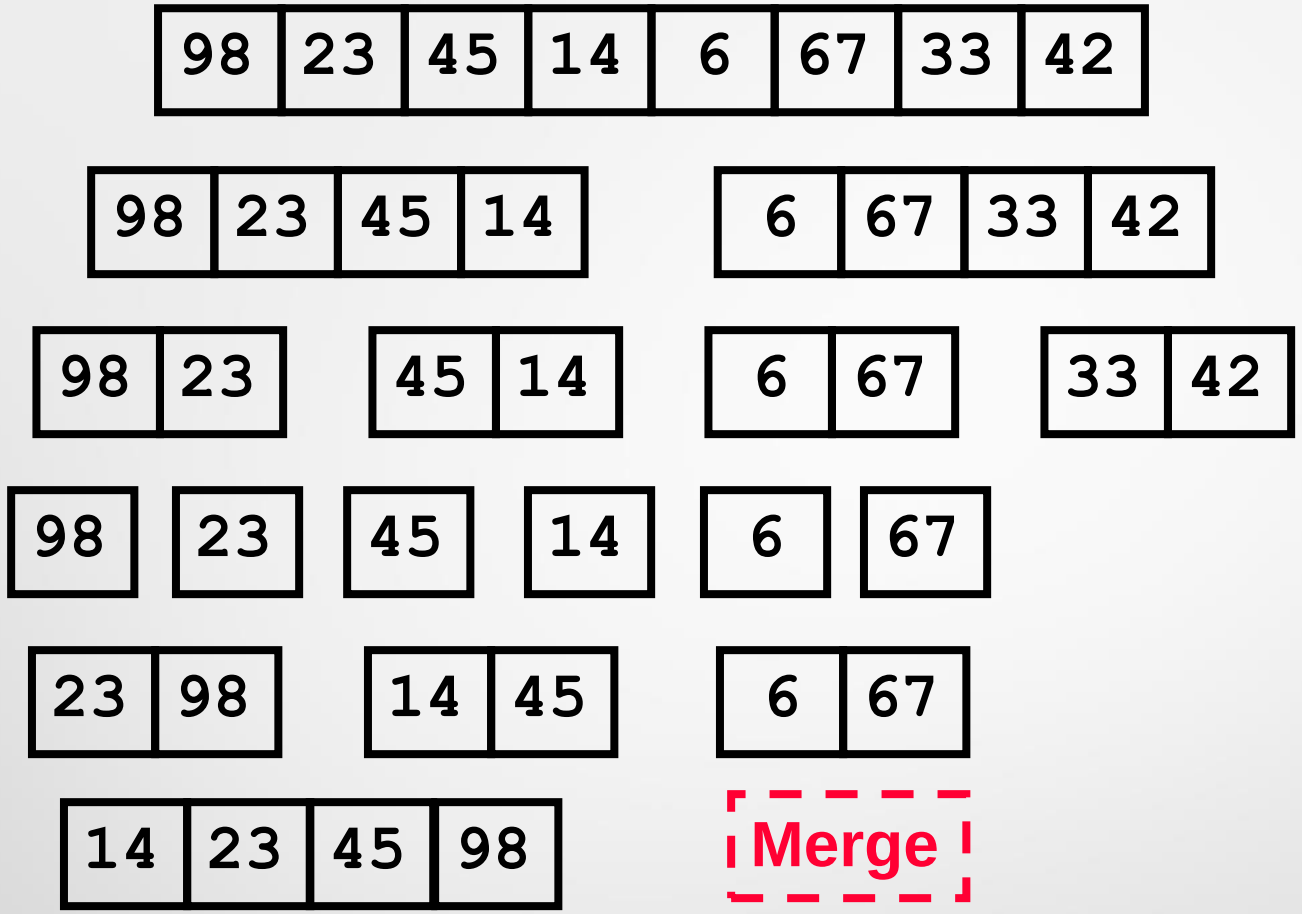

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45 98
Merge

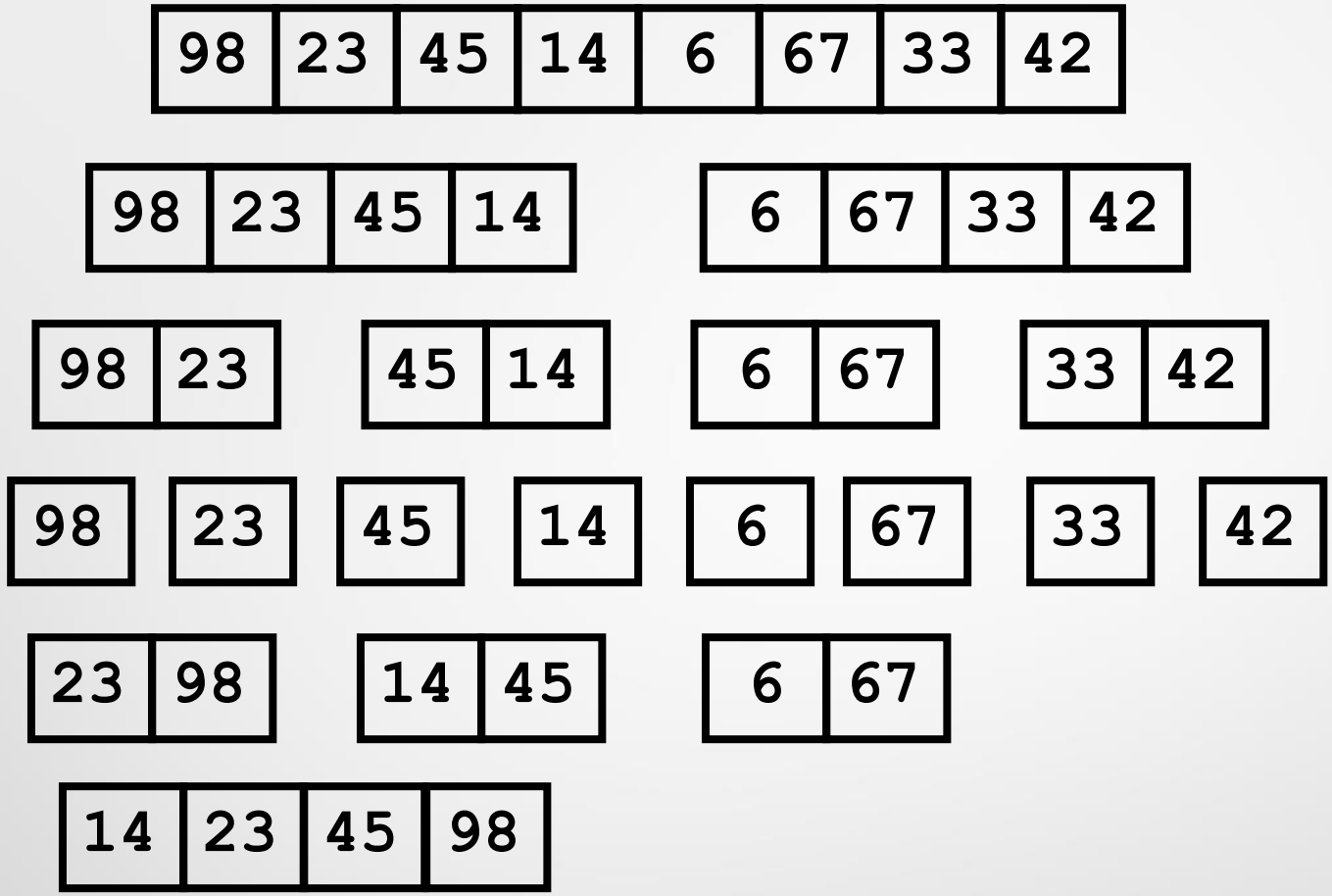

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
23 98
14 45
14 23 45 98

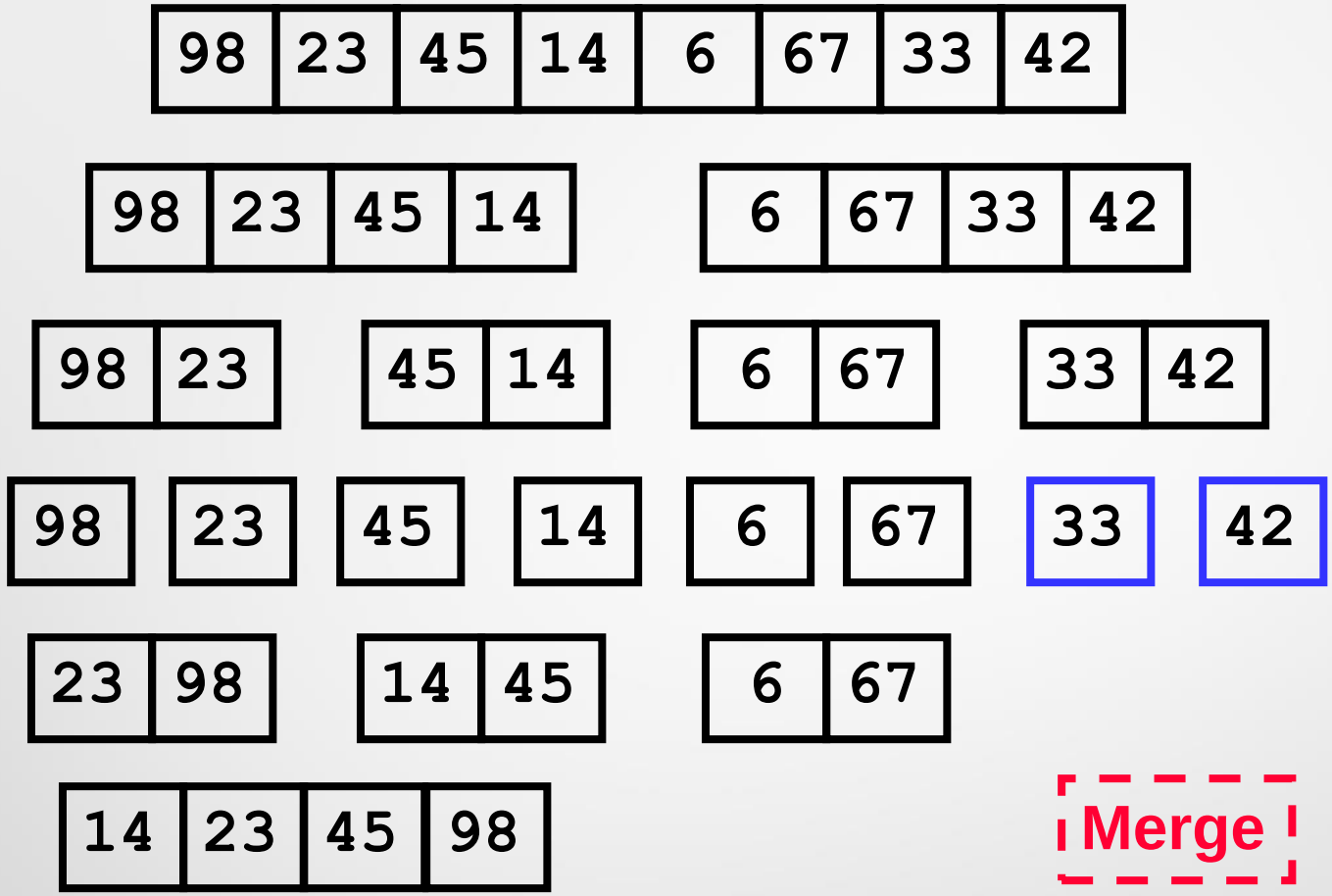

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98

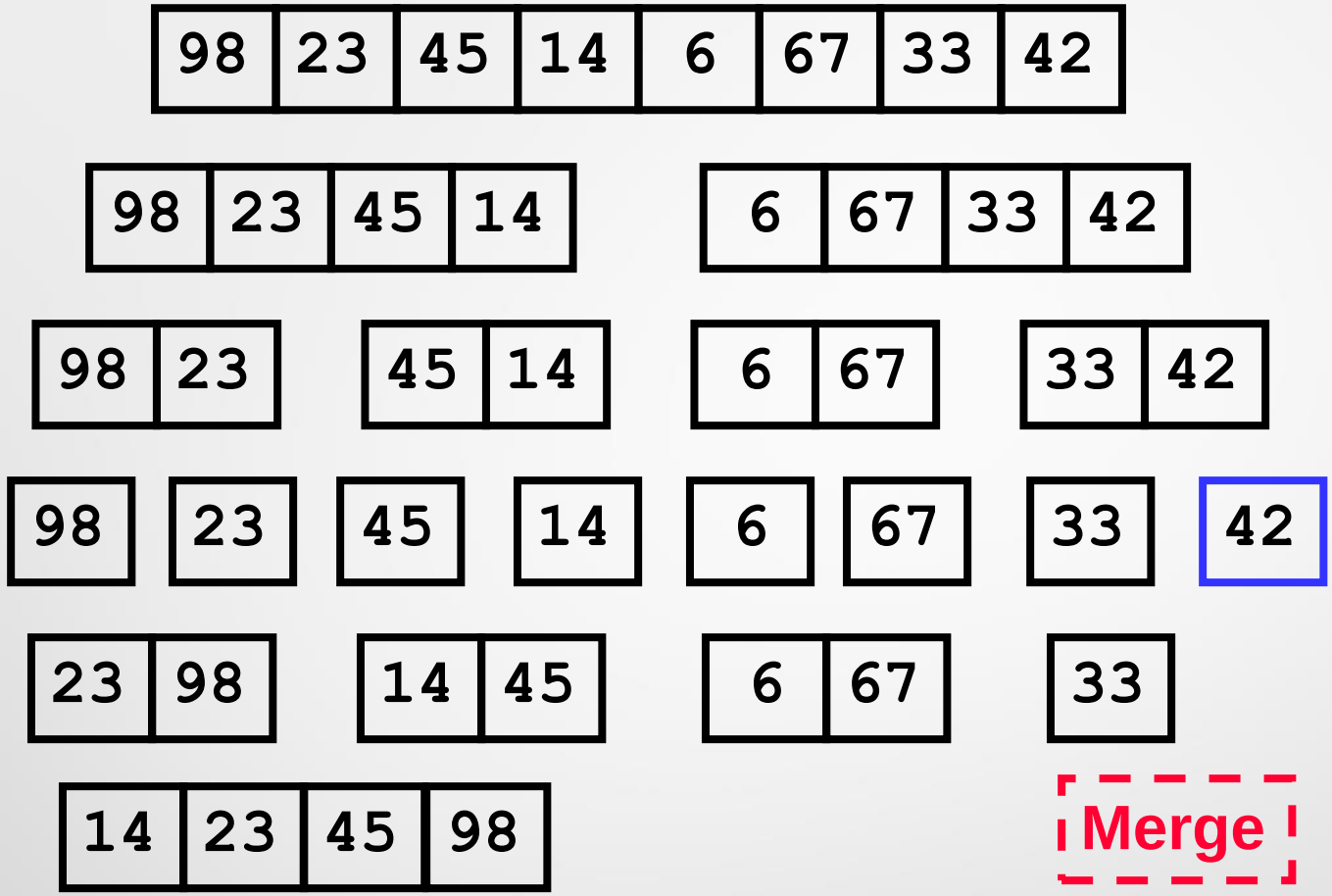

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98
Merge

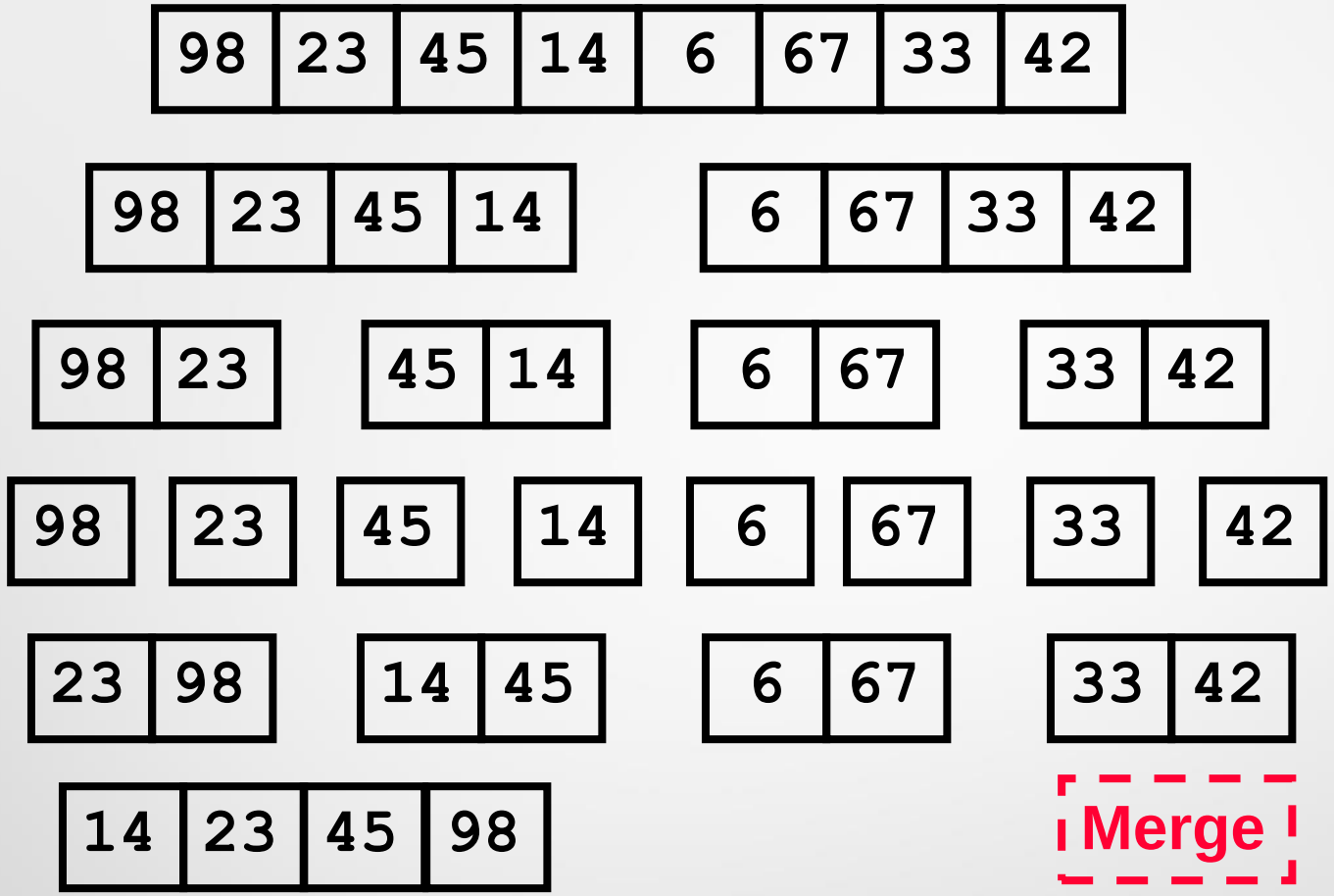

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6
14 23 45 98
Merge

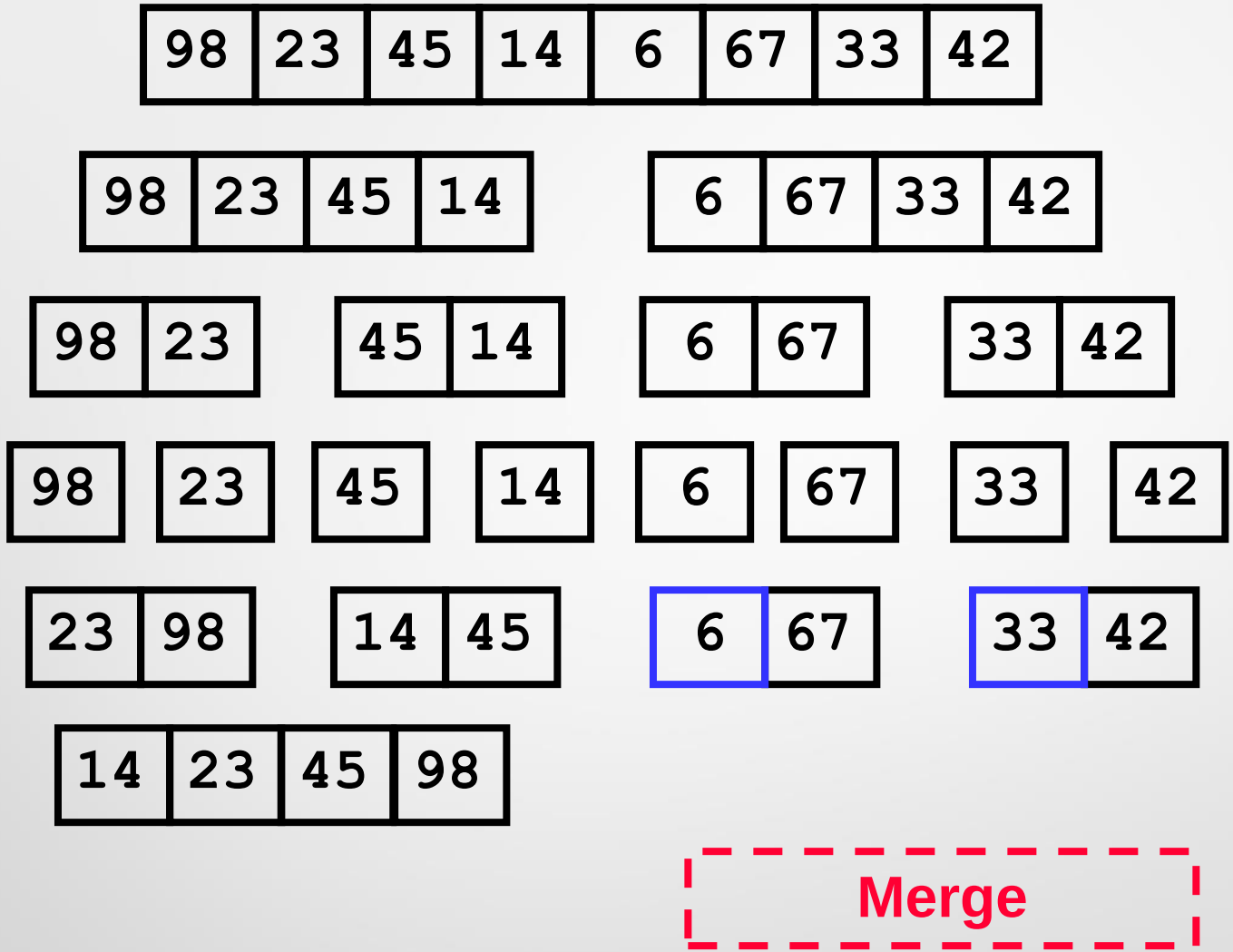

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6 67
14 23 45 98
Merge

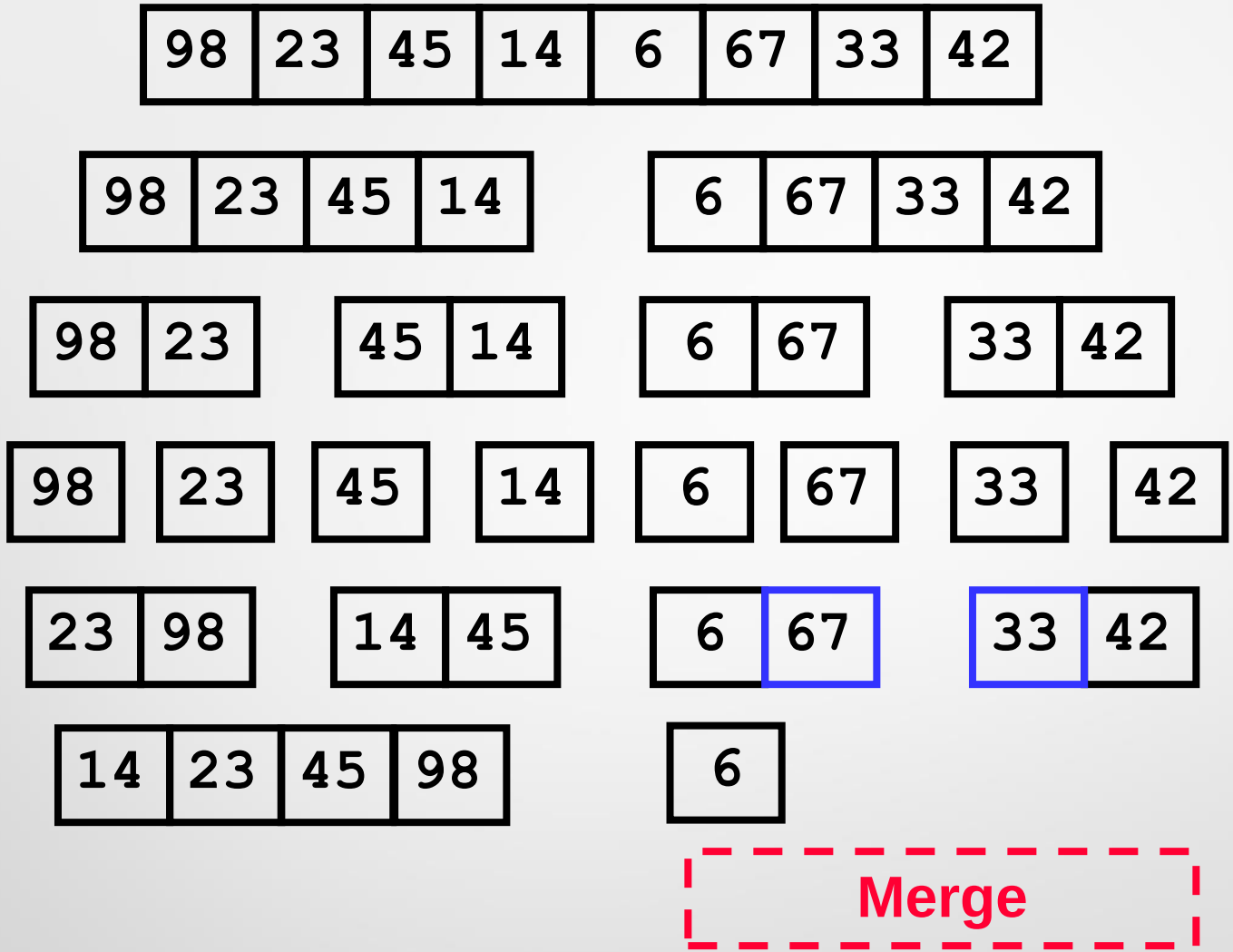

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98

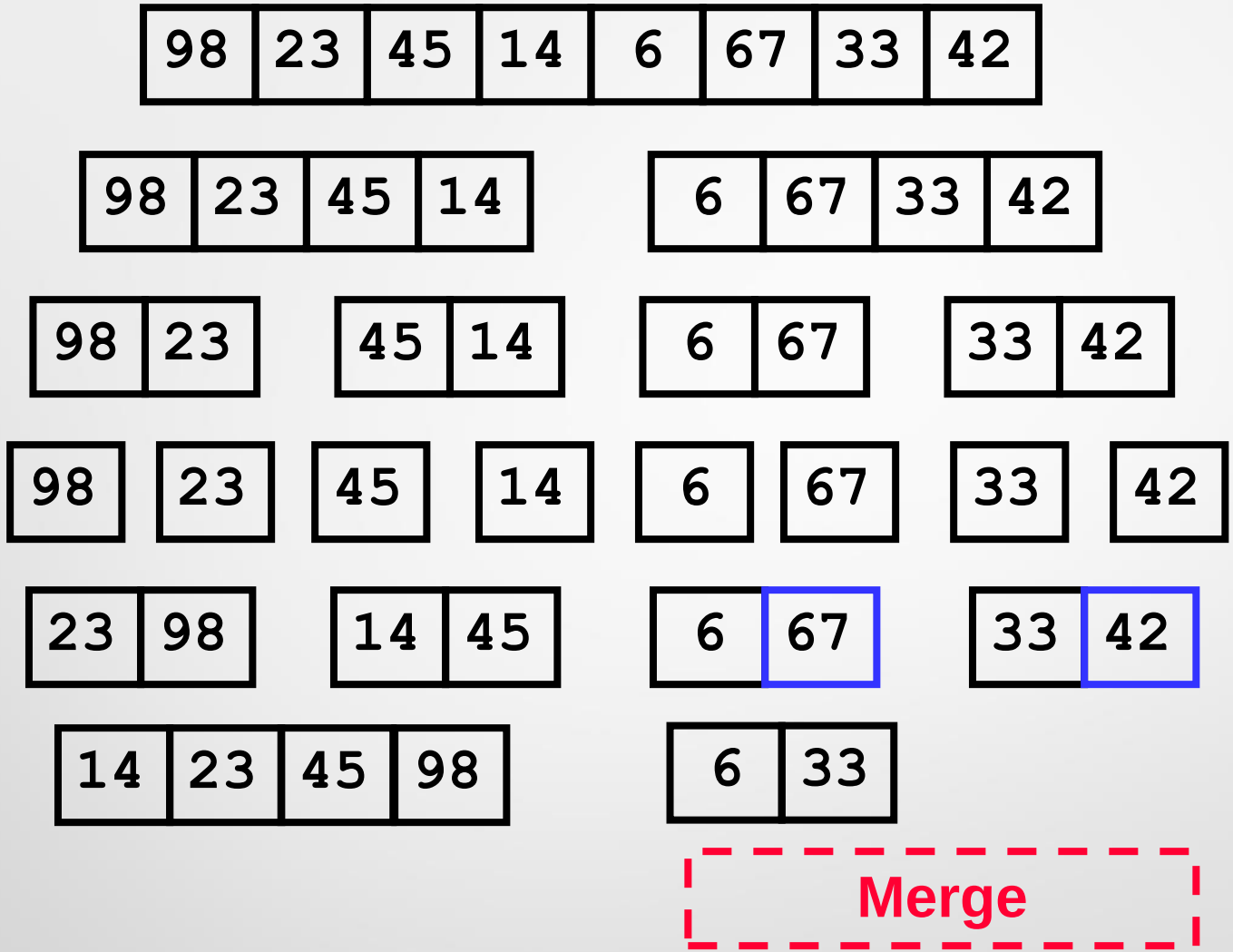

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98
Merge

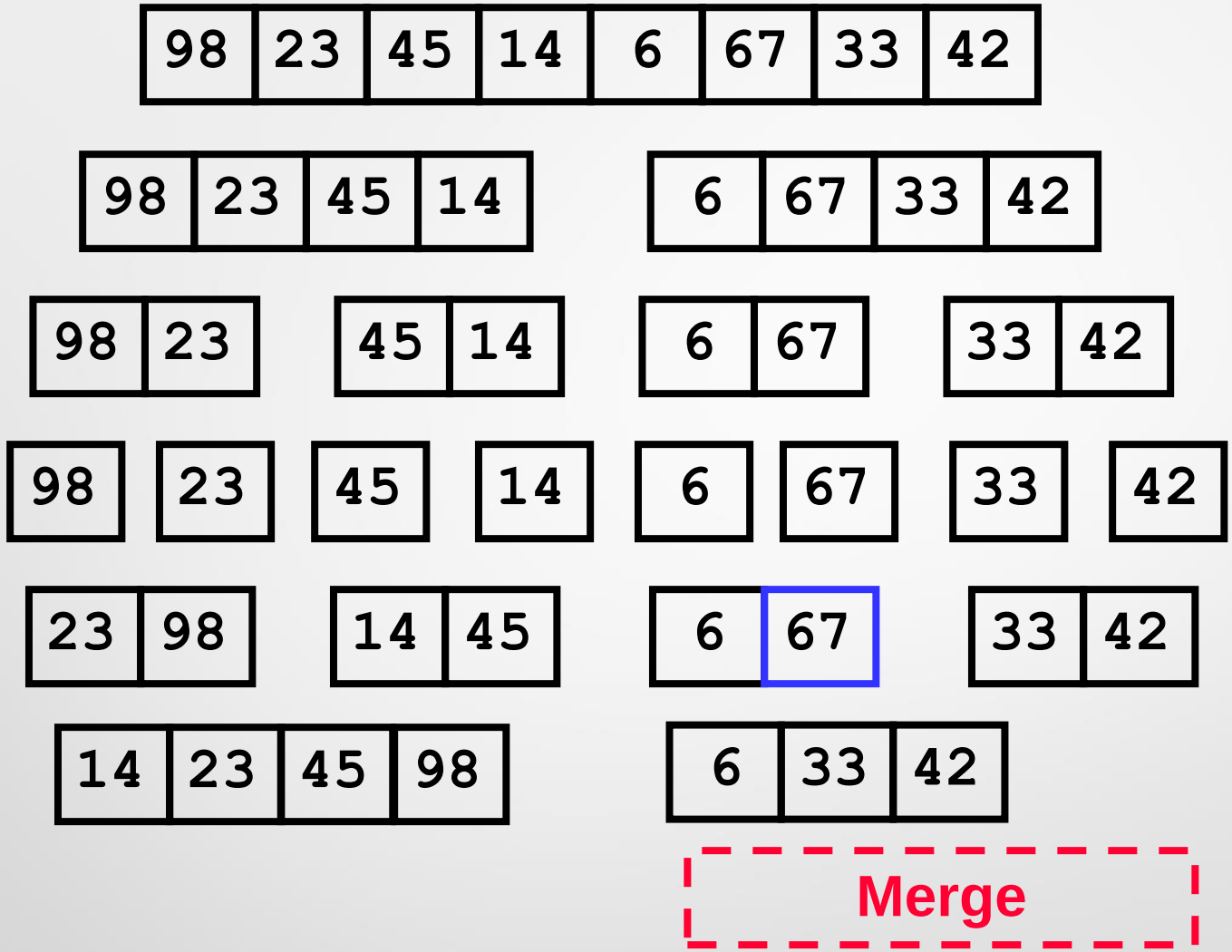

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33
14 23 45 98
Merge

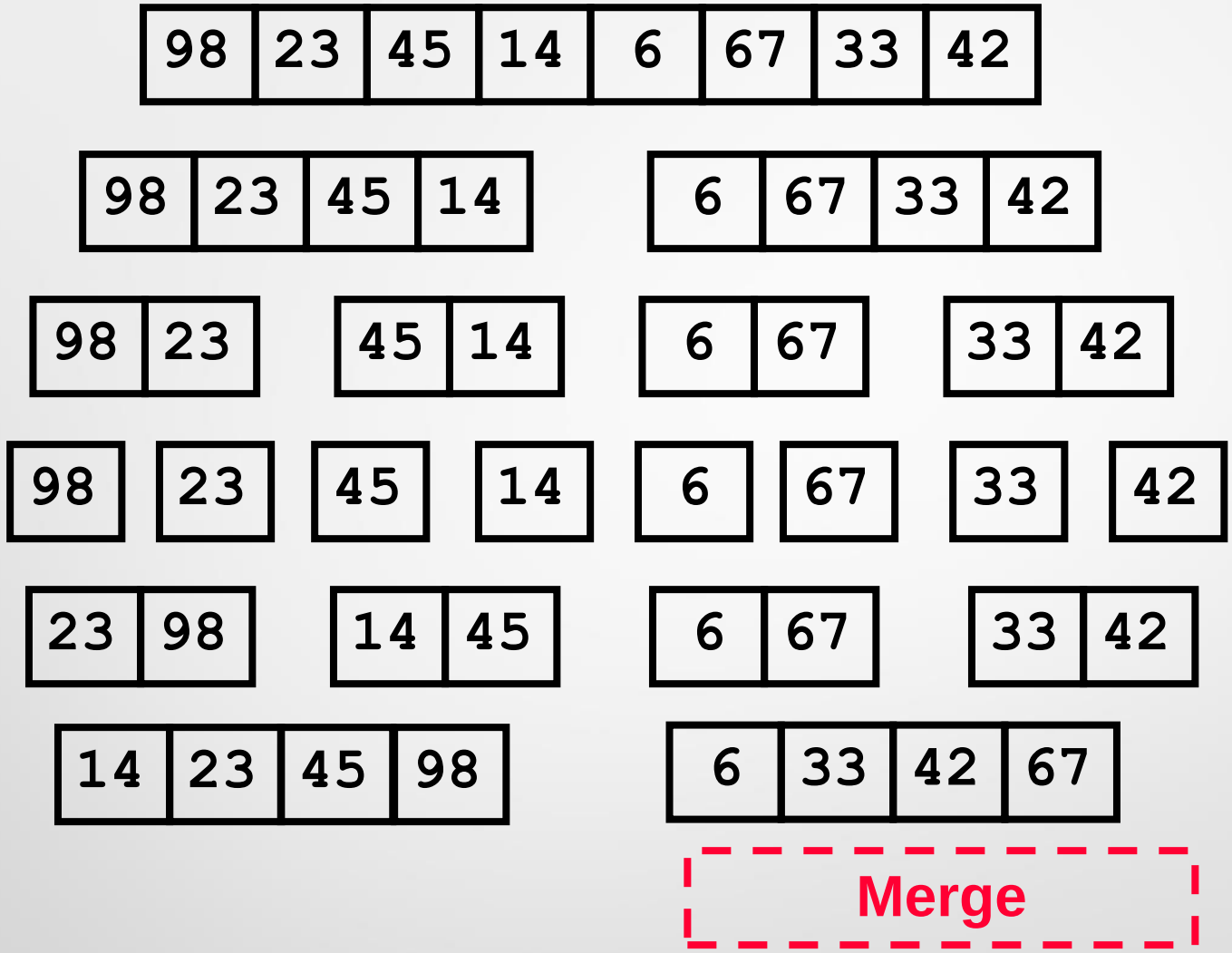

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge

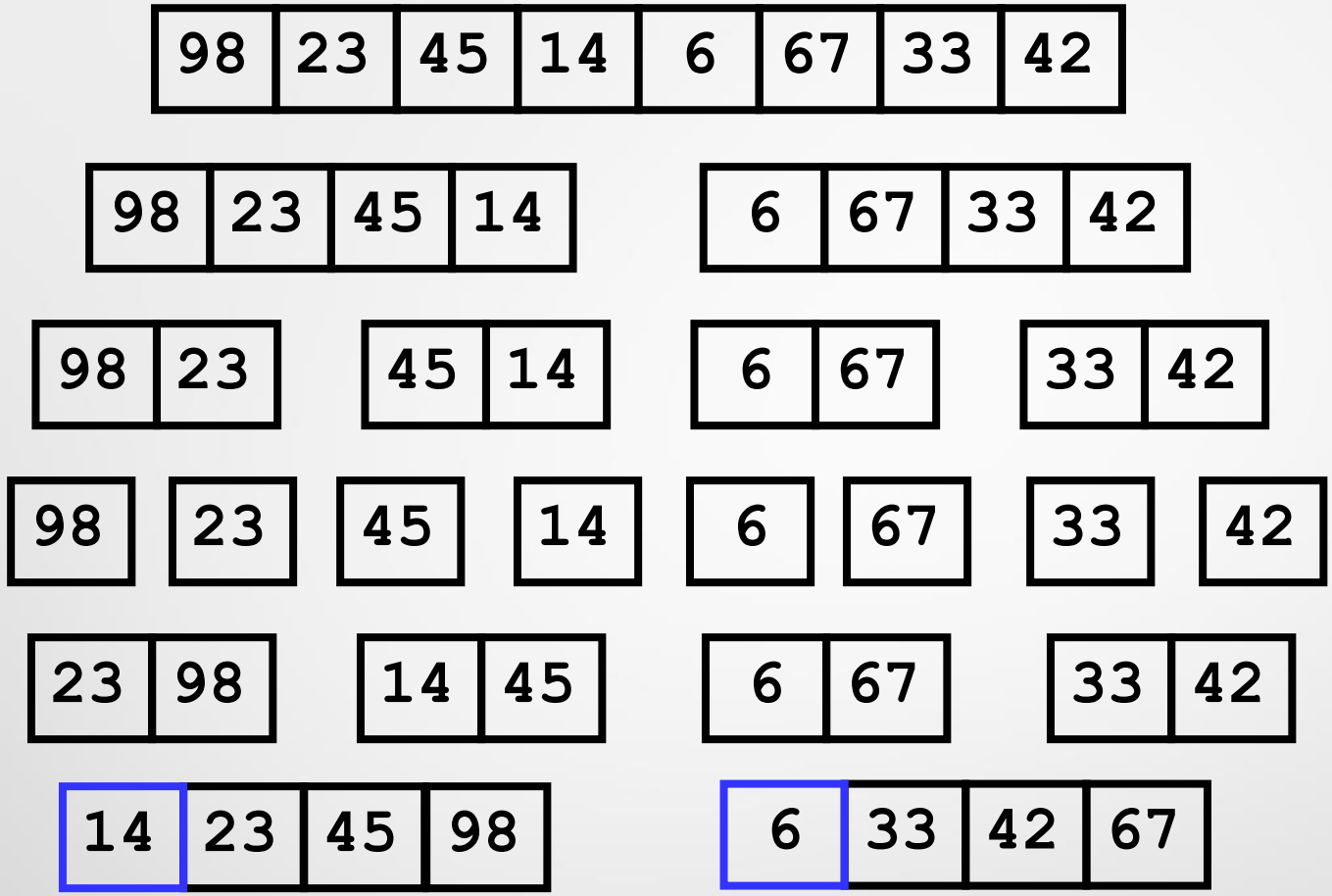

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6
Merge

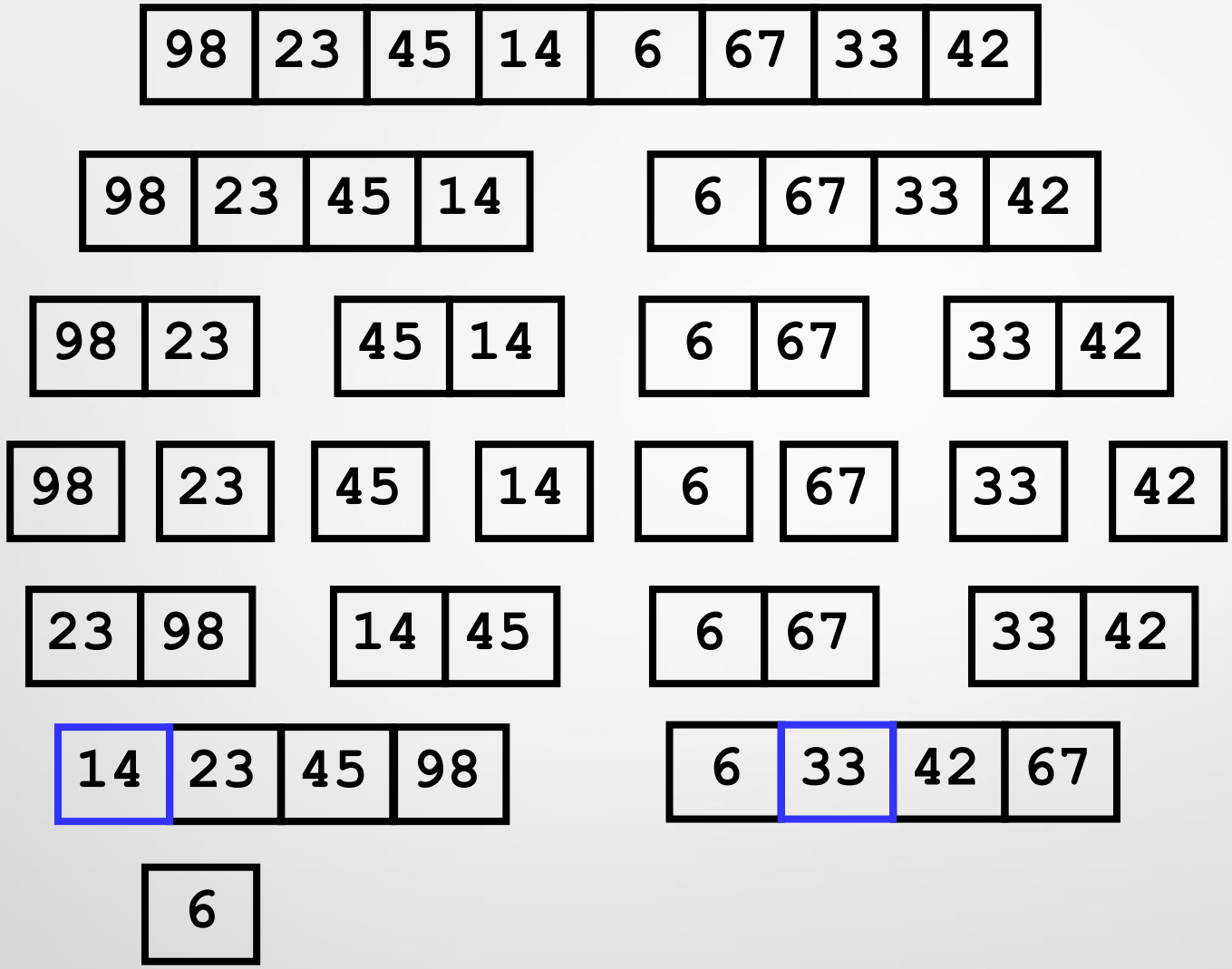

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33
Merge

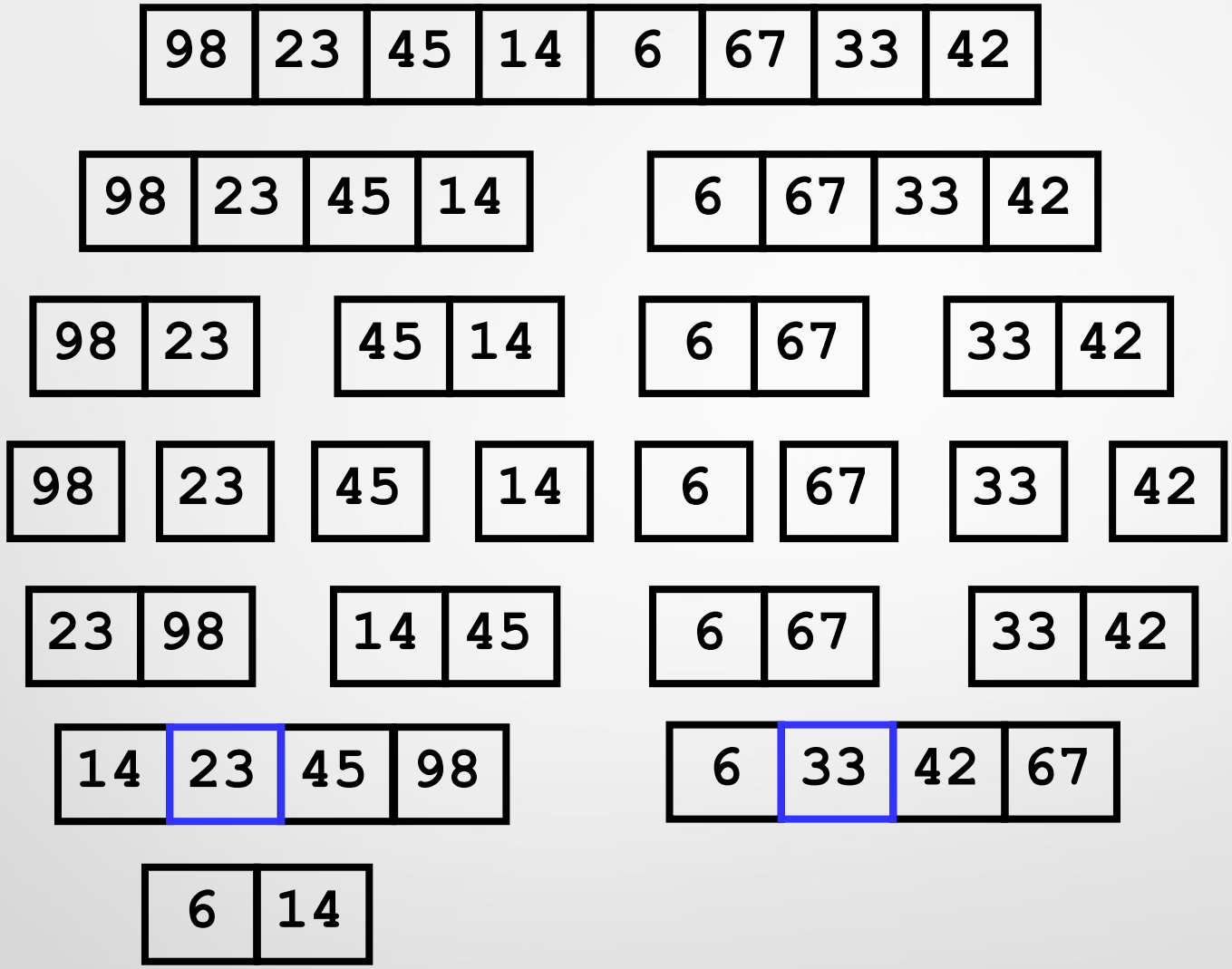

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42
Merge

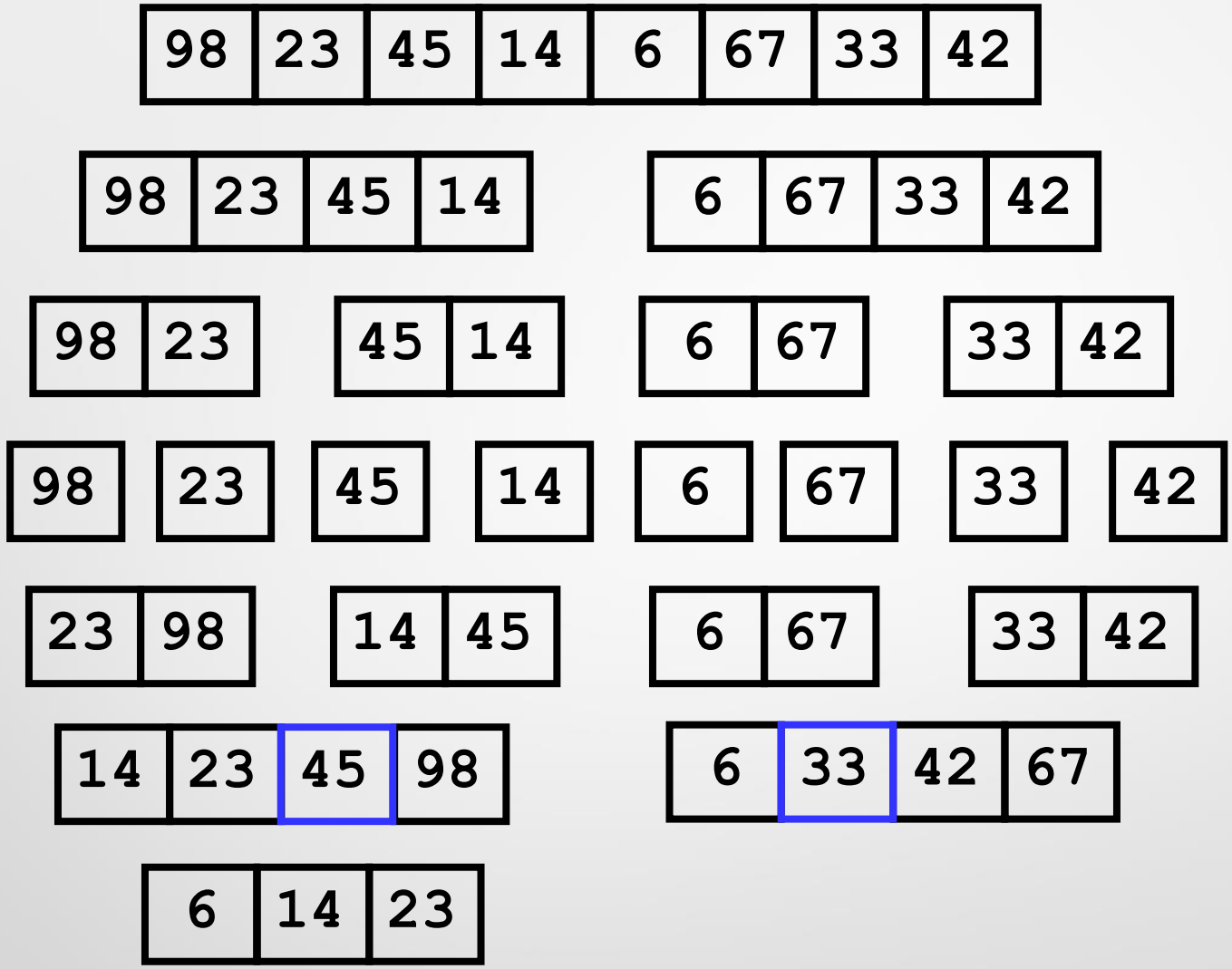

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

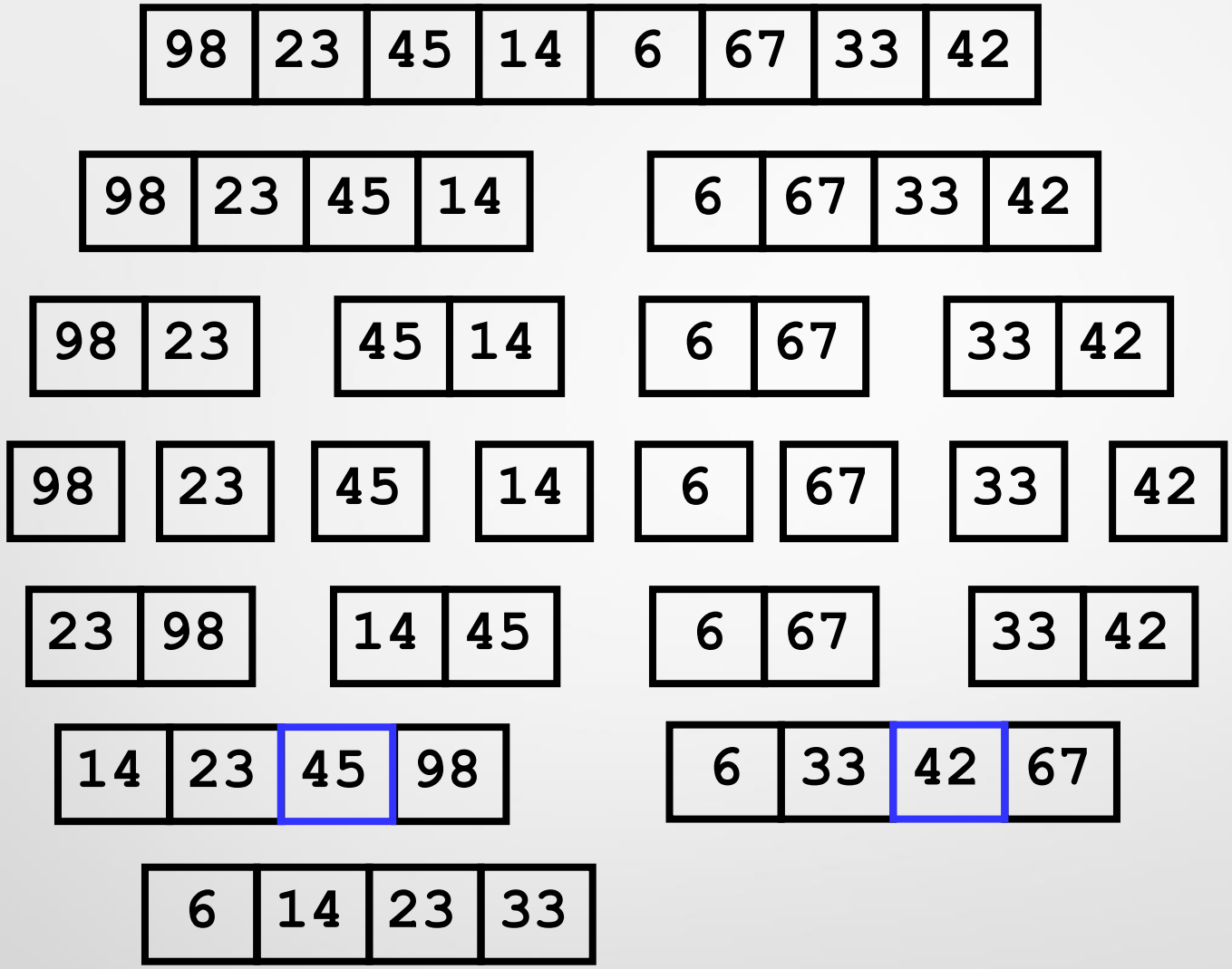

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67

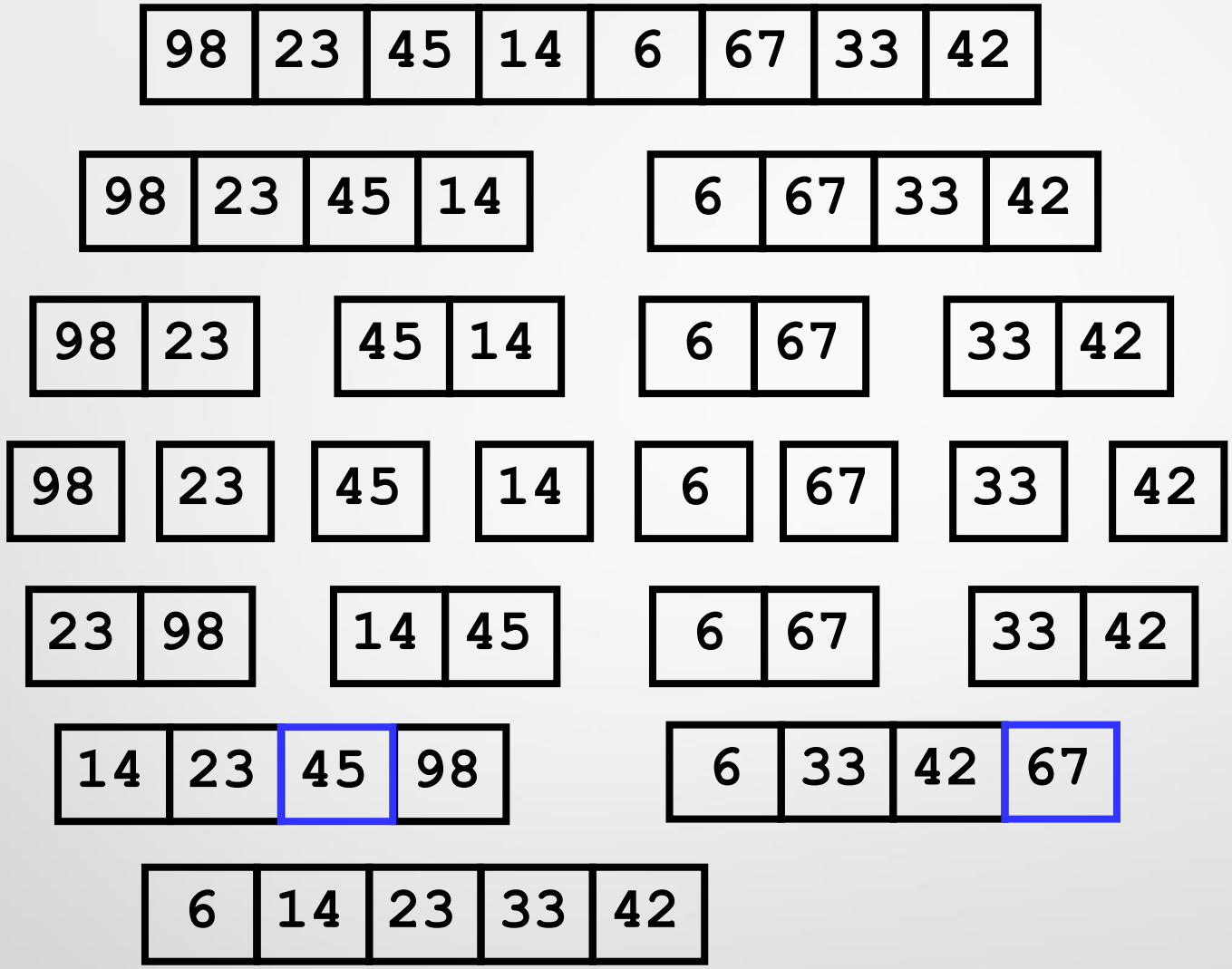
Merge

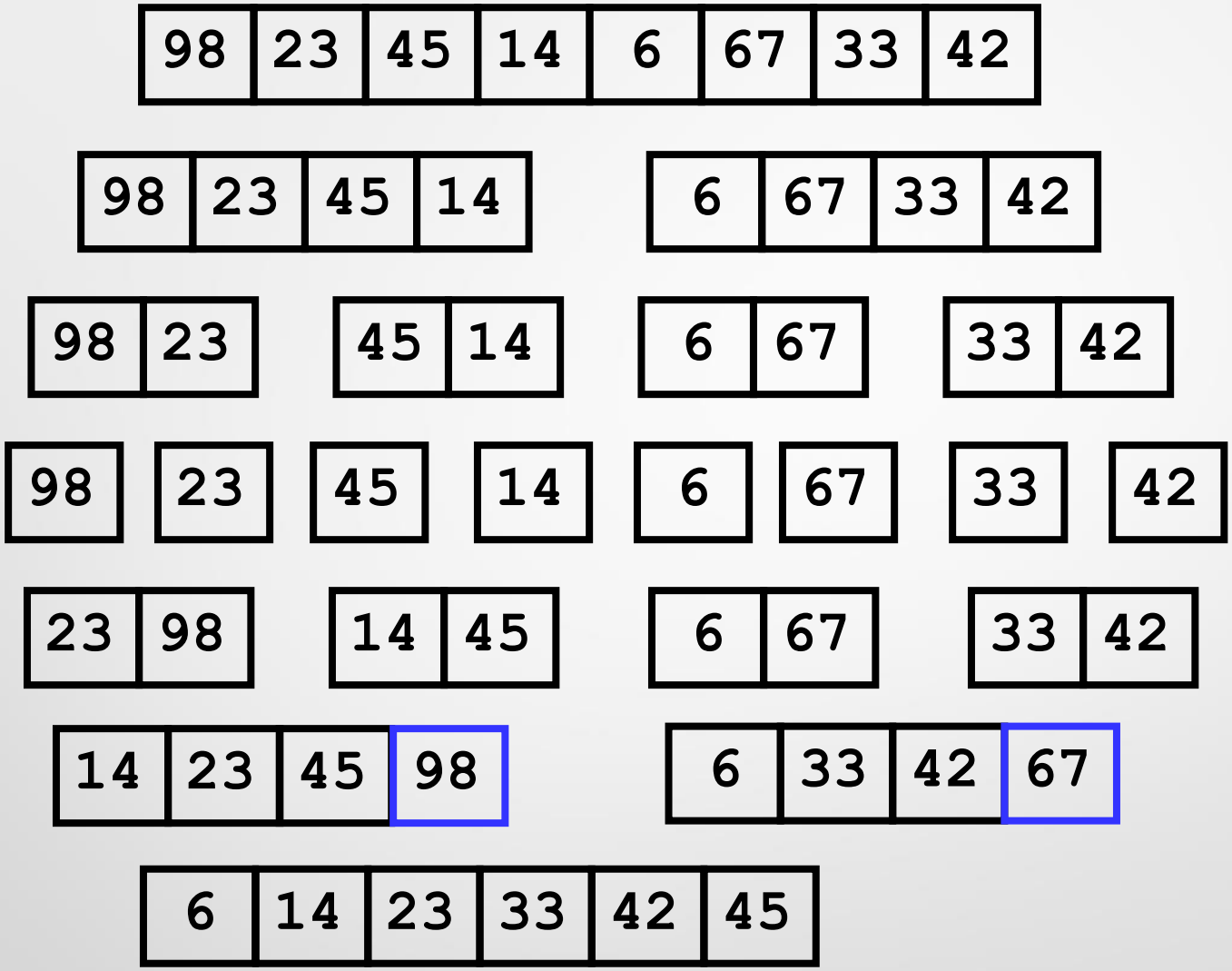

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6
Merge

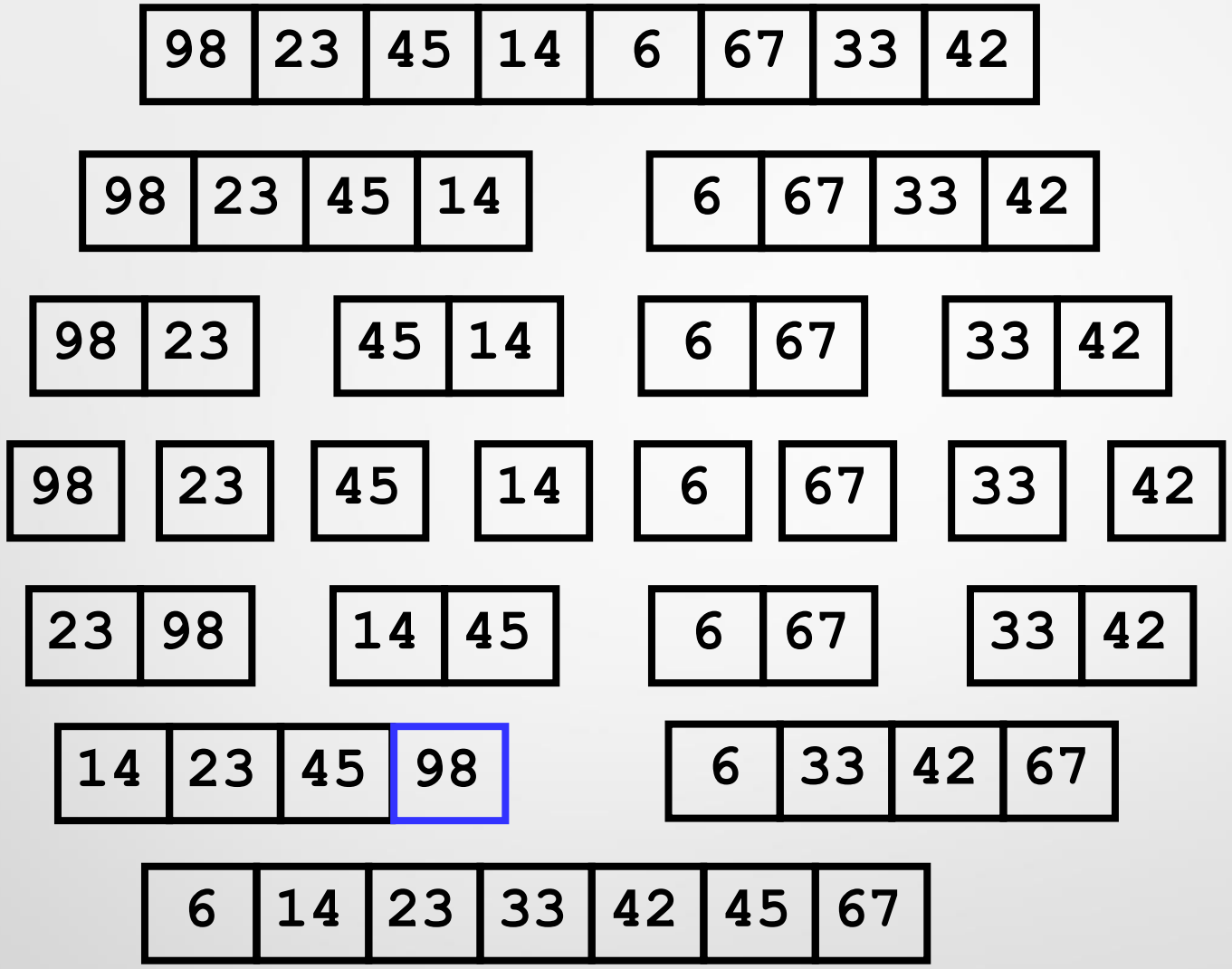

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14
Merge

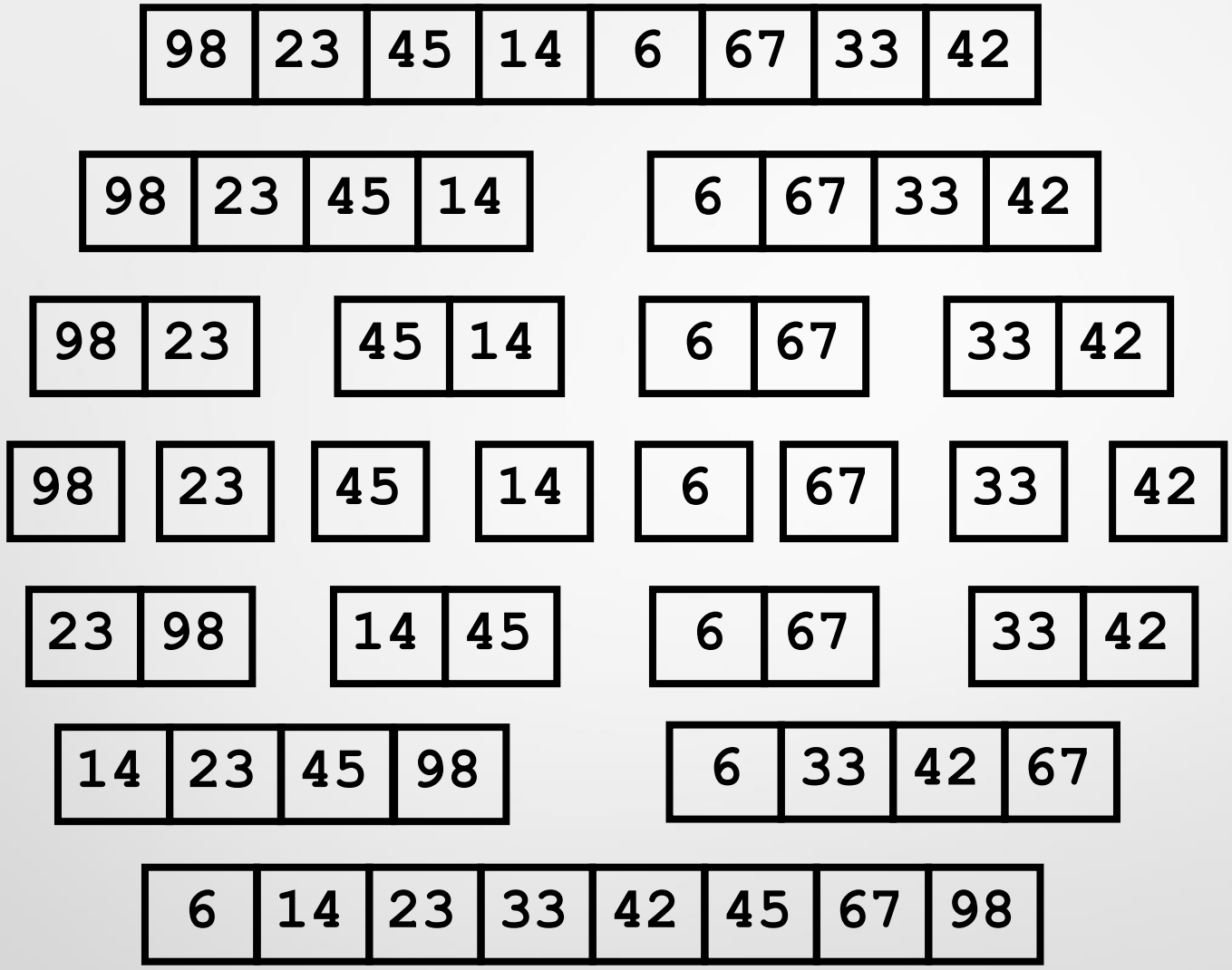

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23
Merge

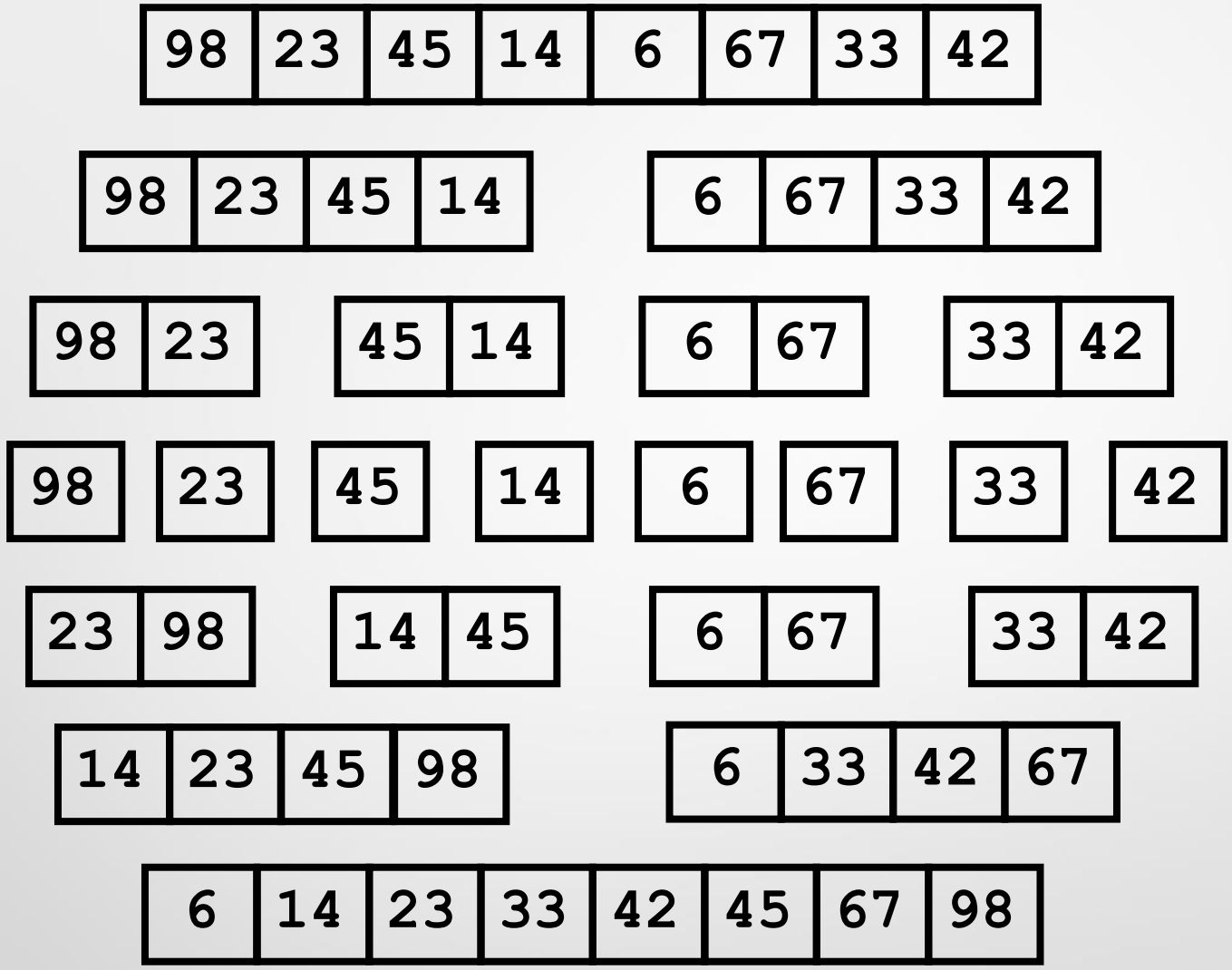

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33
Merge

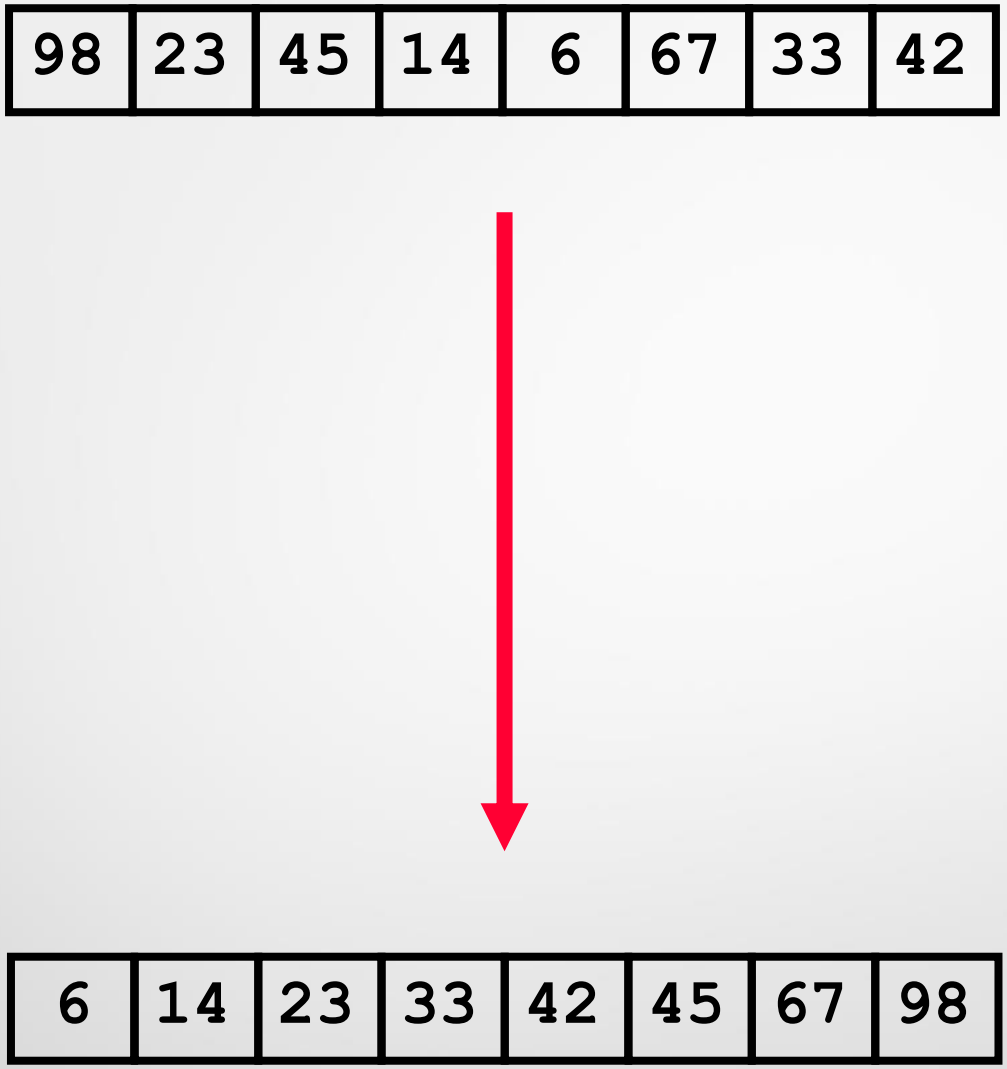

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98


98 23 45 14 6 67 33 42
6 14 23 33 42 45 67 98

Kesimpulan

- **Bagi array yang tidak terurut menjadi dua**
- **Sampai hanya subarray berisi satu elemen**
- **Selanjutnya gabungkan solusi sub problem bersama-sama**

Waktu Kompleksitas Mergesort

Kompleksitas waktu dari proses Rekursif.
T(n) adalah running time untuk worst-case untuk mengurutkan n data/bilangan.
Diasumsikan $n=2^k$, for some integer k.

```
void mergesort(vector<int> & A, int left, int right)
{
    if (left < right) {
        int center = (left + right)/2;
        mergesort(A,left,center);
        mergesort(A,center+1,right);
        merge(A,left,center+1,right);
    }
}
```

T(n/2)

T(n/2)

O(n/2+n/2=n)

Terdapat 2 rekursif merge sort, kompleksitas waktunya T(n/2)

Proses Merge memerlukan waktu O(n) untuk menggabungkan Hasil dari merge sort rekursif

$$\begin{cases} T(n) = 2T(n/2) + O(n) & n>1 \\ T(1)=O(1) & n=1 \end{cases}$$

Waktu Kompleksitas Mergesort

$$T(n)$$

Recursive step

$$= 2T(n/2) + O(n)$$

$$= 2(2T(n/4) + O(n/2)) + O(n)$$

Recursive step

$$= 4T(n/4) + 2 \cdot O(n/2)) + O(n)$$

$$= 4T(n/4) + O(2n/2)) + O(n)$$

$$= 4T(n/4) + O(n) + O(n)$$

Collect terms

$$= 4(2T(n/8) + O(n/4)) + O(n) + O(n)$$

Recursive step

$$= 8T(n/8) + 4 \cdot O(n/4) + O(n) + O(n)$$

$$= 8T(n/8) + O(4n/4) + O(n) + O(n)$$

$$= 8T(n/8) + O(n) + O(n) + O(n)$$

Collect terms

$$T(n) = 2^k T(n/2^k) + k \cdot O(n)$$

Setelah level ke - k

Waktu Kompleksitas Mergesort

$$T(n) = 2^k T(n/2^k) + k \cdot O(n)$$

Karena $n=2^k$, setelah level ke- k ($=\log_2 n$) pemanggilan rekursif, bertemu dengan (n=1)

Put $k= \log_2 n$, ($n=2^k$)
$T(n) = 2^k T(n/2^k) + kO(n) = nT(n/n) + kO(n)$
$= nT(1) + \log_2 nO(n) = nO(1) + O(n \log_2 n)$
$= O(n) + O(n \log_2 n)$
$= O(n \log_2 n) = O(n \log n)$

$$T(n) = O(n \log n)$$

Perbandingan insertion sort dan merge sort (dalam detik)

n	Insertion sort	Merge sort	Ratio
100	0.01	0.01	1
1000	0.18	0.01	18
2000	0.76	0.04	19
3000	1.67	0.05	33
4000	2.90	0.07	41
5000	4.66	0.09	52
6000	6.75	0.10	67
7000	9.39	0.14	67
8000	11.93	0.14	85

SEKIAN

Searching

Muh Izzuddin Mahali, M.Cs.

Topik

- Linear search
- Binary search

Pencarian (Searching)

- Pada suatu data seringkali dibutuhkan pembacaan kembali informasi (*information retrieval*) dengan cara searching.
- Searching adalah proses pencarian data yang ada pada suatu deret data dengan cara menelusuri data-data tersebut.
- Tahapan paling penting pada searching: memeriksa jika data yang dicari sama dengan data yang ada pada deret data.

Algoritma Pencarian

1. Input x (x=data yang dicari)
2. Bandingkan x dengan deret data
3. Jika ada data yang sama cetak pesan “Ada”
4. Jika tidak ada data yang sama cetak pesan “tidak ada”.

Algoritma Pencarian

- Macam algoritma pencarian :
  - Sequantial Search
  - Binary Search

(1) Sequential Search

- Disebut juga linear search atau Metode pencarian beruntun.
- Adalah suatu teknik pencarian data yang akan menelusuri tiap elemen satu per-satu dari awal sampai akhir.
- Data awal = tidak harus dalam kondisi terurut.

Algoritma Sequential Search

1. Input x (data yang dicari)
2. Bandingkan x dengan data **ke-i sampai n**
3. Jika ada data yang sama dengan x maka cetak pesan "Ada"
4. Jika tidak ada data yang sama dengan x cetak pesan "tidak ada"

Ilustrasi Sequential Search

- Misalnya terdapat array satu dimensi sebagai berikut:

0	1	2	3	4	5	6	7	indeks
8	10	6	-2	11	7	1	100	value

- Kemudian program akan meminta data yang akan dicari, misalnya **6 (x = 6)**.
- Iterasi :

  6 = 8 (tidak!)

  6 = 10 (tidak!)

  6 = 6 (Ya!) => output : "Ada" pada index ke-2
- Jika sampai data terakhir tidak ditemukan data yang sama maka output : " data yang dicari tidak ada".

Q & A

- **Problem:** Apakah cara di atas efisien? Jika datanya ada 10000 dan semua data dipastikan unik?
- **Solution**: Untuk meningkatkan efisiensi, seharusnya jika data yang dicari sudah ditemukan maka perulangan harus dihentikan!
  - **Hint**: Gunakan **break**!
- **Question**: Bagaimana cara menghitung ada berapa data dalam array yang tidak unik, yang nilainya sama dengan data yang dicari oleh user?
  - **Hint**: Gunakan variabel counter yang nilainya akan selalu bertambah jika ada data yang ditemukan!

Sequential Search with Sentinel

- Perhatikan array data berikut ini:

- Terdapat 6 buah data dalam array (dari indeks 0 s/d 5) dan terdapat 1 indeks array tambahan (indeks ke 6) yang belum berisi data (disebut sentinel)
- Array pada indeks ke 6 berguna untuk menjaga agar indeks data berada pada indeks 0 s/d 5 saja. Bila pencarian data sudah mencapai array indeks yang ke-6 maka berarti data TIDAK ADA, sedangkan jika pencarian tidak mencapai indeks ke-6, maka data ADA.

Best & Worst Case

- **Best case** : jika data yang dicari terletak di depan sehingga waktu yang dibutuhkan minimal.
- **Worst case** : jika data yang dicari terletak di akhir sehingga waktu yang dibutuhkan maksimal.
- Contoh :

  DATA = 5 6 9 2 8 1 7 4

  bestcase ketika x = 5

  worstcase ketika x = 4

  *x = key/data yang dicari

Latihan

- Buatlah flowchart dari algoritma Sequential Search!

(2) Binary Search

- Teknik pencarian = data dibagi menjadi dua bagian untuk setiap kali proses pencarian.
- Data awal harus dalam kondisi terurut. Sehingga harus dilakukan proses sorting terlebih dahulu untuk data awal.
- Mencari posisi tengah :

Posisi tengah = (posisi awal + posisi akhir) / 2

Algoritma Binary Search

1. Data diambil dari posisi awal 1 dan posisi akhir N
2. Kemudian cari posisi data tengah dengan rumus: **(posisi awal + posisi akhir) / 2**
3. Kemudian data yang dicari dibandingkan dengan data yang di tengah, apakah sama atau lebih kecil, atau lebih besar?
4. Jika data sama, berarti ketemu.
5. Jika lebih besar, maka ulangi langkah 2 dengan posisi awal adalah **posisi tengah + 1**
6. Jika lebih kecil, maka ulangi langkah 2 dengan posisi akhir adalah **posisi tengah – 1**

Ilustrasi

Contoh Data:

Misalnya data yang dicari **17**

- 01 2 3 4 5 6 7 8
- **39 11 12 15 17 23 31 35**
- **A B C**
- Karena 17 > 15 (data tengah), maka: awal = tengah + 1

- 01 2 3 4 5 6 7 8
- **39 11 12 15 17 23 31 35**
- **A B C**
- Karena 17 < 23 (data tengah), maka: akhir = tengah – 1

- 01 2 3 4 5 6 7 8
- **39 11 12 15 17 23 31 35**
- **A=B=C**
- Karena 17 = 17 (data tengah), maka KETEMU!

Best & Worst Case

- **Best case** : jika data yang dicari terletak di posisi tengah.
- **Worst case** : jika data yang dicari tidak ditemukan.
- Contoh :

  DATA = 5 6 9 2 8 1 7 4 3

  bestcase ketika x = 8 (T(n)=1)

  worstcase ketika x = 25 (T(n) = 5 atau n/2)

  *x = key/data yang dicari

Latihan

- Buatlah flowchart dari algoritma binary search!

Interpolation Search

- Teknik ini dilakukan pada data yang sudah terurut.
- Teknik searching ini dilakukan dengan perkiraan letak data.
  - Contoh ilustrasi: jika kita hendak mencari suatu nama di dalam buku telepon, misal yang berawalan dengan huruf T, maka kita tidak akan mencarinya dari awal buku, tapi kita langsung membukanya pada 2/3 atau ¾ dari tebal buku.
- Rumus posisi relatif kunci pencarian dihitung dengan rumus:

$$Posisi = \frac{kunci - data[low]}{data[high] - data[low]} x(high - low) + low$$

- Jika data[posisi] > data yg dicari, high = pos – 1
- Jika data[posisi] < data yg dicari, low = pos + 1

Kondisi Berhenti

- ada 2 kondisi berhenti :
  - jika data ditemukan
  - jika data tidak ditemukan, sampai data[low] lebih besar dari data yang dicari.

Kasus

- Misal terdapat data sebagai berikut:

Kode	Judul Buku	Pengarang
25	The C++ Programming	James Wood
34	Mastering Delphi 6	Marcopolo
41	Professional C#	Simon Webe
56	Pure JavaScript v2	Michael Bolton
63	Advanced JSP & Servlet	David Dunn
72	Calculus Make it Easy	Gunner Christian
88	Visual Basic 2005 Express	Antonie
96	Artificial Life : Volume 1	Gloria Virginia

Penyelesaian

- Kunci Pencarian ? **88**
- Low ? **0**
- High ? **7**
- Posisi = **(88 - 25) / (96 - 25) * (7 - 0) + 0 = [6]**
- Kunci[6] = kunci pencarian, data ditemukan : **Visual Basic 2005,** Antonie

- Kunci Pencarian ? **60**
- Low ? **0**
- High ? **7**
- Posisi = **(60 – 25) / (96 – 25) * (7 – 0) + 0 = [3]**
- Kunci[3] < kunci pencarian, maka teruskan
- Low = **3 + 1 = 4**
- High = **7**
- Ternyata Kunci[4] adalah **63** yang lebih besar daripada **60**.
- Berarti tidak ada kunci **60.**

Latihan

- lakukan pencarian dengan interpolation search dari deret data berikut :

  39 11 12 15 17 23 31 35

- x = 15 => low? high?
- x = 5 => low? high?
- x = 35 => low? high?

Latihan

- Buatlah flowchart dari algoritma interpolation search!

Pustaka

- Sartaj Sahni , “Data Structures & Algorithms”, Presentation L20-24.
- Mitchell Waite, “Data Structures & Algorithms in Java”, SAMS, 2001

SEKIAN

Searching

Muh Izzuddin Mahali, M.Cs.

Struktur Data Linear

- Adalah kumpulan komponen-komponen yang tersusun membentuk satu garis linear.
- Stack: struktur data linear dimana penambahan atau pengurangan komponen dilakukan di satu ujung saja.
- Queue: struktur data linear dimana penambahan komponen dilakukan di satu ujung, sementara pengurangan dilakukan di ujung lain (yang satu lagi).
- Kedua struktur tersebut merupakan struktur data abstrak dimana implementasi pada tingkat lebih rendah dapat menggunakan struktur *sequential* (array) atau struktur berkait (linear linked-list).

STACK (TUMPUKAN)

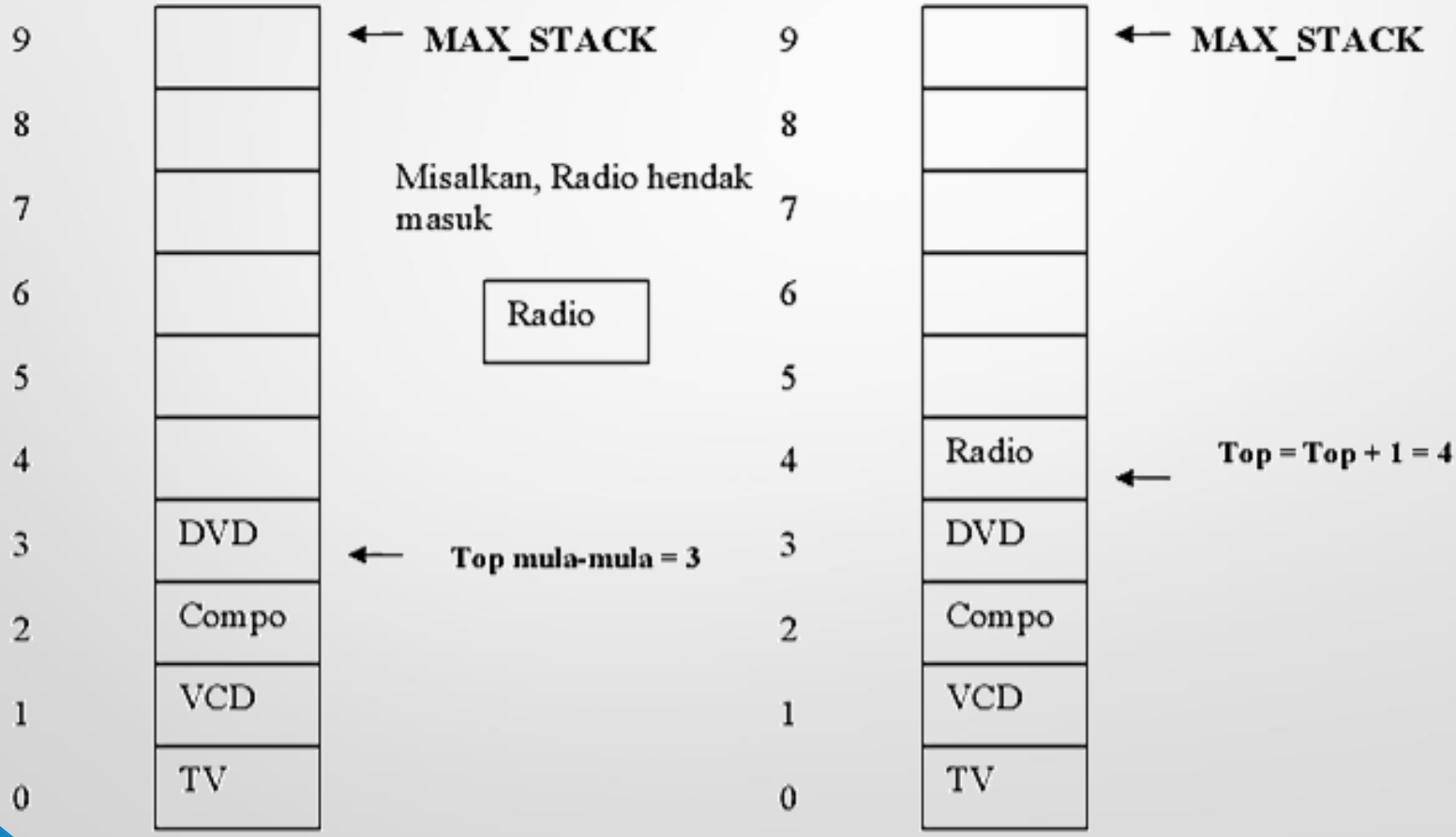
Stack (tumpukan) sebenarnya secara mudah dapat diartikan sebagai ***list (urutan) dimana penambahan dan pengambilan elemen hanya dilakukan pada satu sisi yang disebut top (puncak) dari stack.***

STACK (TUMPUKAN)

Dengan melihat definisi tersebut maka jelas bahwa pada stack berlaku aturan LIFO (Last In First Out), yaitu elemen yang terakhir masuk akan pertama kali diambil atau dilayani. Salah satu analogi yang dapat dikemukakan di sini adalah tumpukan piring atau barang lain. Pada saat kita hendak menumpuk piring-piring tersebut tentulah yang kita lakukan adalah meletakkan piring pertama pada tempatnya, selsnjutnya meletakkan piring kedua di atas piring pertama dan demikian seterusnya. Pada saat kita hendak mengambil satu piring dari tumpukan tersebut, tentu yang diambil adalah piring teratas (yang terakhir kali ditaruh), bukan yang terbawah (yang pertama kali diletakkan).

STACK (TUMPUKAN)

Ilustrasi stack



Operasi dalam stack

Operasi dasar pada stack adalah **PUSH** (operasi pemasukan elemen ke dalam stack) dan **POP** (operasi pengambilan satu elemen dari dalam stack).

- **Push** : digunakan untuk menambah item pada stack pada tumpukan paling atas
- **Pop** : digunakan untuk mengambil item pada stack pada tumpukan paling atas

Operasi tambahan

- **Clear** : digunakan untuk mengosongkan stack
- **IsEmpty** : fungsi yang digunakan untuk mengecek apakah stack sudah kosong
- **IsFull** : fungsi yang digunakan untuk mengecek apakah stack sudah penuh

Operasi dalam Stack

- Elemen paling kanan adalah elemen yang ada pada TOS (Top Of the Stack)
- stack yang dipakai bernama S
- PUSH(S,B) berarti memasukkan elemen B ke dalam stack S
- POP(B,S) berarti mengambil elemen dari stack S dan menaruhnya ke dalam variabel B

Operasi yang dilakukan	Isi Stack	Keterangan
Kondisi Awal	kosong	-
PUSH('A',S)	A	-
PUSH('B',S)	AB	-
PUSH('C',S)	ABC	-
POP(Data,S)	AB	Variabel Data berisi 'C'
PUSH('D',S)	ABD	-
POP(Data,S)	AB	Data berisi 'D'
POP(Data,S)	A	Data berisi 'B'

Operasi Stack

Implementasi dalam bahasa Pascal dapat dilakukan dengan memanfaatkan struktur data record dan array. Array dipergunakan untuk menyimpan elemen-elemen yang dimasukkan. Selain itu diperlukan pula suatu variabel untuk mencatat banyaknya elemen yang ada di dalam array yang sekaligus menunjukkan TOS (Top of Stack)

- konstanta *maxelm* menyatakan banyaknya elemen maksimum yang dapat ditampung oleh stack
- *typeelemen* adalah tipe data yang akan disimpan di dalam stack (bisa integer, word, real, boolean, char , string atau lainya)
- *banyak* adalah field yang menyatakan banyaknya elemen dalam stack saat itu, yang sekaligus menyatakan TOS

Implementasi dalam Pascal

Deklarasi tipe untuk tumpukan (stack):

```
type tumpukan = record
   atas :0..maxelm;
   isi : array[1..maxelm] of typeelemen;
end;
```

Implementasi dalam Pascal

Selain prosedur untuk POP dan PUSH, kita dapat pula menambahkan sejumlah fungsi untuk membantu penanganan kesalahan diantaranya adalah
fungsi **PENUHS** (untuk mengecek apakah stack penuh)
fungsi **KOSONGS** (untuk mengecek apakah stack kosong)
dan fungsi **SIZES** (untuk mengetahui banyaknya elemen di dalam stack).

Implementasi dalam Pascal

Fungsi SIZES

```
Function SIZES(S : tumpukan): integer;
begin
 SIZES := S.atas;
end;
```

Fungsi PENUHS

```
Function PENUHS(S : tumpukan): boolean;
begin
 Jika S.atas = maxelm then
    PENUHS := true
 else
    PENUHS :=false ;
end;
```

Fungsi KOSONGS

```
Function KOSONGS(S : tumpukan):boolean;
begin
 If S.atas = 0 then
    KOSONGS := true;
 else
    KOSONGS := false;
end;
```

Implementasi dalam Pascal

Procedure Push

```
procedure PUSH( var T:tumpukan; var
penuh:boolean;x:integer);
begin
    if T.atas = maxElm then penuh:=true
    else
    begin
       penuh := false;
       T.isi[T.atas]:=x;
       T.atas:=T.atas+1;
    end;
end;
```

Implementasi dalam Pascal

Procedure Pop

```
procedure POP( var T:tumpukan; var
habis:boolean;x:integer);
begin
    if T.banyak = 0 then habis:=true
    else
    begin
       habis := false;
       X:=T.isi[T.atas];
       T.atas:=T.atas-1
    end;
end;
```

QUEUE (ANTRIAN)

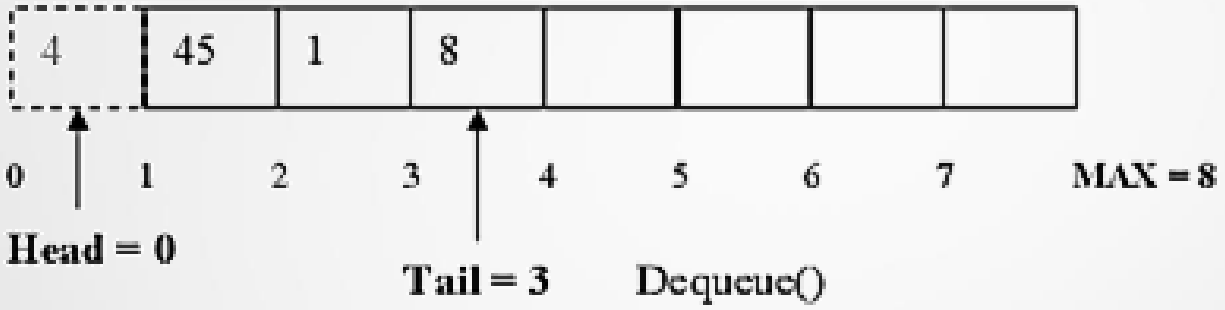
QUEUE

Queue atau antrian sebenarnya juga merupakan suatu list. Salah satu sifat yang membedakan queue dengan stack adalah bahwa pada queue penambahan elemen dilakukan pada salah satu ujung (ujung depan) dan pengambilan dilakukan pada ujung yang lain (ujung belakang) . Dengan demikian queue menggunakan prinsip **FIFO** (First In First Out), yaitu elemen yang pertama masuk akan pertama kali pula dikeluarkan.

Seperti pada stack, operasi-operasi dasar pada queue adalah operasi penambahan elemen (sebut "ADDQ") dan operasi pengambilan elemen (sebut DELQ). Di bawah ini diberikan contoh pemakaian operasi PUSH dan POP dan isi stack untuk setiap selesai eksekusi satu operasi.

QUEUE

Ilustrasi Queue



QUEUE

- elemen paling kanan adalah elemen yang ada pada ujung belakang (yang terakhir kali masuk)
- queue yang dipakai bernama Q
- ADDQ(Q,B) berarti memasukkan elemen B ke dalam queue Q
- DELQ(B,Q) berarti mengambil elemen dari queue Q dan menaruhnya ke dalam variabel B

Operasi yang dilakukan	Isi Queue	Keterangan
Kondisi Awal	kosong	-
ADDQ('A',Q)	A	-
ADDQ('B',Q)	AB	-
ADDQ('C',Q)	ABC	-
DELQ(Data,Q)	BC	Variabel Data berisi 'A'
ADDQ('D',Q)	BCD	-
DELQ(Data,Q)	CD	Data berisi 'B'
POP(Data,S)	D	Data berisi 'C'

Implementasi Queue

Implementasi dalam bahasa Pascal dapat dilakukan dengan memanfaatkan struktur data record dan array. Array dipergunakan untuk menyimpan elemen-elemen yang dimasukkan. Selain itu diperlukan pula suatu variabel untuk mencatat banyaknya elemen yang ada di dalam array. Pada implementasi

- konstanta *maxelm* menyatakan banyaknya elemen maksimum yang dapat ditampung oleh queue
- *typeelemen* adalah tipe data yang akan disimpan di dalam queue(bisa integer, word, real, boolean, char , string atau lainya)
- *Depan*, *belakang* adalah field yang menyatakan banyaknya elemen depan dan belakang dalam queue saat itu
- queue diimplementasikan sebagai array linier dengan memandang bahwa elemen terdepan selalu berada pada sel pertama (implementasi fisik), sehingga bila terjadi pengambilan satu elemen maka semua elemen di belakang elemen terambil (bila ada) harus digeser ke depan satu langkah

Deklarasi tipe untuk antrian (queue):

```
type antrian= record
   depan,belakang :0..maxelm;
   isi : array[1..maxelm] of typeelemen;
end;
```

Implementasi Queue dalam Pascal

Selain prosedur untuk ADDQ dan DELQ, kita dapat pula menambahkan sejumlah fungsi untuk membantu penanganan kesalahan diantaranya adalah fungsi **PENUHQ** (untuk mengecek apakah antrian penuh) fungsi**KOSONGQ** (untuk mengecek apakah antrian kosong) dan fungsi **SIZEQ** (untuk mengetahui banyaknya elemen di dalam queue).

Fungsi PENUHQ

```
Function PENUHQ(q : antrian): boolean;
begin
 Jika Q.banyak = maxelm then PENUHQ := true
 else PENUHQ := false;
end;
```

Fungsi KOSONGQ

```
Function KOSONGQ(q : antrian):boolean;
begin
 If Q.banyak = 0 then KOSONGQ := true
 else KOSONGQ := false;
end;
```

Implementasi Queue dalam Pascal

Procedure ADDQ

```
procedure ADDQ(data:integer; var q:queue);
var sisip :boolean;
i,j,pos:integer;
begin
    sisip:=false;
    i:=q.depan;
    while (q.isi[i]<>0) and (data>=q.isi[i]) do inc(i);
    if data<q.isi[i] then
       begin
           pos:=i;
           for j:=q.belakang downto pos do
           q.isi[j+1]:=q.isi[j];
           q.isi[pos]:=data;
           inc(q.belakang);
       end
    else
       if q.belakang<max then
       begin
           inc(q.belakang);
           q.isi[q.belakang]:=data;
       end;
end;
```

Implementasi Queue dalam Pascal

Prosedur DELQ

```
Procedure DeQueue(var q:queue; var hsl:integer);
var
   i:integer;
begin
   if q.belakang>0 then
   begin
      hsl:=q.isi[q.depan];
      dec(q.belakang);
      for i:=1 to q.belakang do
       q.isi[i]:=q.isi[i+1] ;
   end;
end;
```

Latihan

1. Tambahkanlah function untuk mencari suatu elemen dalam queue & stack
2. Tambahkan function untuk mengedit suatu elemen dalam queue & stack
3. Carilah nilai total, rata-rata, terbesar dan terkecil dari elemen-elemen queue dalam function tersendiri

S E K I A N

Algoritma dan Struktur Data
Pointer

Muh. Izzuddin Mahali, M.Cs.

PT. Elektronika FT UNY

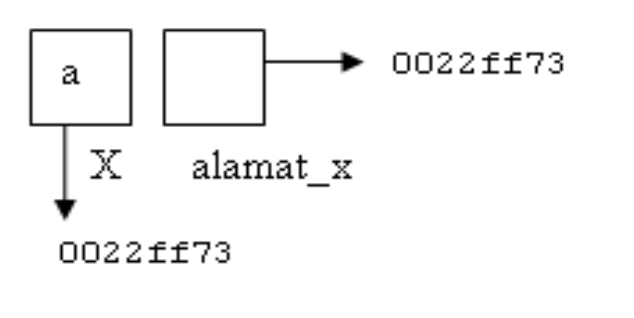
Pendahuluan

Cakupan bahasan

1. Mengenal tipe data Pointer.
2. Manipulasi memori lewat Pointer bertipe dan tak bertipe.
3. Linked List; meliputi operasi inisialisasi, menambah node baru, menyisipkan node baru, menghapus node yang berisi data, membaca data dari node.

Pendahuluan

Pointer merupakan suatu tipe data dalam Pascal yang berfungsi untuk menunjuk dan menyimpan alamat memori. Dalam penulisan pointer biasa digambar dengan panah, sedangkan bagian memori yang ditunjuk digambar dengan kotak, dan isinya ditulis di dalam kotak.



Ilustrasi Pointer

- Kita memiliki variabel X yang berisi nilai karakter ‘a’
- Oleh kompiler pascal, nilai ‘a’ ini akan disimpan di suatu alamat tertentu di memori.
- Alamat variabel X dapat diakses dengan menggunakan statemen **&X**.
- Jika kita ingin menyimpan alamat dari variabel X ini, kita dapat menggunakan suatu variabel

  misalnya **char alamat_x = &X;**
- alamat_x adalah suatu variabel yang berisi alamat dimana nilai X, yaitu ‘a’ disimpan.
- Variabel alamat_x disebut variabel pointer atau sering disebut **pointer** saja.

Deklarasi Pointer

Bentuk umum dari deklarasi tipe pointer:

Untuk pointer **bertipe**:

```
<nama_var> : ^<tipe_data>;
```

Untuk pointer **tidak bertipe**:

```
<nama_var> : pointer;
```

Suatu pointer dapat menunjuk ke data **bertipe** *elementer*, *terstruktur*, *pointer yang lain*, atau *tidak bertipe*. Jika suatu pointer *tidak menunjuk ke mana-mana*, pointer itu dinamakan *dangling*, sedangkan bagian memori yang tidak dapat diakses karena tidak ada pointer yang menunjuk dinamakan *garbage* (sampah)

Deklarasi Pointer

Dalam Pascal, pointer dapat diisi dengan nilai yang berasal dari:\

1. NIL
2. Fungsi Ptr
3. Operator @
4. Prosedur New dan GetMem
5. Pointer yang lain

NIL

Reserved word NIL

NIL merupakan reserved word dalam Pascal, di mana pointer yang bernilai NIL dianggap tidak menunjuk alamat memori manapun.NIL biasa digambarkan dengan lambang ground

Fungsi Ptr

Fungsi Ptr mengembalikan pointer dari segmen dan offset yang dimasukkan.

Sintaks:

```
Function Ptr(Seg, Ofs : word) : pointer;
```

dengan *Seg* : segmen memori.

Ofs : offset memori.

Operator @

Operator @ digunakan untuk mengambil alamat variabel yang akan ditunjuk.

Sintaks:

```
<nama_var>:=@<variabel_yang_alamatnya_diambil>;
```

Prosedur New dan GetMem

Prosedur **New** digunakan untuk memesan memori untuk *pointer bertipe*, sedangkan prosedur **GetMem** untuk *pointer tidak bertipe*. Kedua prosedur ini akan membentuk suatu variabel dinamik yang diletakkan dalam *Heap*. **Heap** adalah memori-memori di komputer yang belum dialokasikan, yaitu memori yang tidak digunakan oleh DOS, oleh program-program resident, oleh program Turbo Pascal, internal stack yang digunakan oleh Turbo Pascal dan variabel-variabel di data segmen

Sintaks:

```
New(var P : pointer);
GetMem(var P : pointer, size : word);
```

Dengan *P* : pointer yang akan diisi.

Size : ukuran yang dipesan.

- Pointer yang belum digunakan sebaiknya diisi dengan NIL, dan untuk pointer
- yang telah menunjuk sebuah alamat yang sudah dipesan memorinya, isinya dapat
- dimanipulasi melalui pointer.

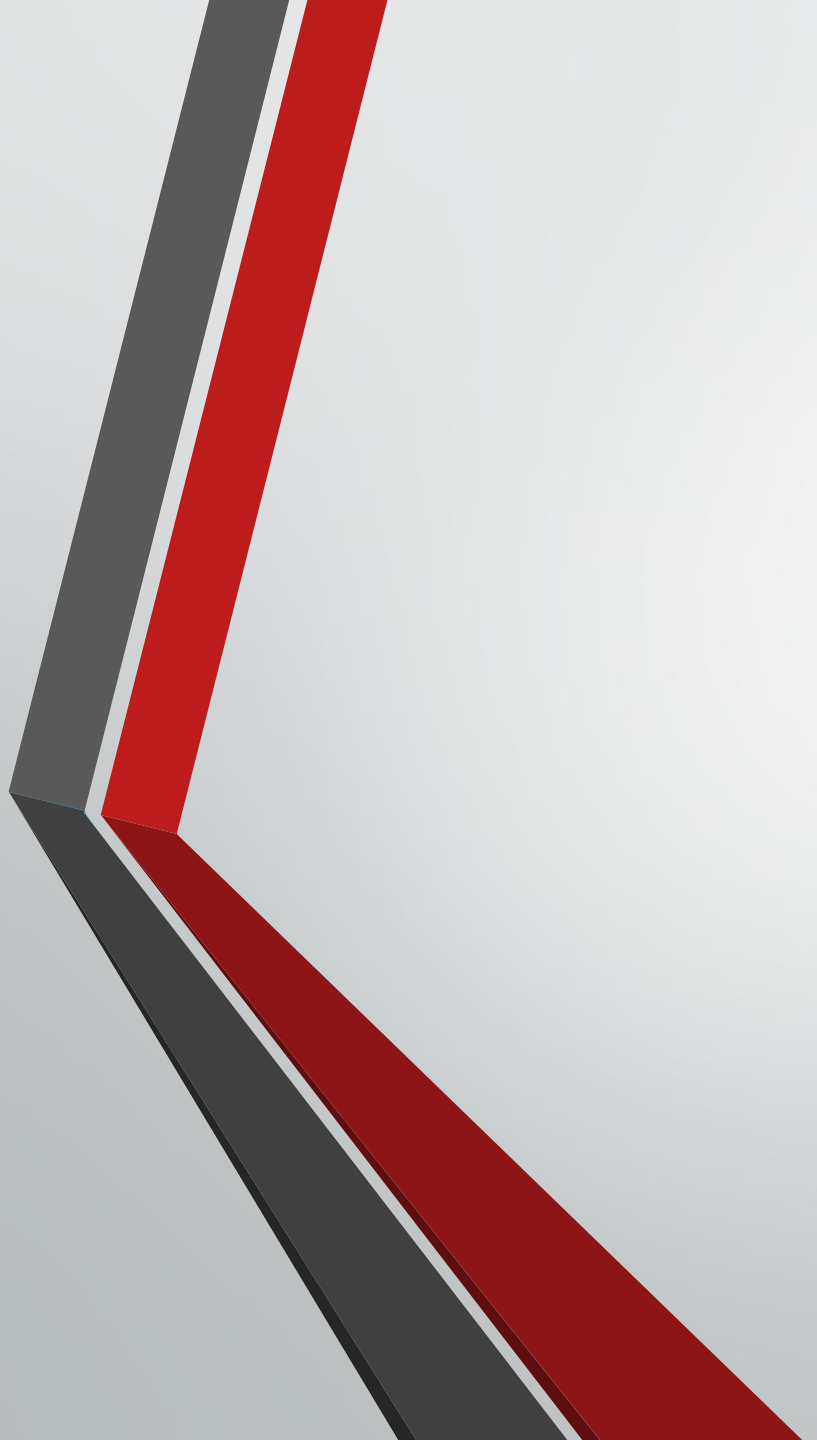
Contoh Penggunaan Pointer

```
program deklarasi;
   uses crt;
var
   p : ^integer;
   nilai : integer;
begin
clrscr;
   nilai:=12;
   p:=@nilai;
   writeln(p^);
   p^:=100;
   writeln(p^);
   writeln(nilai);
   readln;
end.
```

Pembahasan

Pada contoh program deklarasi ini, pertama-tama dideklarasikan variabel *p* sebagai pointer yang bertipe integer. Dibuat sebuah variabel lagi yang diberi nama *nilai* dan bertipe integer.

Variabel nilai diisi dengan nilai 12. Kemudian variabel p menunjuk alamat dari variabel nilai dengan operator @, sehingga variable p berisi nilai 12, dan ditampilkan outputnya di layar. Kemudian variabel p diberi nilai 100, dan secara otomatis variabel nilai juga bernilai 100 karena sudah ditunjuk oleh variabel p. Kemudian isi dari variabel p yang baru dan variabel nilai ditampilkan di layar.



LinkedList

PENDAHULUAN

- Dalam suatu linear list kita dapat melakukan operasi penyisipan atau penghapusan atas elemen-elemennya pada sembarang posisi.
- Misalkan ada 1500 item yang merupakan elemen dari suatu linear list.
- Jika elemen ke-56 akan kita keluarkan, maka elemen ke-1 s/d elemen ke-55 tidak akan berubah posisinya pada linear list tersebut. Tetapi elemen ke-57 akan menjadi elemen ke-56, elemen ke-58 akan menjadi elemen ke-57 dst. Selanjutnya, jika kita sisipkan satu elemen pada posisi setelah elemen ke-41, maka elemen ke-42 s/d elemen ke-1500 akan berubah posisinya.
- Untuk menyatakan keadaan diatas diperlukan suatu konsep yang berbeda dengan konsep sekuensial sebelumnya.
- Linked list merupakan suatu cara non-sekuensial yang digunakan untuk merepresentasikan suatu data.

DEFINISI

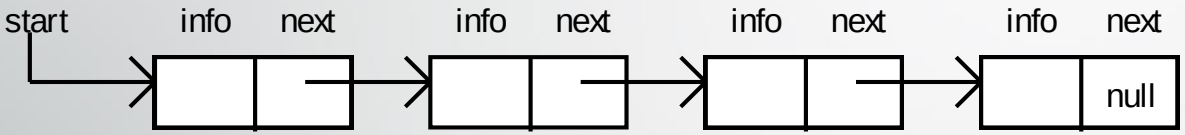
Linked list (one way list) adalah suatu kumpulan elemen data (yang disebut sebagai node) dimana urutannya ditentukan oleh suatu pointer.

Setiap elemen (node) dari suatu linked list terdiri atas dua bagian, yaitu :

- INFO , berisi informasi tentang elemen data yang bersangkutan.
- NEXT (link field/next pointer field), berisi alamat dari elemen node) selanjutnya yang dituju.

Berikut ini sebuah contoh linked list yang terdiri atas 4 node :

DEFINISI (1)



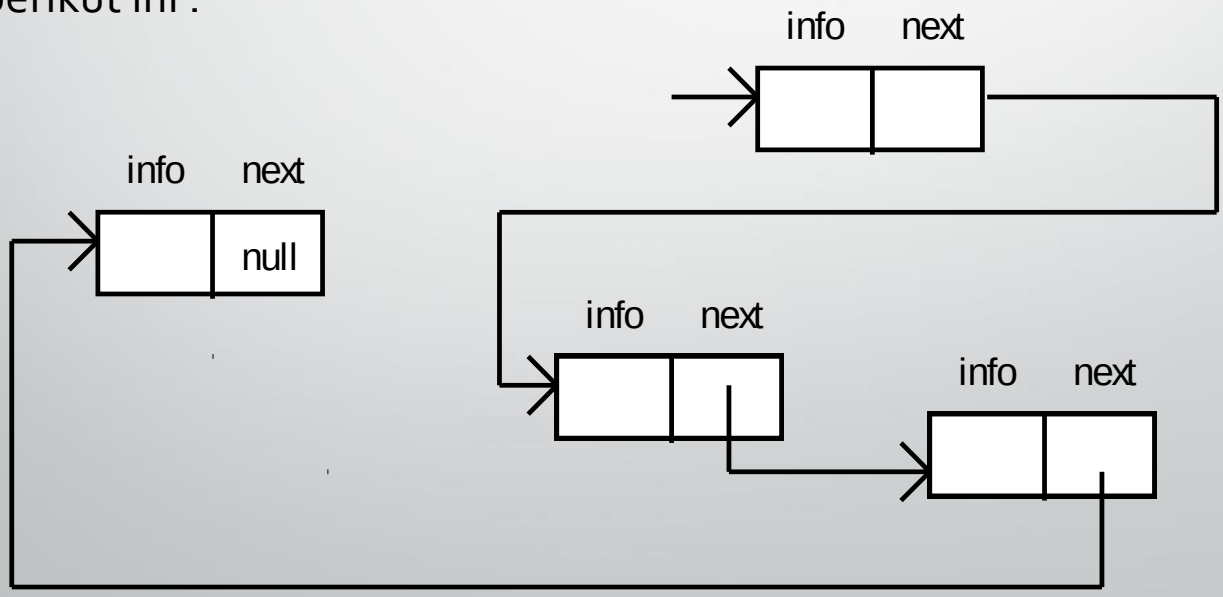
Berikut ini sebuah contoh linked list yang terdiri atas 4 node :

node ke-1 node ke-2 node ke-3 node ke-4

Pada node ke-4 field NEXT-nya berisi NULL, artinya node ke-4 tsb. adalah node terakhir

DEFINISI (2)

Node-node dalam linked list tidak harus selalu digambarkan paralel seperti pada gambar diatas. Linked list pada contoh diatas dapat pula digambarkan seperti berikut ini :



DEFINISI (3)

CATATAN :

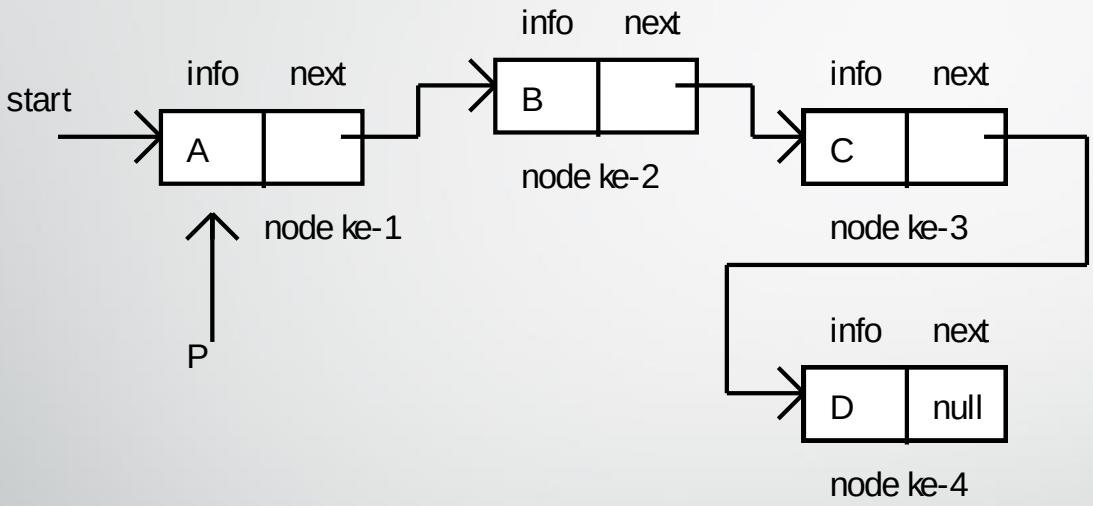
- Ada dua hal yang menjadi kerugian dengan representasi suatu data dengan linked list ini,
  yaitu :
  1. Diperlukan ruang tambahan untuk menyatakan/tempat field pointer.
  2. Diperlukan waktu yang lebih banyak untuk mencari suatu node dalam linked list.
- Sedangkan keuntungannya adalah :
  1. Jenis data yang berbeda dapat di-link.
  2. Operasi REMOVE atau INSERT hanya dilakukan dengan mengubah pointer-nya saja.

OPERASI DASAR PADA LINKED LIST

Ada beberapa aturan yang didefinisikan pada operasi didalam linked list, yaitu :
- Jika P adalah suatu variabel pointer, maka nilainya adalah alamat atau lokasi dari variabel lain yang dituju.
- Operasi yang didefinisikan pada suatu variabel pointer adalah :
    1. Test apakah sama dengan NULL.
    2. Test untuk kesamaan dengan variabel pointer lain.
    3. Menetapkan sama dengan NULL.
    4. Menetapkan menuju ke node lain.

Notasi yang didefinisikan sehubungan dengan operasi diatas adalah :
    1. NODE(P), artinya node yang ditunjuk oleh pointer P.
    2. INFO(P), artinya nilai INFO dari node yang ditunjuk pointer P.
    3. NEXT(P), artinya hubungan (link) selanjutnya dari node yang ditunjuk oleh pointer P

OPERASI DASAR PADA LINKED LIST



NODE(P) = node yang ditunjuk oleh P yaitu node pertama.

INFO(P) = A

NEXT(P) = node ke-dua

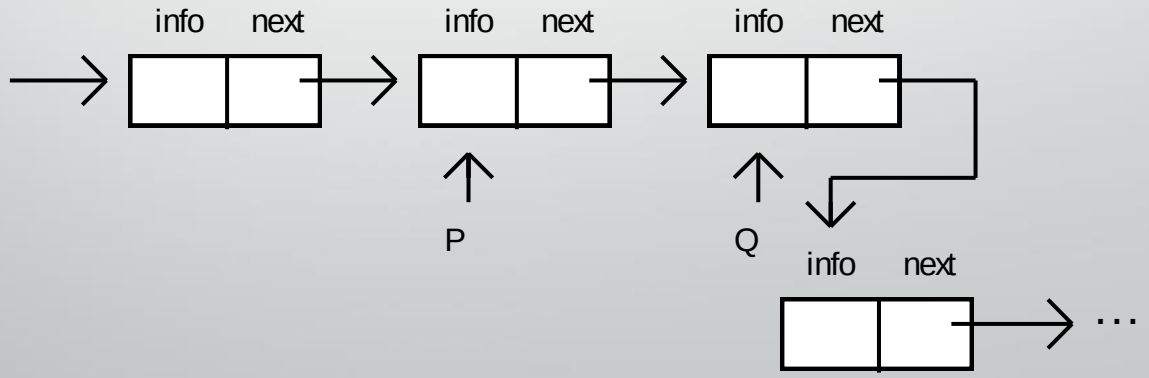
INFO(NEXT(NEXT(P))) = C

MENGHAPUS SUATU NODE DARI LINKED LIST (REMOVE)

- Untuk menghapus node dalam linked list digunakan procedure FREENODE.
- Jika Q adalah suatu variabel pointer, maka FREENODE(Q) akan menyebabkan node yang ditunjuk oleh variabel pointer Q dihapus dari linked list.
- Perhatikan linked list berikut :

langkah ke-1 :

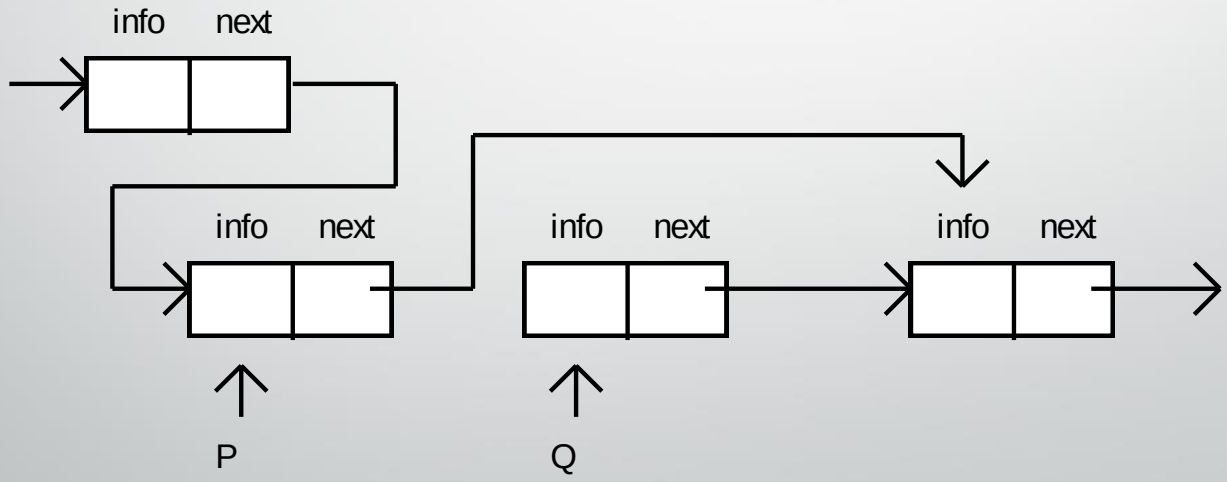
Q := Next(P)



MENGHAPUS SUATU NODE DARI LINKED LIST (REMOVE)

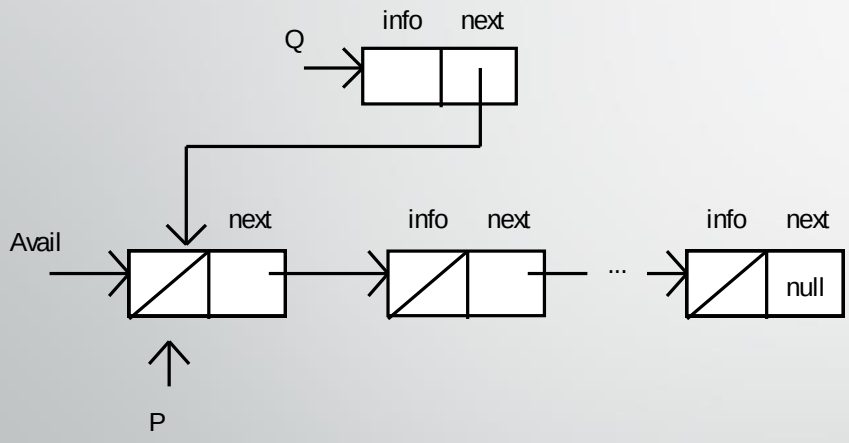
langkah ke-2 :

Next(P) := Next(Q)

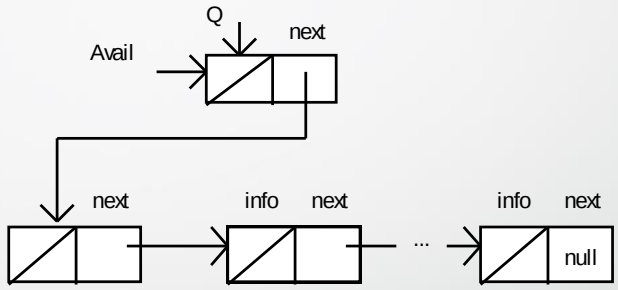


procedure Freenode(Q)

(a) Next(Q) := Avail



(b) Info(Q) := Null



(c) Avail := Q

MENYISIPKAN SUATU NODE KE DALAM LINKED LIST

- Untuk menyisipkan node dalam linked list digunakan procedure GETNODE.
- Jika NEW adalah suatu variabel pointer, maka GETNODE(NEW) akan menyebabkan node yang ditunjuk oleh variabel pointer NEW disisipkan ke dalam linked list.

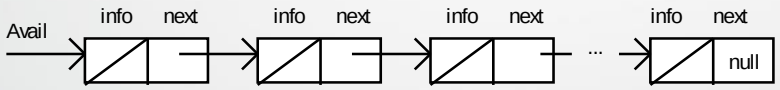
procedure Getnode(NEW)

if Avail = Null

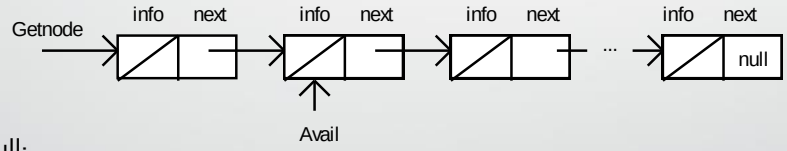
then out-of-free-space

(a) else begin

Getnode := Avail;

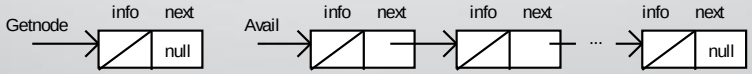


(b) Avail := Next(Avail);



(c) Next(Getnode) : = Null;

end;



Logika Linked List pada Array

(a) Jika tidak menggunakan logika linked list

(pada umumnya dalam meng-input data digunakan cara c sequential)

	Awal		Insert E		Delete C		Insert F
1	A	1	A	1	A	1	A
2	C	2	C	2		2	
3		3	E	3	E	3	E
4		4		4		4	F

Jika menggunakan logika Linked List

Keadaan awal

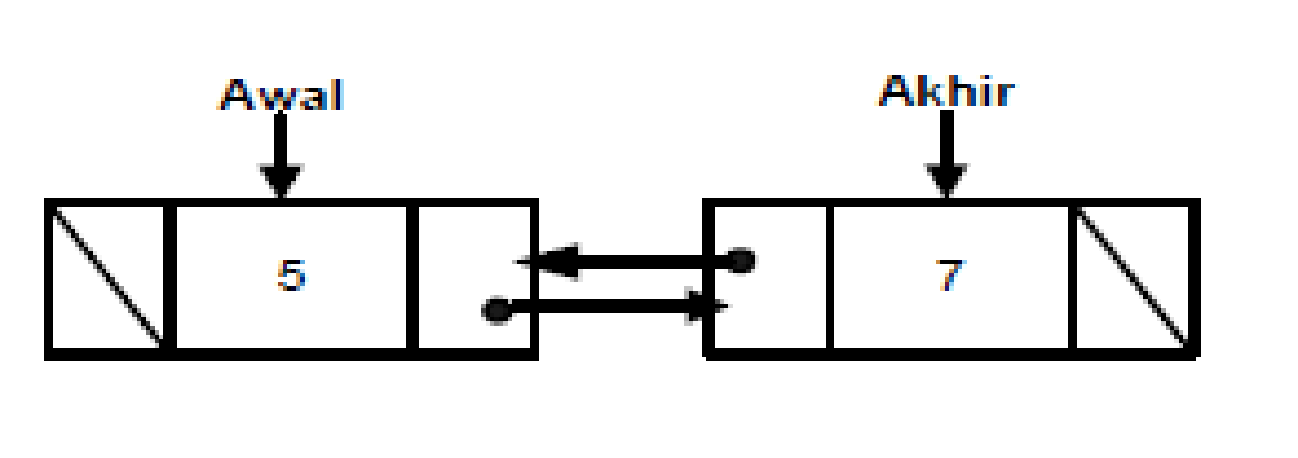
	Info	Next
1	A	2
2	C	0
3		4
4		0

Insert E

	Info	Next
1	A	2
2	C	3
3	E	0
4		0

Delete C

	Info	Next
1	A	3
2		4
3	E	0
4		0


Awal
Akhir
5
7

S E L E S A I