Algoritma Divide And Coquer Sorting

wijanarto

Definisi

- *Divide*: membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),
- *Conquer*: memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan
- *Combine:* mengabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

Outline

- Min Max
- Perkalian 2 Digit Besar
- Sorting
  - Insertion Sort
  - Selection Sort
  - Merge Sort
  - Quick Sort

Min dan Max

Pseudocode

```
Procedure MaxMin(x, awal, akhir, Max, Min) Begin
        if akhir-awal ≤ 1 then
            if x(awal) ≥ x(akhir) then
                    Max ← x(awal), Min ← x(akhir)
            else Max ← x(akhir), Min ← x(awal)
        else Begin
            Mid ← (awal + akhir) DIV 2
            MaxMin(x, awal, mid, Max1, Min1)              devide
            MaxMin(x, mid+1, akhir, Max2, Min2)
            if Max1 ≥ Max2 then
                    Max ← Max1
            else Max ← Max2
            if Min1 ≤ Min2 then                            conquer
                    Min ← Min1
            else Min ← Min2
        end
end
```

Min dan Max

- MaxMin dengan metode biasa → g(n)= 2n-1
- MaxMin dengan metode devide and conquer → g(n)= $\begin{cases} 1 & , utk\ n < 2 \\ \underbrace{2g\left(\frac{n}{2}\right)}_{\text{Rekursif}} + 2 & , \underbrace{utk\ n > 2}_{\text{conquer}} \end{cases}$
- Rekursif conquer

Contoh Mencari g(n)

- n adalah power of 2

$$\begin{cases} 1 & ,utk\, n \le 2 \\ 2g\left(\frac{n}{2}\right) + 2 & ,utk\, n > 2 \end{cases}$$

N	g(n)
2	1
4	2g(2)+2 = 4
8	2.4+2 = 10
16	2.10+2 = 22
32	2.22+2 = 46
64	2.46+2 = 94
n	n − 2

Perkalian 2 Bilangan Besar n Digit

- Misal n=4
- x = 6789
- y = 2476   x
- z = ... ?
- Problem Size = n
- Operasi Dominan = perkalian
- algoritma biasa $g(n) = n^2 + cn \rightarrow O(n^2)$

Dengan metode devide and conquer

- a=67 b=89 $x=67*10^2 + 89 = a.10^{n/2} + b$
- c=24 d=76 $y=24*10^2 = 76 = c.10^{n/2} + d$

X	a	b
Y	c	d

- z = ...
- $z = x.y = (a.10^{n/2} + b).(c.10^{n/2} + d)$
- $z = ac.10^n + (ad+bc)\ 10^{n/2} + bd$
- $g(n) = 4g(\frac{n}{2}) + cn \rightarrow O(n^2)$ →berdasarkan teorema
- 4 kali rekursif (ac, ad, bc, bd)

Perkalian 2 Bilangan Besar n Digit

$O(n^2)$ tidak berubah menjadi lebih efisien, maka conquer perlu diubah
pseudo code

```
Begin
        u ← (a+b).(c+d)
        v ← ac
        w← bd
        z ← v.10^n + (u–v–w) 10^(n/2) + w
End
```

maka $g(n) = 3g(\frac{n}{2}) + cn \quad \rightarrow O(n^{\log 3}) = O(n^{1,59})$

TEOREMA MENCARI O(f(n))

- jika g(n) = $\begin{cases} b & ,utk\, n \le n0 \\ ag\left(\frac{n}{c}\right) + bn & ,utk\, n > n0 \end{cases}$

- Maka g(n)= $\begin{cases} \Theta(n) & jika\, a < c \\ \Theta(n \log n) & jika\, a = c \\ \Theta(\log_c a) & jika\, a > c \end{cases}$

Algoritma Sorting

- Mengurutkan data dilihat dari struktur data dapat di bagi menjadi dua bagian yaitu statis (larik/array) dan dinamis (pointer). Dari dua macam cara penyimpanan data tersebut masing-masing mempunyai keuntungan dan kerugian baik dalam hal kecepatan, efisiensi dan aspek kapasitas memori.

Insertion Sort $O(n^2)$

- Idenya seperti pemain kartu yang membagi elemen menjadi 2 kelompok, yaitu kelompok kartu yang terurut berada di tangan pemain, dan kelompok kartu sumber yang akan diambil untuk disisipkan secara urut ke dalam kelompok kartu pertama.

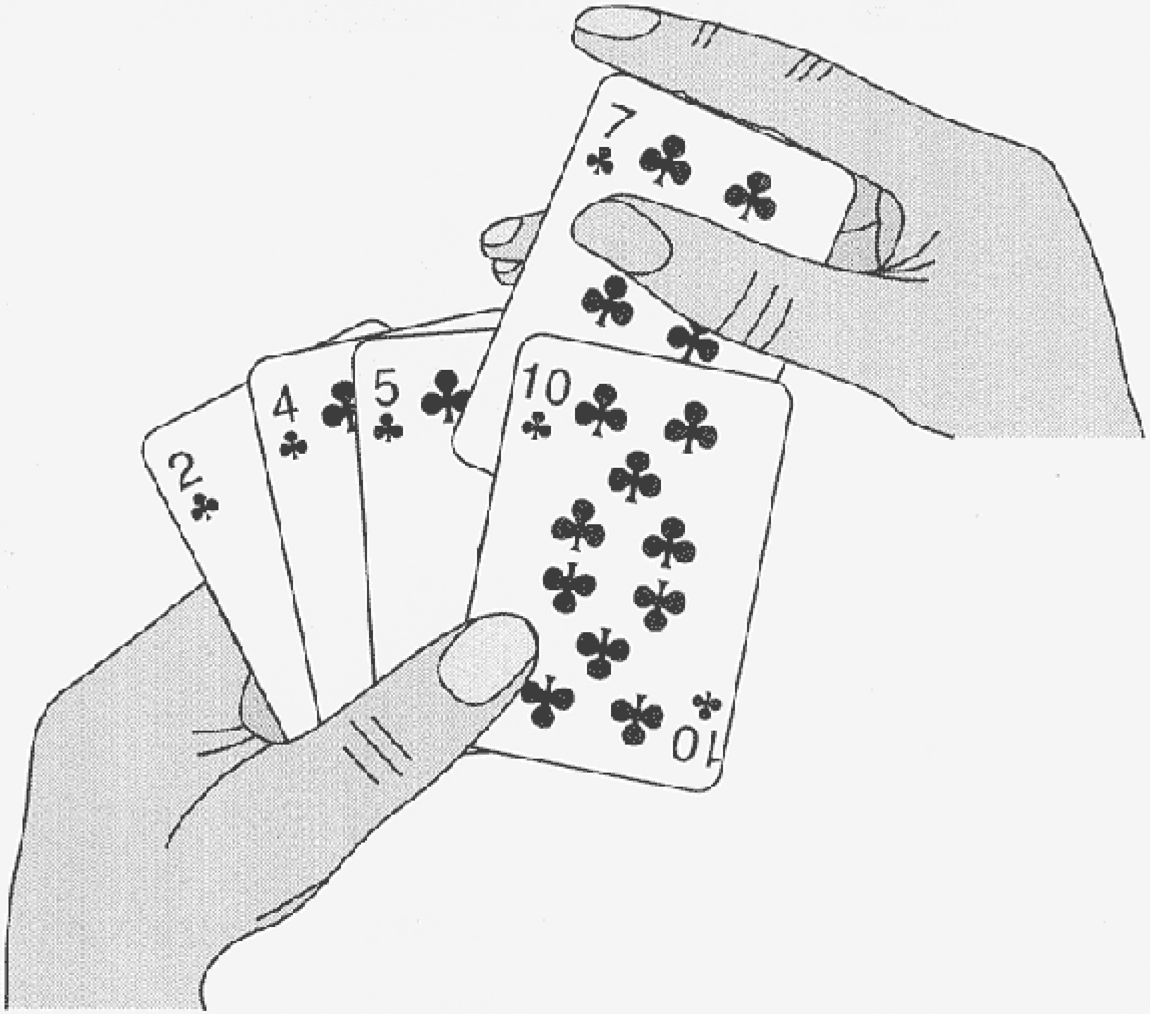


Figure 2.1 Sorting a hand of cards using insertion sort.

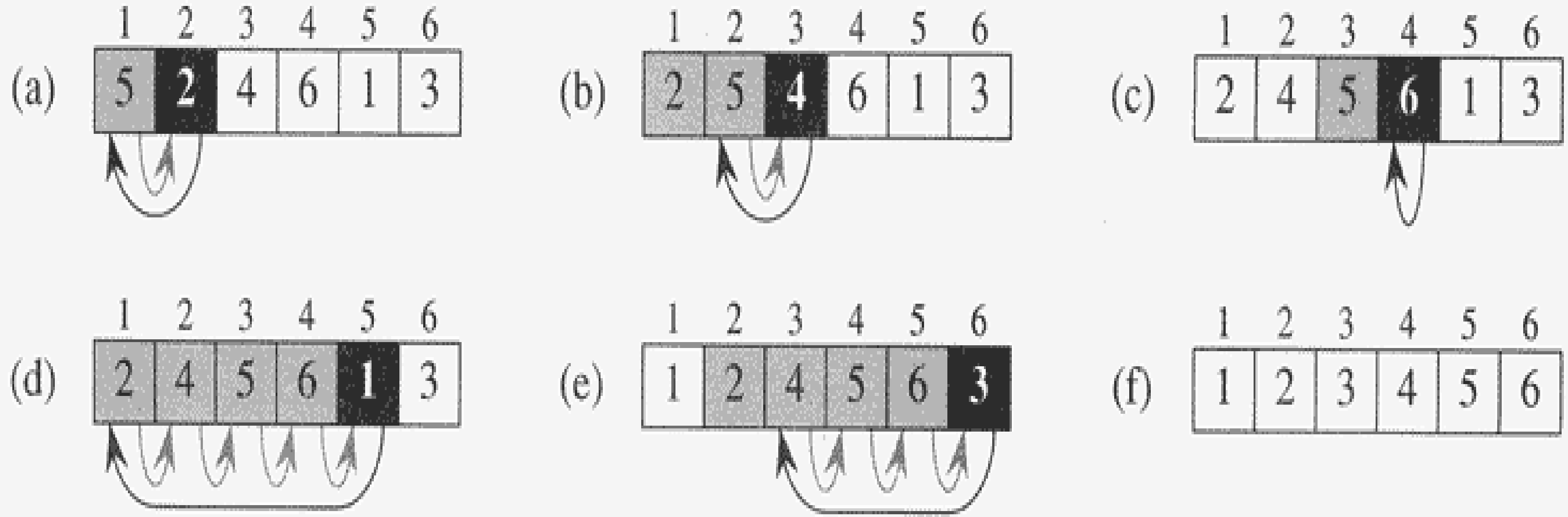


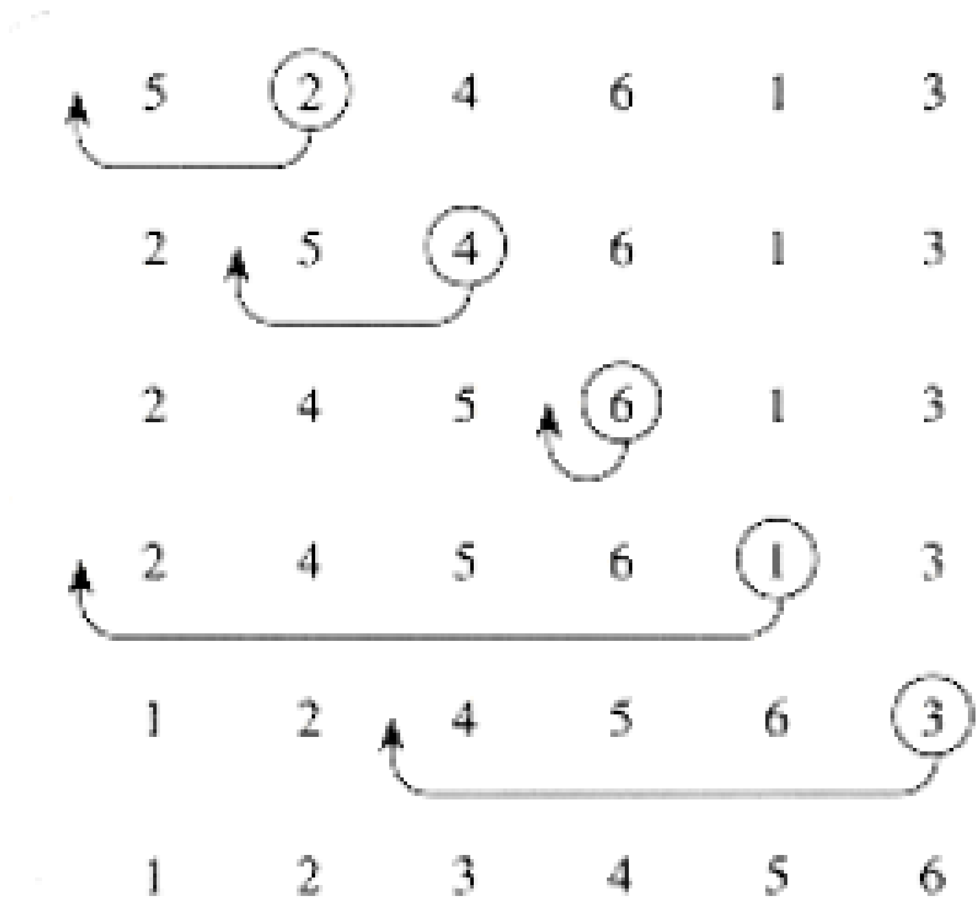
Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key is moved to in line 8. **(f)** The final sorted array.

```
INSERTION-SORT(A)
for j ← 2 to length[A]
     do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
        i ← j − 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
               i ← i − 1
        A[i + 1] ← key
```

Loop invariants and the correctness of insertion sort

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Contoh 1



Contoh 2

Data Sumber : [8, 4, 7, 3, 1, 2, 6, 5]

Index : 1 2 3 4 5 6 7 8

Data Sumber : [8, 4, 7, 3, 1, 2, 6, 5]
Index : 1 2 3 4 5 6 7 8

1. Ambil data mulai index 1 (8)
2. Ambil data index 2 (4), lalu bandingkan dengan nilai sebelumnya, jika lebih kecil dari sebelumnya, taruh di kiri dan jika tidak (lebih besar) taruh di kanan , dr contoh diatas maka susunannya menjadi [4, 8]

Data Sumber : [4, 8, 7, 3, 1, 2, 6, 5]

Index : 1 2 3 4 5 6 7 8

- Ambil data index 3 (7), bandingkan dengan data index sebelumnya (4,8), 7>4 tapi lebih 7<8 sehingga susunannya menjadi [4, 7, 8]

Data Sumber : [4, 7, 8, 3, 1, 2, 6, 5]
Index : 1 2 3 4 5 6 7 8

- Ambil data index 4 (3), bandingkan dengan data index sebelumnya (4,7,8) dan susunanya menjadi [3, 4, 7, 8]

Data Sumber : [3, 4, 7, 8, 1, 2, 6, 5]
Index : 1 2 3 4 5 6 7 8

- Ambil data index 5 (1), bandingkan dengan data index sebelumnya (3,4,7,8) dan susun menjadi [1, 3, 4, 7, 8]

Data Sumber : [1, 3, 4, 7, 8, 2, 6, 5]
Index : 1 2 3 4 5 6 7 8

- Ambil data index 6 (2), bandingkan dengan data index sebelumnya (1,3,4,7,8) dan susun menjadi [1, 2, 3, 4, 7, 8]

Data Sumber : [1, 2, 3, 4, 7, 8, 6, 5]

Index : 1 2 3 4 5 6 7 8

4. Ambil data index 7 (6), bandingkan dengan data index sebelumnya (1, 2, 3, 4, 7, 8) dan susun menjadi [1, 2, 3, 4, 6, 7, 8]

Data Sumber : [1, 2, 3, 4, 6, 7, 8, 5]
Index : 1 2 3 4 5 6 7 8

- Ambil data index 8 (5), bandingkan dengan data index sebelumnya (1, 2, 3, 4, 7, 6, 8) dan susun menjadi [1, 2, 3, 4, 5, 6, 7, 8]
- Hasil Akhir **1, 2, 3, 4, 5, 6, 7, 8**

Bagaimana jika urutannya kita balik dari besar ke kecil ???
Apakah Order fungsinya tetap atau lain, jika lain masuk dalam Apa ?

Source Simulasi

Analisa Insertion Sort

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} t_j - 1 + c_7 \sum_{j=2}^{n} t_j - 1 + c_8(n-1)$$

Jika $$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n} j - 1 = \frac{n(n+1)}{2}$$

Maka $$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n+1)}{2}\right) + c_7\left(\frac{n(n+1)}{2}\right) + c_8(n-1)$$

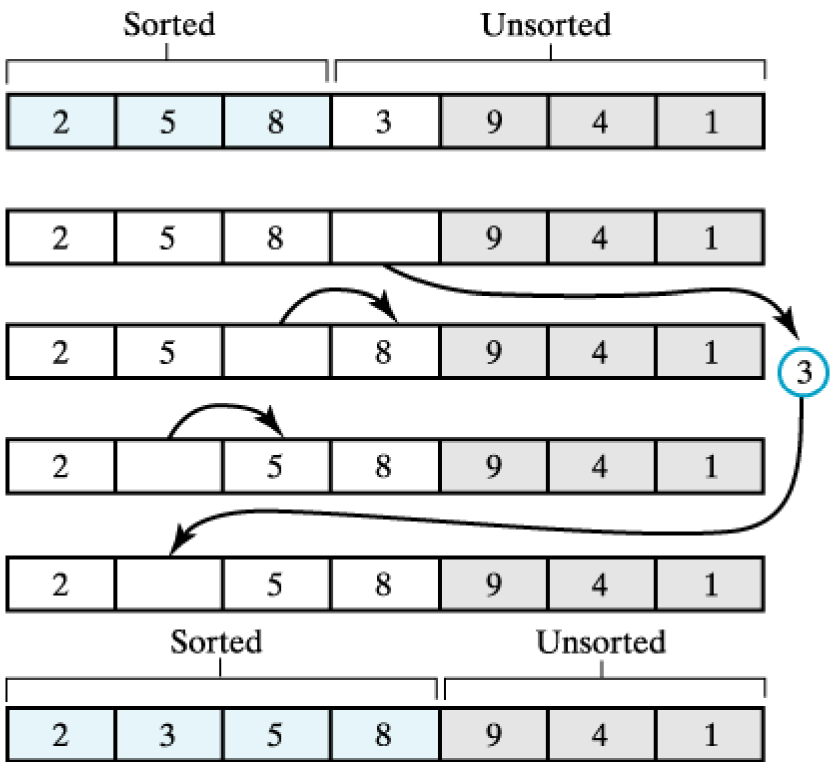
$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

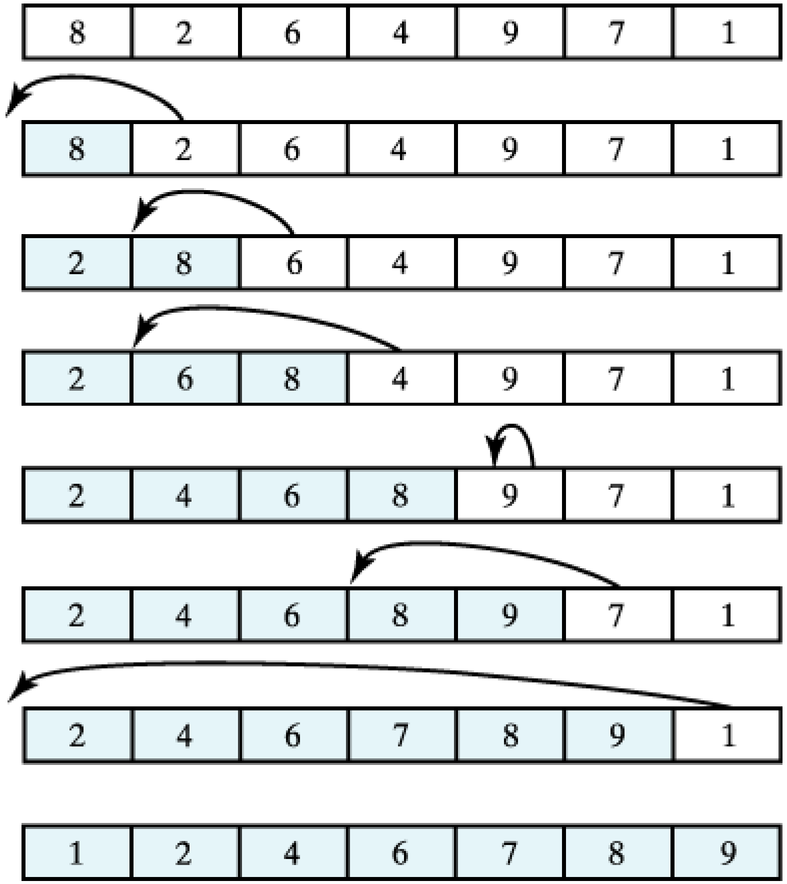
T(n) di atas Ini berbentuk quadratik an^2 + bn+ c, sehingga order fungsinya $O(n^2)$

Dekomposisi Iteratif

***Algorithm* insertionSort(a, first, last)**
for (unsorted = first+1 *hingga* last)
{ firstUnsorted = a[unsorted]
insertInOrder(firstUnsorted, a, first, unsorted-1)
}
***Algorithm* insertInOrder(element, a, begin, end)**
index = end
while ((index >= begin) *dan* (element < a[index]))
{ a[index+1] = a[index]
index - -
}
a[index+1] = element // *insert*

Contoh lain


Sorted
Unsorted
2 5 8 3 9 4 1
2 5 8 9 4 1
2 5 8 9 4 1
3
2 5 8 9 4 1
2 5 8 9 4 1
Sorted
Unsorted
2 3 5 8 9 4 1


8 2 6 4 9 7 1
8 2 6 4 9 7 1
2 8 6 4 9 7 1
2 6 8 4 9 7 1
2 4 6 8 9 7 1
2 4 6 8 9 7 1
2 4 6 7 8 9 1
1 2 4 6 7 8 9

Insertion Sort Recursive

Algorithm **insertionSort(a, first, last)**
if (jika isi a > 1)
{
    Urutkan elemen array a[first] hingga a[last-1]
    Insert last element a[last] ke posisi array terurut
}

Insertion Sort dalam java

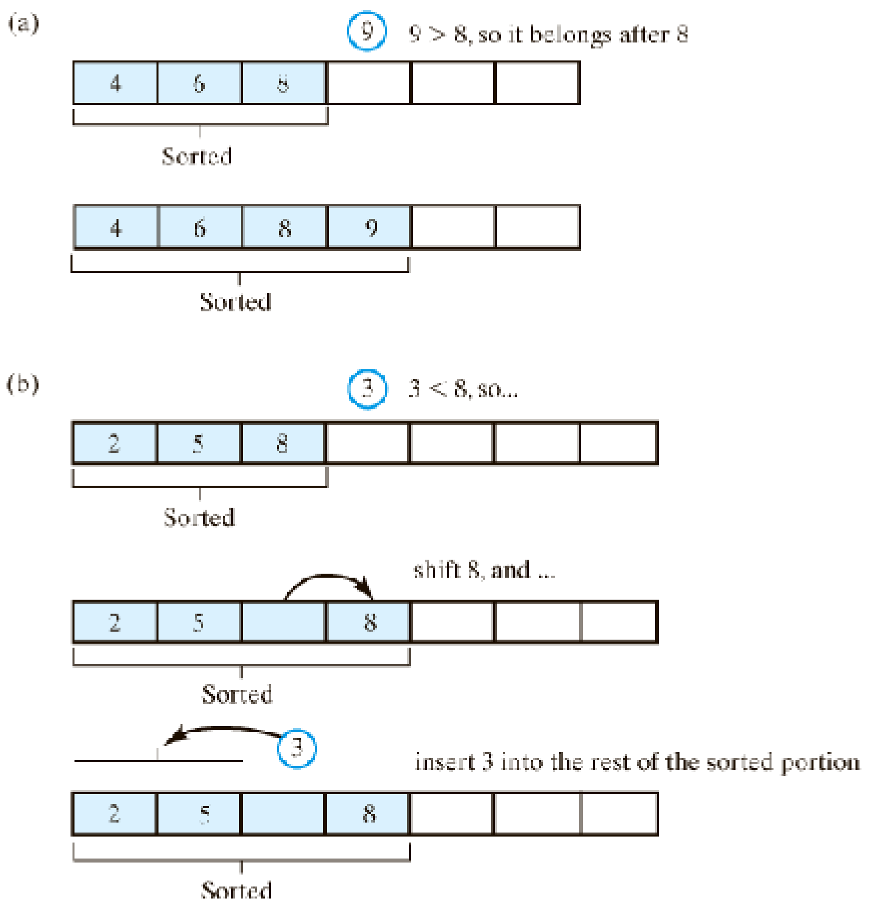
```
public static void insertionSort( Comparable[] a, int first, int
   last)
{
     If ( first < last)
     {
          //sort kecuali last element
          insertionSort( a, first, last -1 );
          //insert last element dlm sorted order
          //dari first hingga last
          insertInOrder(a[last], a, first, last-1);
     }
}

insertInorder( element, a, first, last)
If (element >= a[last])
   a[last+1] = element;
else if (first < last)
{
   a[last+1] = a[last];
   insertInOrder(element, a, first, last-1);
}
else // first == last and element < a[last]
{
   a[last+1] = a[last];
   a[last] = element
}
```

IS recursive

Insert first unsorted element into the sorted portion of the array.

(a) The element is ≥ last sorted element;

(b) The element is < than last sorted



Analisa IS

- Best case : O(n)
- Worst case : $O(n^2)$
- Jika array hampir terurut, maka
  - IS bekerja sedikit lebih efficient
- Insertion sort untuk array ukuran kecil

Selection Sort

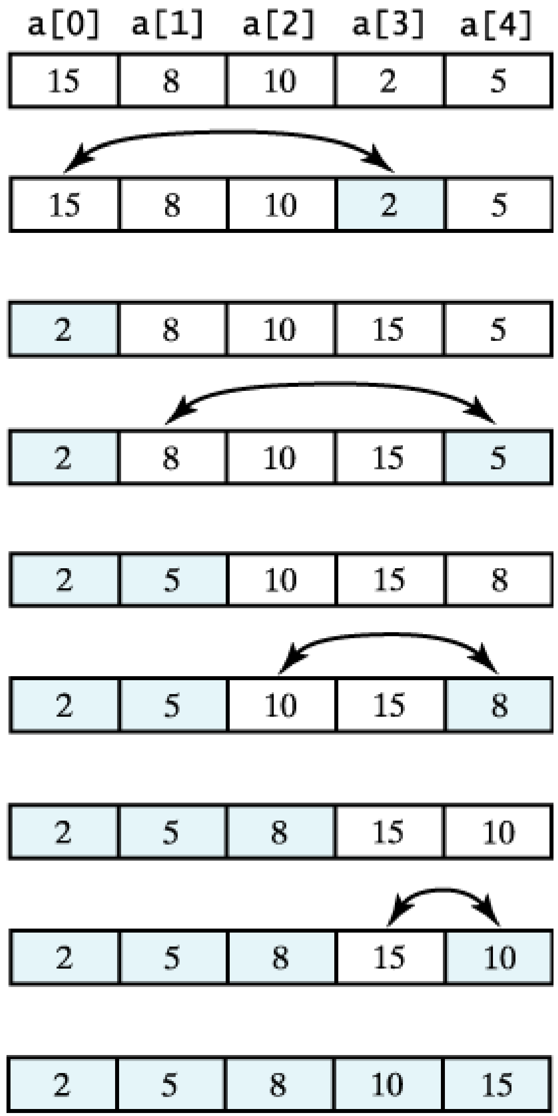
- Idenya adalah mencari membandingkan data dengan data lain yang terkecil kemudian menukar posisinya (index-nya), hal tersebut dilakukan urut mulai dari index terkecil hingga habis.

Algoritma versi 1

```
selectionsort(int A[], int besararray){
int i, j;
int min, temp;
for (i = 0; i < besararray -1; i++){
     min = i;
     for (j = i+1; j < besararray; j++){
       if (A[j] < A[min]) min = j;
     }
     temp = A[i]; /*pertukaran data*/
     A[i] = A[min];
     A[min] = temp;
   }
}
```

Algoritma versi 2

```
var lowkey: keytype;
{ kunci terkecil yg ada dari larik A[i], . . . , A[n] }
 lowindex : integer; { posisi lowkey  (kunci terkecil)}
 begin
1. for i := 1 to n-1 do begin
    { pilih data terendah dr A[i], . . . , A[n] dan pertukarkan dg A[i] }
2.    lowindex := i;
3.    lowkey := A[i].key;
4.    for j := i + 1 to n do { bandingkan key dg lowkey  saat ini}
5.       if A[j].key < lowkey then begin
6.          lowkey := A[j].key;
7.          lowindex := j end;
8.          swap(A[i], A[lowindex])
9.       end
10. end;
```


a[0] a[1] a[2] a[3] a[4]
15 8 10 2 5
15 8 10 2 5
2 8 10 15 5
2 8 10 15 5
2 5 10 15 8
2 5 10 15 8
2 5 8 15 10
2 5 8 15 10
2 5 8 10 15

Contoh

Data : [8, 4, 7, 3, 1, 2, 6, 5] (Data Sumber)

index   1 2 3 4 5 6 7 8

Data : [8, 4, 7, 3, 1, 2, 6, 5]
index 1 2 3 4 5 6 7 8

- untuk i=1 (8), cari dan bandingkan dengan data lainnya yang terkecil di sebelah ***kanannya***, ditemukan pada i=5 (1), lalu tukar nilai datanya pada posisi index-nya data i[1] ditukar ke i[5], sehingga menjadi [**1**, 4, 7, 3, **8**, 2, 6, 5].

Data : [**1**, 4, 7, 3, **8**, 2, 6, 5]
index 1 2 3 4 5 6 7 8

- Untuk i=2 (4), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=6 (2), lalu tukar nilai datanya pada posisi index-nya data i[2] ditukar ke i[6], sehingga menjadi [1, **2**, 7, 3, 8, **4**, 6, 5].

Data : [1, **2**, 7, 3, 8, **4**, 6, 5]
index 1 2 3 4 5 6 7 8

- Untuk i=3 (7), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=4 (3), lalu tukar nilai datanya pada posisi index-nya data i[3] ditukar ke i[4], sehingga menjadi [1, 2, **3**, **7**, 8, 4, 6, 5].

Data : [1, 2, **3**, **7**, 8, 4, 6, 5]

index 1 2 3 4 5 6 7 8

- Untuk i=4 (7), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=6 (4), lalu tukar nilai datanya pada posisi index-nya data i[4] ditukar ke i[6], sehingga menjadi [1, 2, 3, **4**, 8, **7**, 6, 5].

Data : [1, 2, 3, **4**, 8, **7**, 6, 5]

index 1 2 3 4 5 6 7 8

- Untuk i=5 (8), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=8 (5), lalu tukar nilai datanya pada posisi index-nya data i[5] ditukar ke i[8], sehingga menjadi [1, 2, 3, 4, **5**, 7, 6, **8**].

Data : [1, 2, 3, 4, **5**, 7, 6, **8**]

index 1 2 3 4 5 6 7 8

- Untuk i=6 (7), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=7 (6), lalu tukar nilai datanya pada posisi index-nya data i[6] ditukar ke i[7], sehingga menjadi :

  [1, 2, 3, 4, 5, 6, 7, 8].

Data : [1, 2, 3, 4, **5**, 7, 6, **8**]

index 1 2 3 4 5 6 7 8

- Untuk i=6 (7), cari dan bandingkan dengan data lainnya yang terkecil disebelah kanannya, ditemukan pada i=7 (6), lalu tukar nilai datanya pada posisi index-nya data i[6] ditukar ke i[7], sehingga menjadi :

  [1, 2, 3, 4, 5, 6, 7, 8].

Analisa Selection Sort

```
for i := 1 to n-1 do begin        (akhir – awal + 2) + (akhir – awal + 1) .1 + ∑_{i=1}^{n} P(i)
 lowindex := i;
 lowkey := A[i].key;
 for j := i + 1 to n do           (akhir – awal + 2) + (akhir – awal + 1) (p + 1))
  if A[j].key < lowkey then          c
   begin
    lowkey := A[j].key;                    1
    lowindex := j                          1
   end;
 swap(A[i], A[lowindex])                   1
end;
```

Analisa Selection Sort

Inner Loop

$(akhir - awal + 2) + (akhir - awal + 1)\ (p + 1))$

$= ((n - (i+1)) + 2) + ((n - (i+1)) + 1)\ (2 + 1)$

$= ((n - (i+1)) + 2) + ((n - (i+1)) + 1)\ .\ 3$

$= ((n - (i+1)) + 2) + 3(n - (i+1)) + 3$

$= 4\ (n - (i+1)) + 5$

$= 4n - 4i + 4 + 5$

$= 4n - 4i + 9$

$(P(i))$ = Banyak Langkah dalam S = 1 + banyak langkah inner loop

$= 1 + 4n - 4i + 9$

$= 4n - 4i + 10$

Analisa Selection Sort

- Outer Loop
- Banyak langkah= (akhir – awal + 2) + (akhir – awal + 1) .1 + $\sum_{i=1}^{n} P(i)$

•= (((n – 1)-1) + 2) + (((n – 1)-1) + 1) .1 + $\sum_{i=1}^{n} (4n - 4i + 10)$

- 2n + 3 + $\sum_{i=1}^{n} 4n$ - $\sum_{i=1}^{n} 4i$ + $\sum_{i=1}^{n} 10$

•= 2n + 3 + 4n.n – 4. $\left(\frac{1}{2}n(n+1)\right)$ + 10.n

$$\sum_{i=1}^{n} i = \left(\frac{1}{2}n(n+1)\right)$$

•= 2n + 3 + 4n² - $\frac{4}{2}n^2 - \frac{4}{2}n$ +10n

•= 2n + 3 + 6n² – 2n² - 2n + 10n

•= 4n² + 10 n + 3

$4n^2 + 10n + 3 \in O(n^2)$

Source Simulasi

Iteratif VS DANDC

```
Algorithm selectionSort(a, n)
// Sort n elm pertama dr array a.
for (index = 0; index < n - 1; index++){
   imin=min(a[index],a[index+1],…, a[n-1])
   Tukar(a[index],a[imin]);
  }
```

```
Algorithm selectionSort(a, first, last)
if (first < last)
{  imin = min(a[first], a[first+1], . . . , a[last])
   Tukar (a[first],a[imin])
   selectionSort(a, first+1, last)
}
```

Perbandingan Waktu tempuh

- Iterative : loop dieksekusi sebanyak n–1 kali
  – Tiap pemangilan n–1, indexOfSmallest dipanggil, *last* =n-1, dan *first* berada dalam range 0 hingga n-2.
  – Tiap indexOfSmallest, dibandingkan sbyk l*ast – first* kali
  – Total Operas: $(n – 1) + (n – 2) + \ldots + 1 = n(n – 1)/2 = O(n^2)$
- Recursive :
  – Juga $O(n^2)$

Merge sort

- Divide
  - Jika S memiliki setidaknya 2 elemen , pindahkan seluruh elemen S dan letakan dalam 2 urutan S1 dan S2, tiap urutan mengandung setengah dari S
    - S1=$\lceil n/2 \rceil$ elemen dan s2= sisanya $\lfloor n/2 \rfloor$ elemen
- Conquer
  - SORT s1 dan s1 dengan merge sort
- Combine
  - Kembalikan elemen tadi ke S dengan menggabungkan urutan yang telah di sort S1 dan S2 menjadi 1 urutan sort

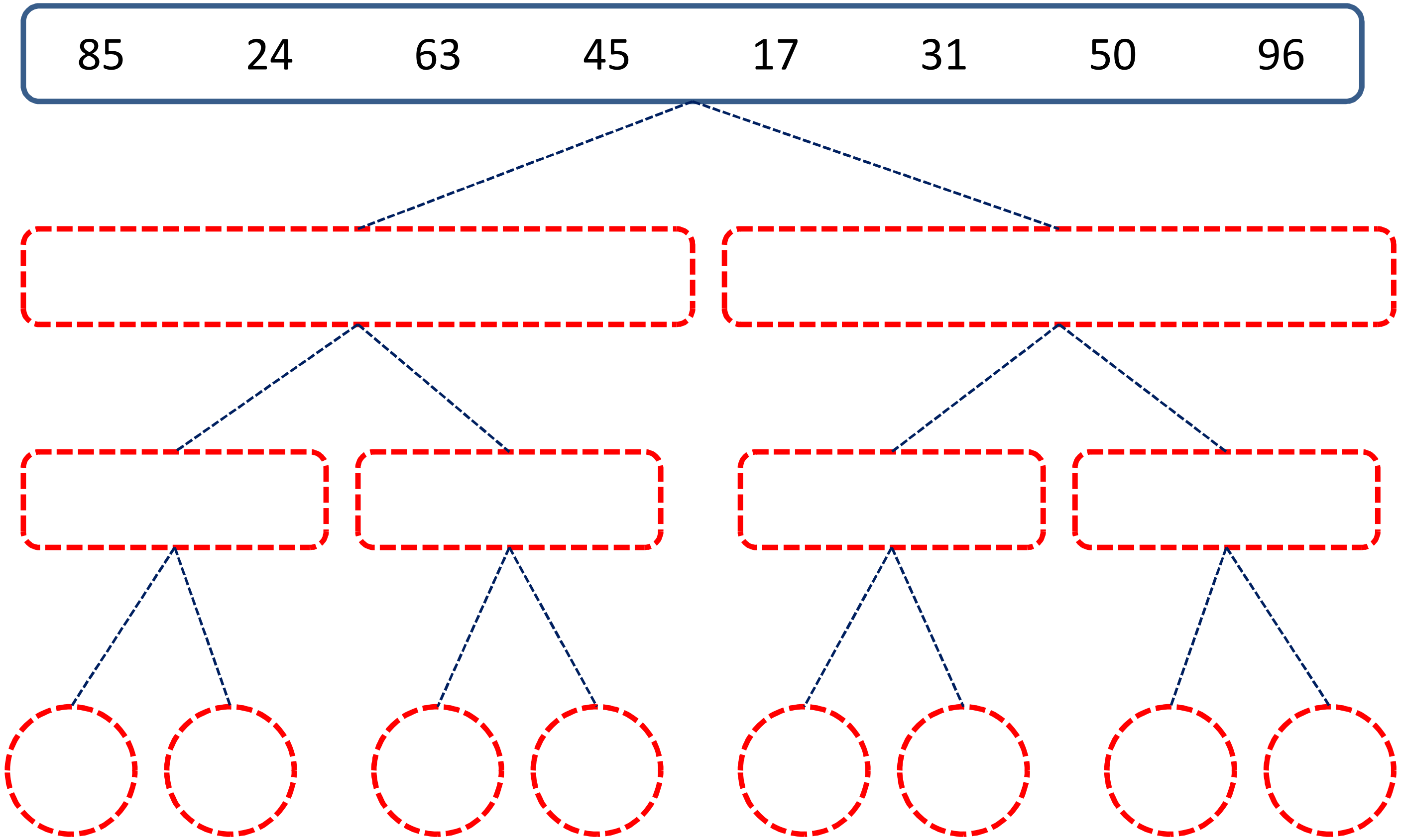
Algoritma Merge Sort

P dan r adalah
Bagian dari array A, yaitu
Lower bound dan
upper bound

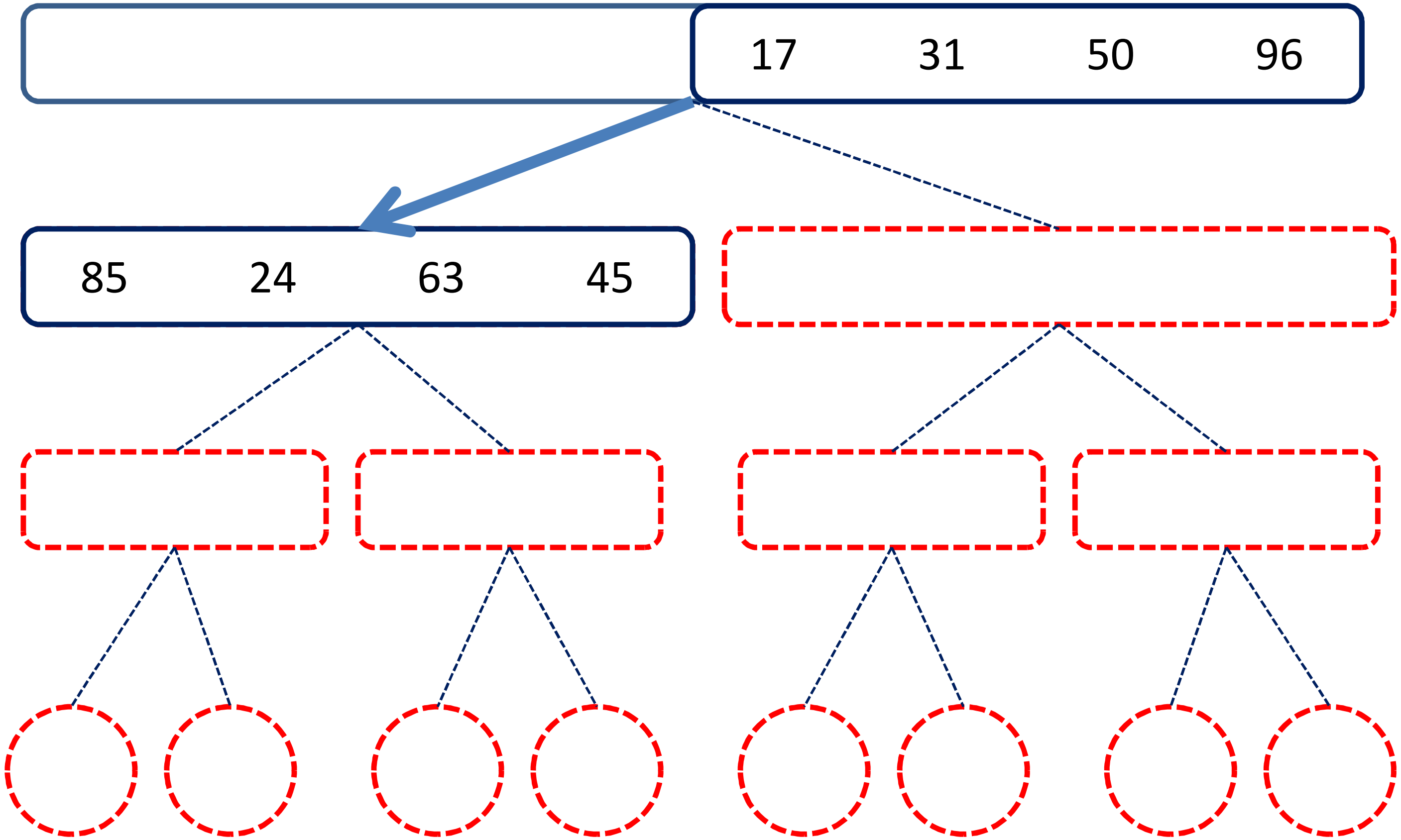
```
Merge-Sort(A,p,r)
If p<r  then
   q← (p+r)/2                divide
   Merge-Sort(A,p,q)         conquer
   Merge-Sort(A,q+1,r)
   Merge (A,p,q,r)           combine
```

```
Merge(A,p,q,r)
      Ambil 2 elemen teratas yang terkecil dari A[p..q]
Dan A[q+1..r], lalu letakan  ke hasil pengurutan.
Ulangi urutan (s1 dan s2) tersebut hingga kosong.
Kopi hasil pengurutan  ke dalam A[p..r]
```

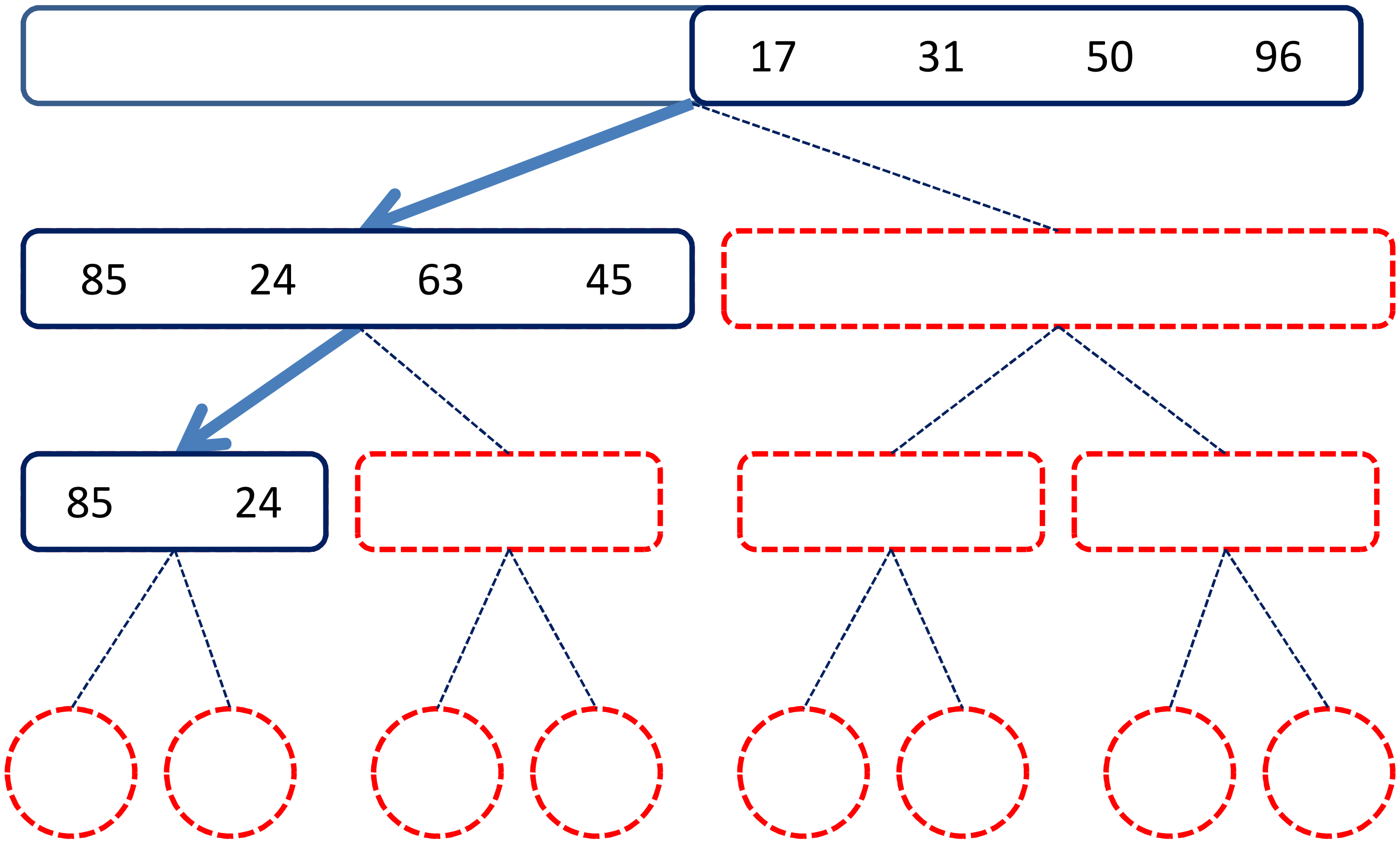
contoh



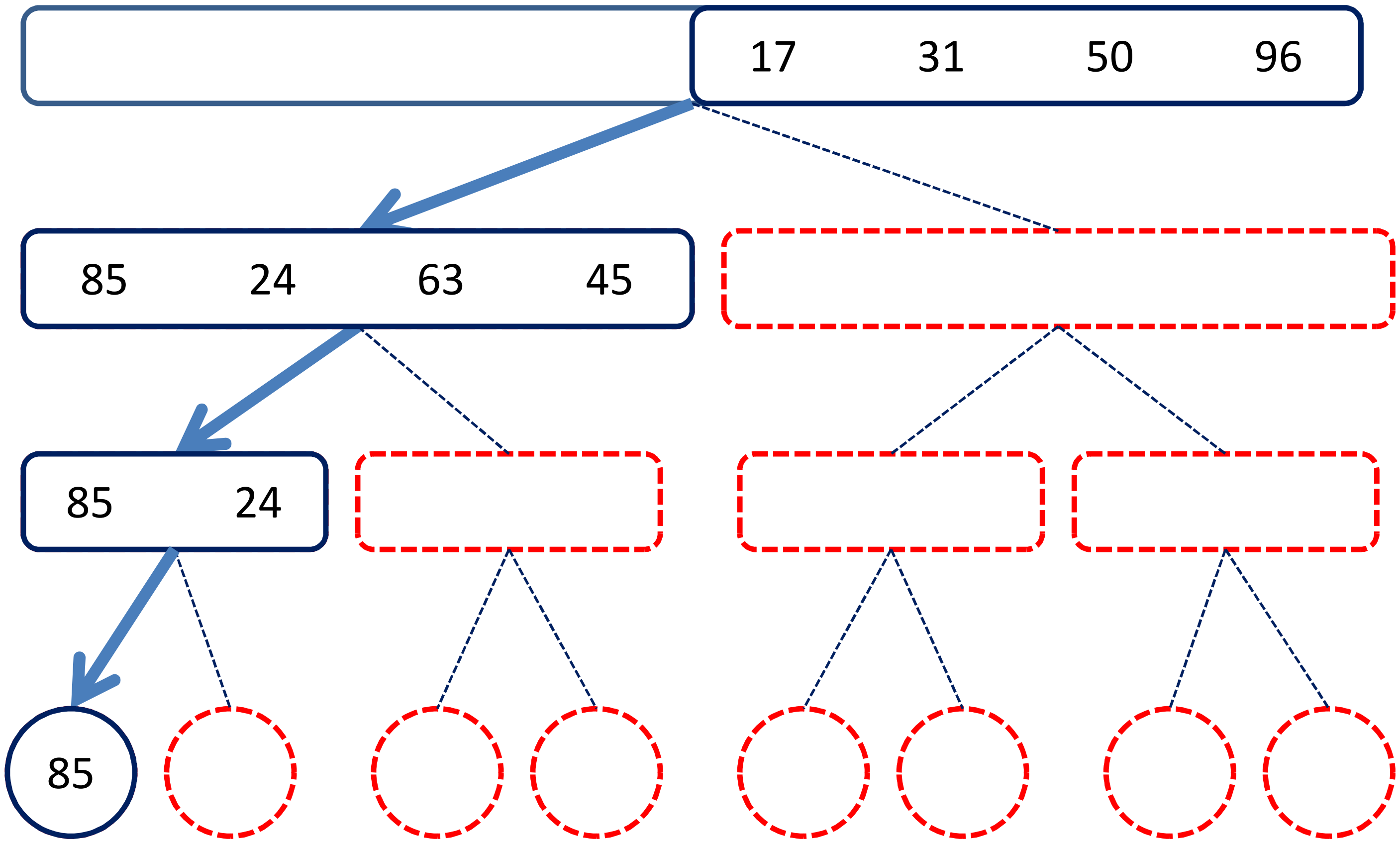
Running algoritma



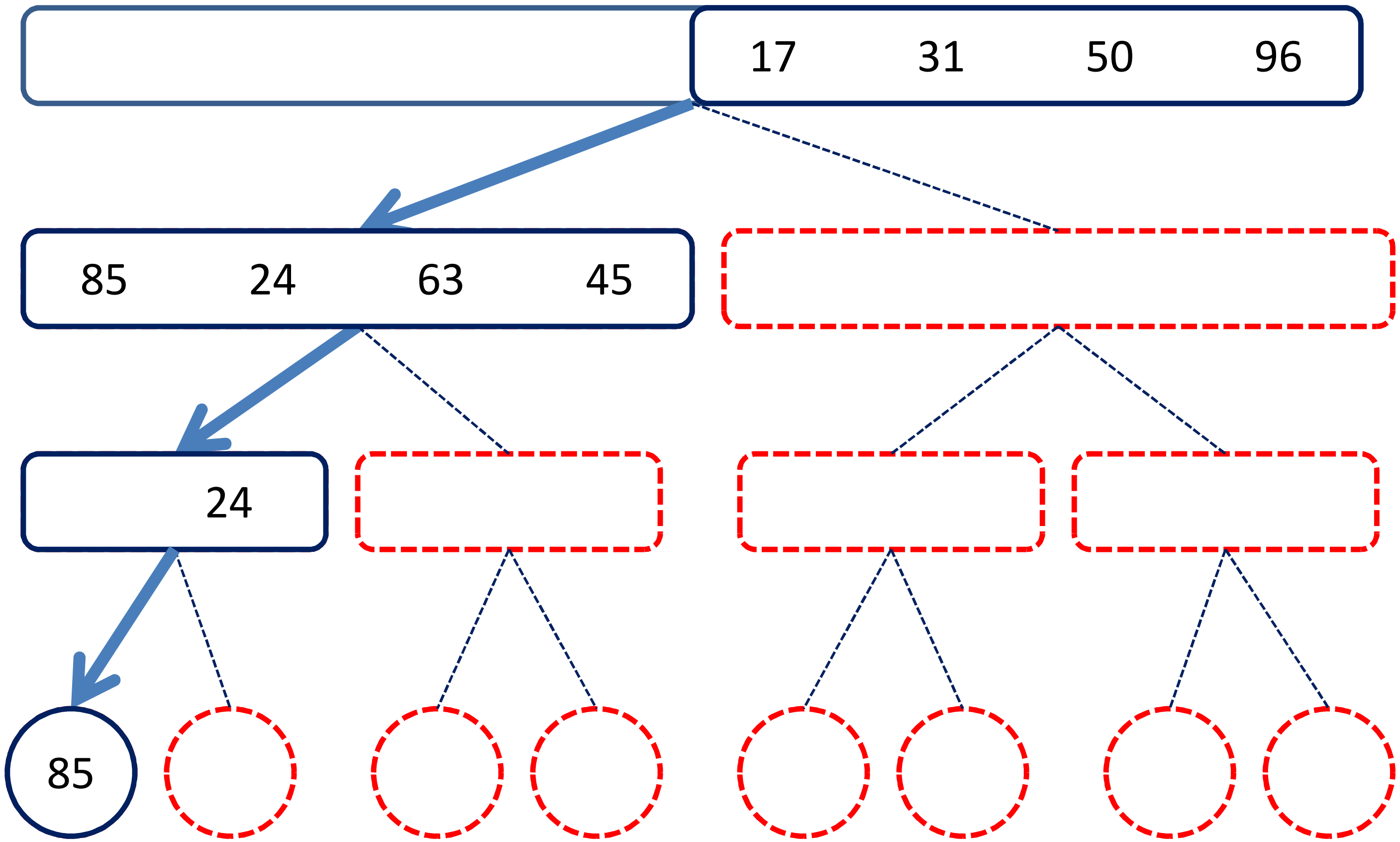
Running algoritma



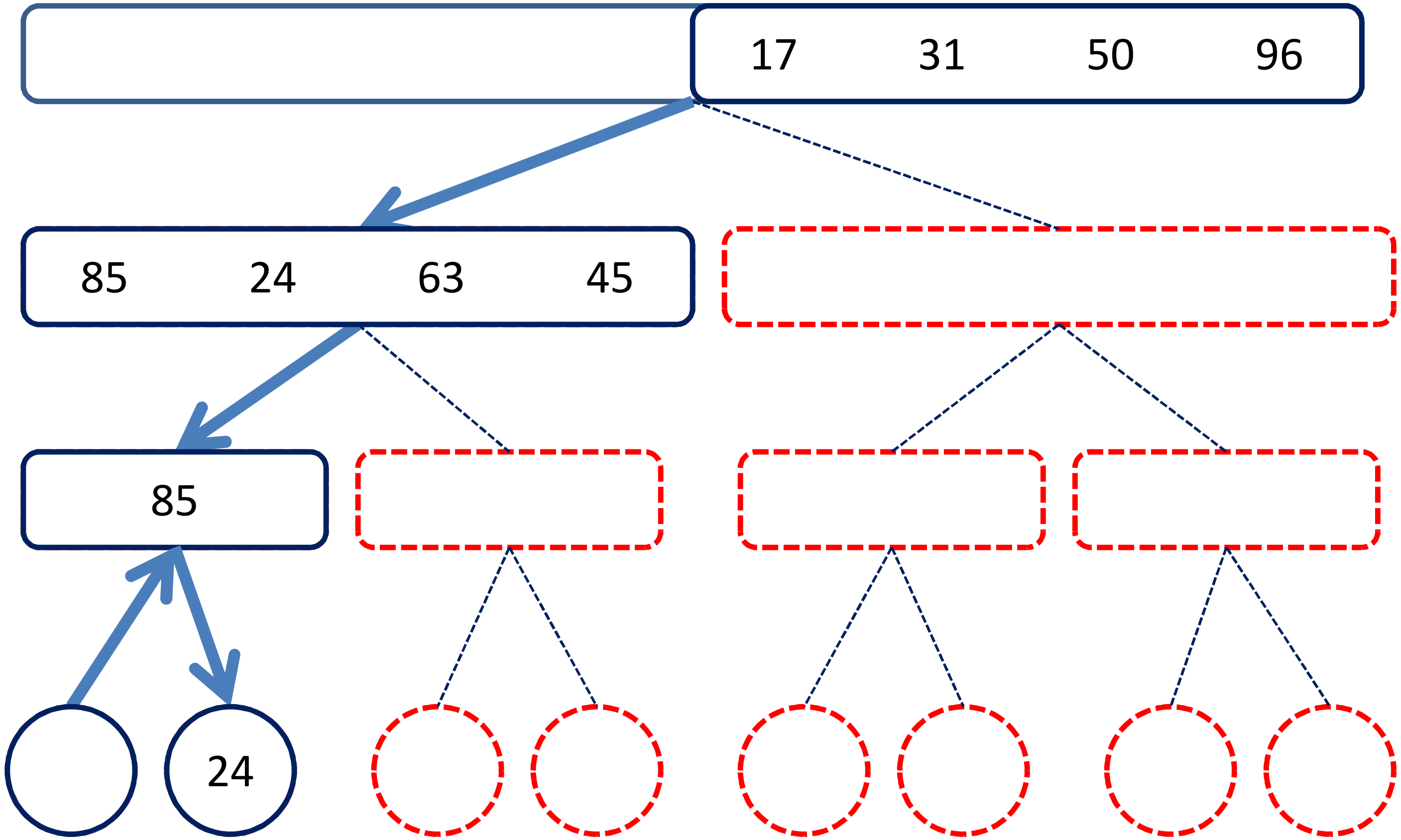
Running algoritma



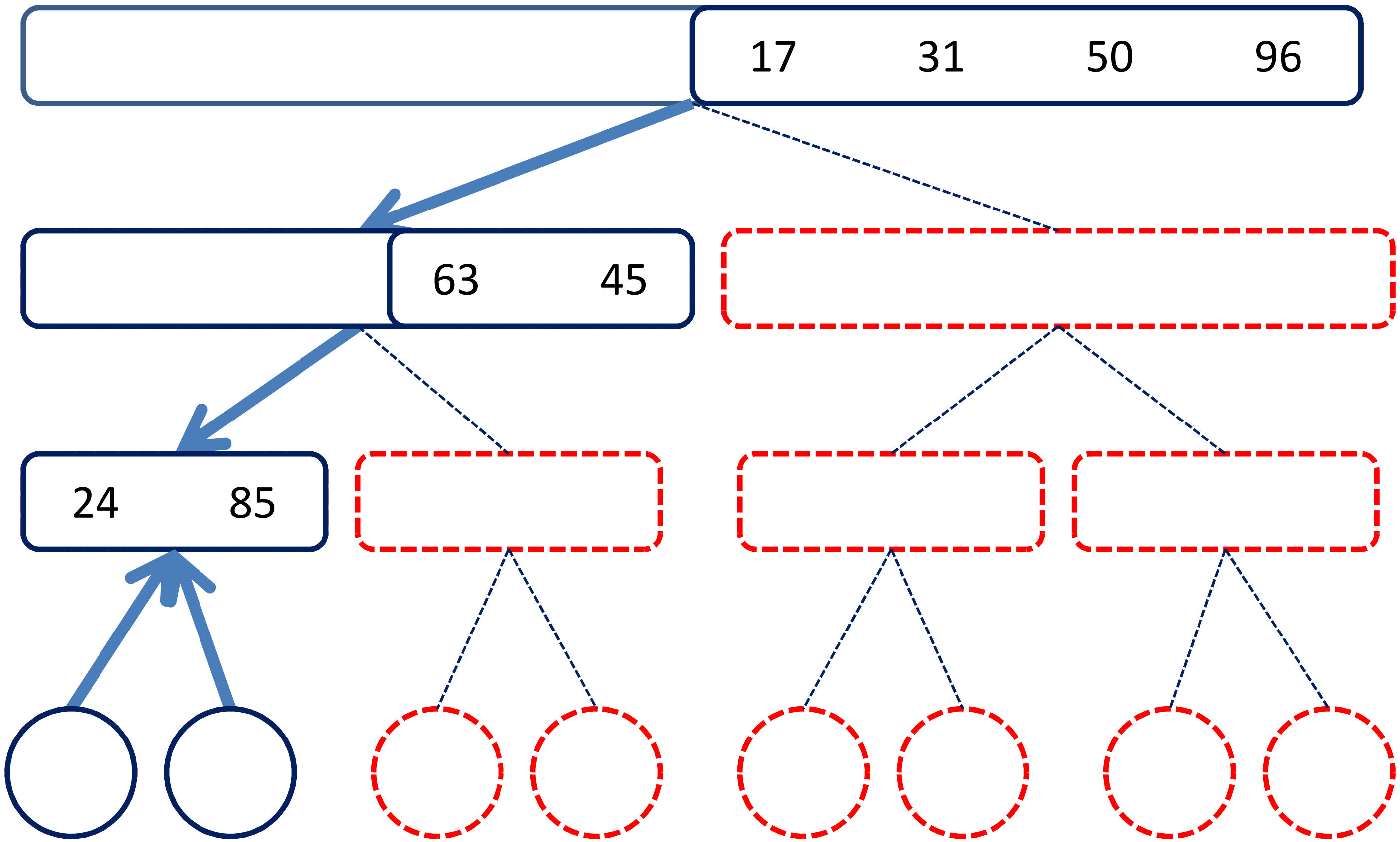
Running algoritma



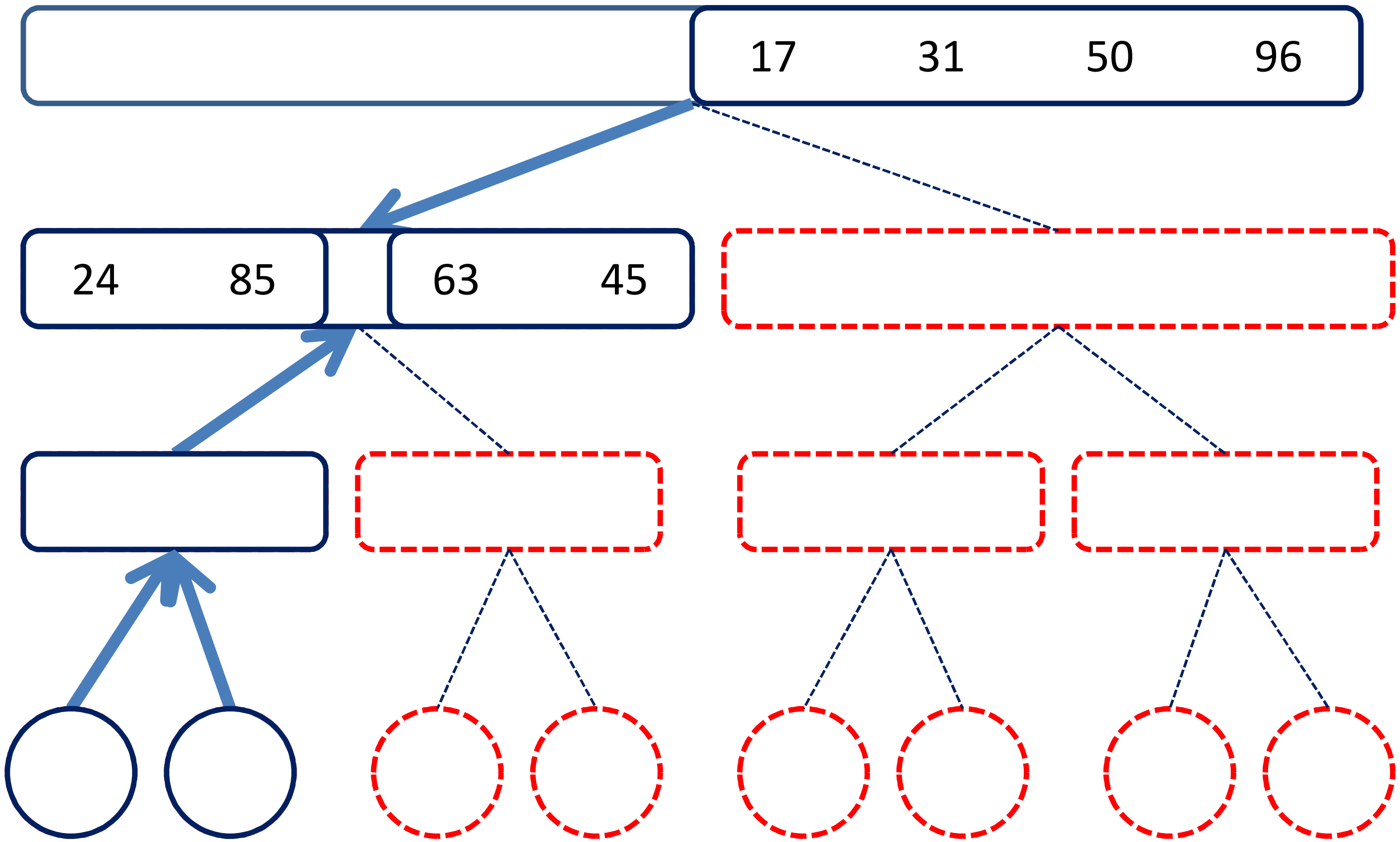
Running algoritma



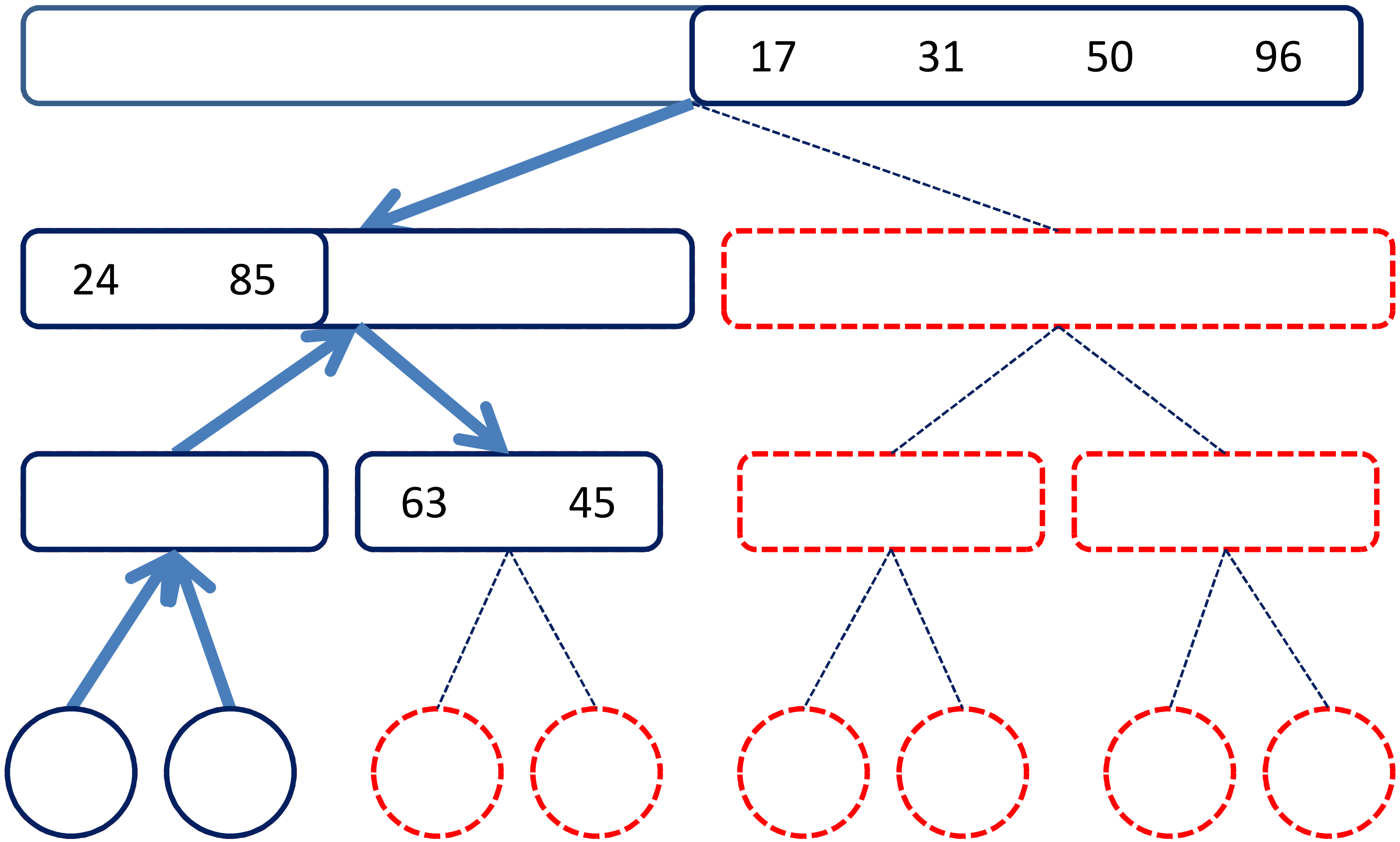
Running algoritma



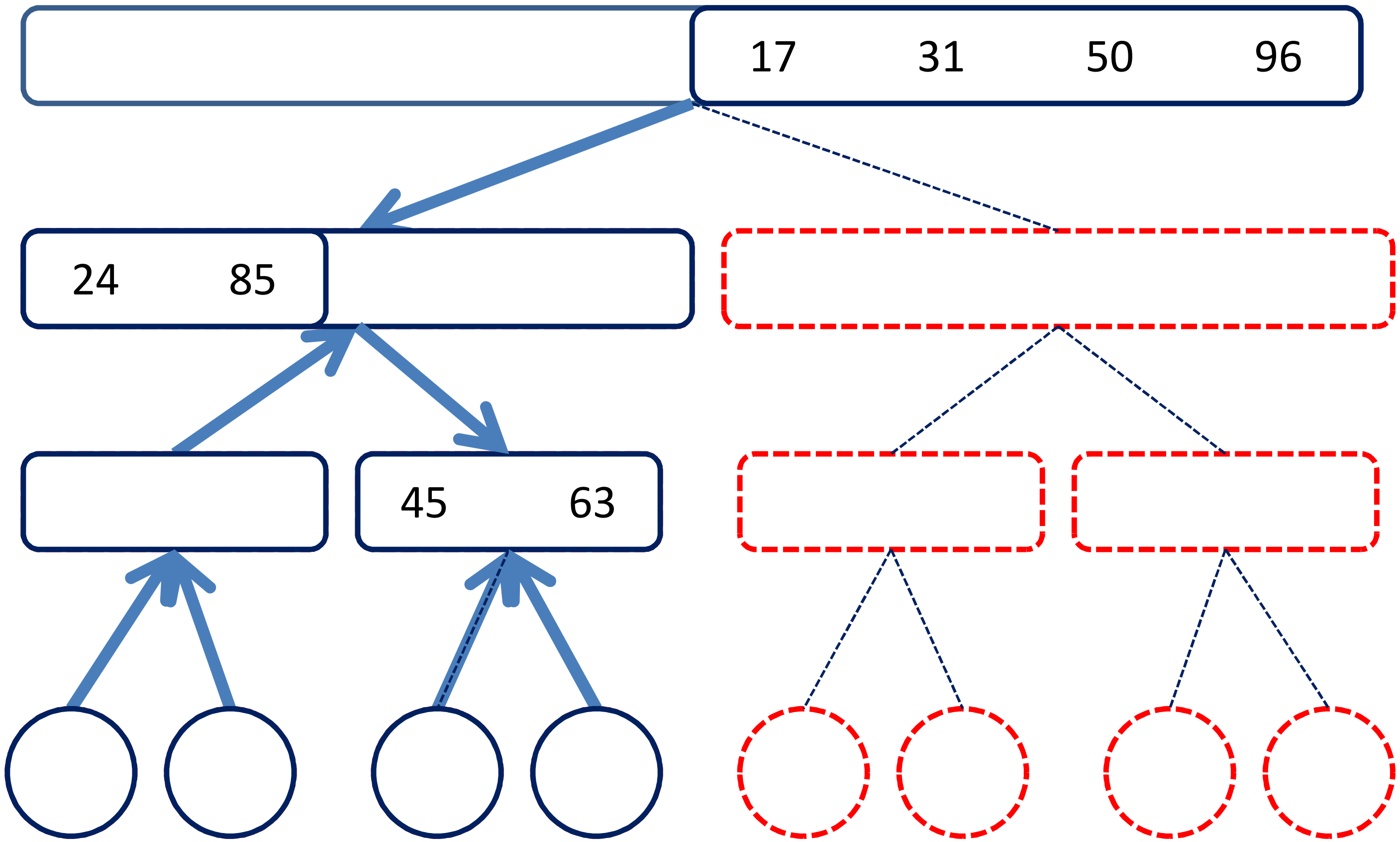
Running algoritma



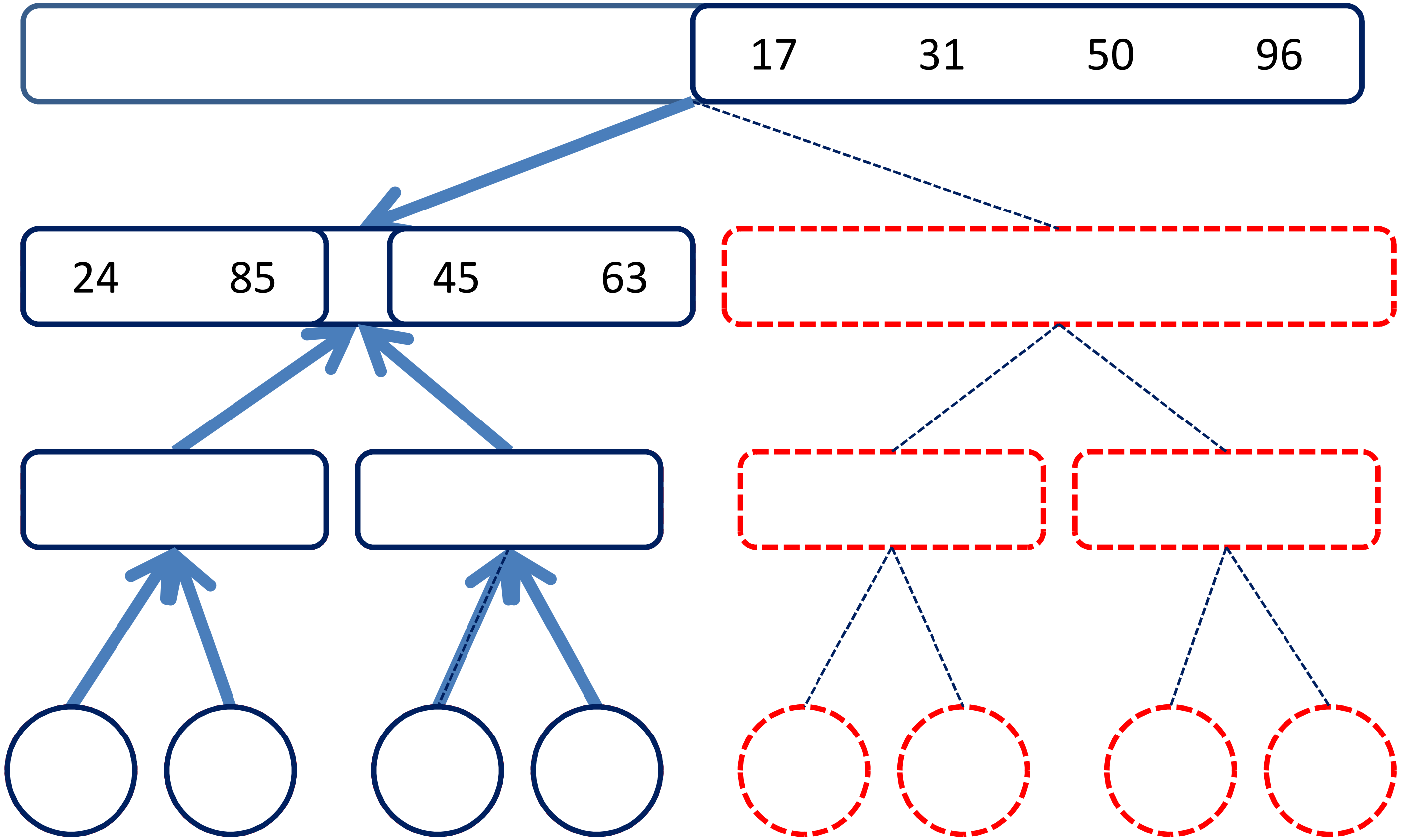
Running algoritma



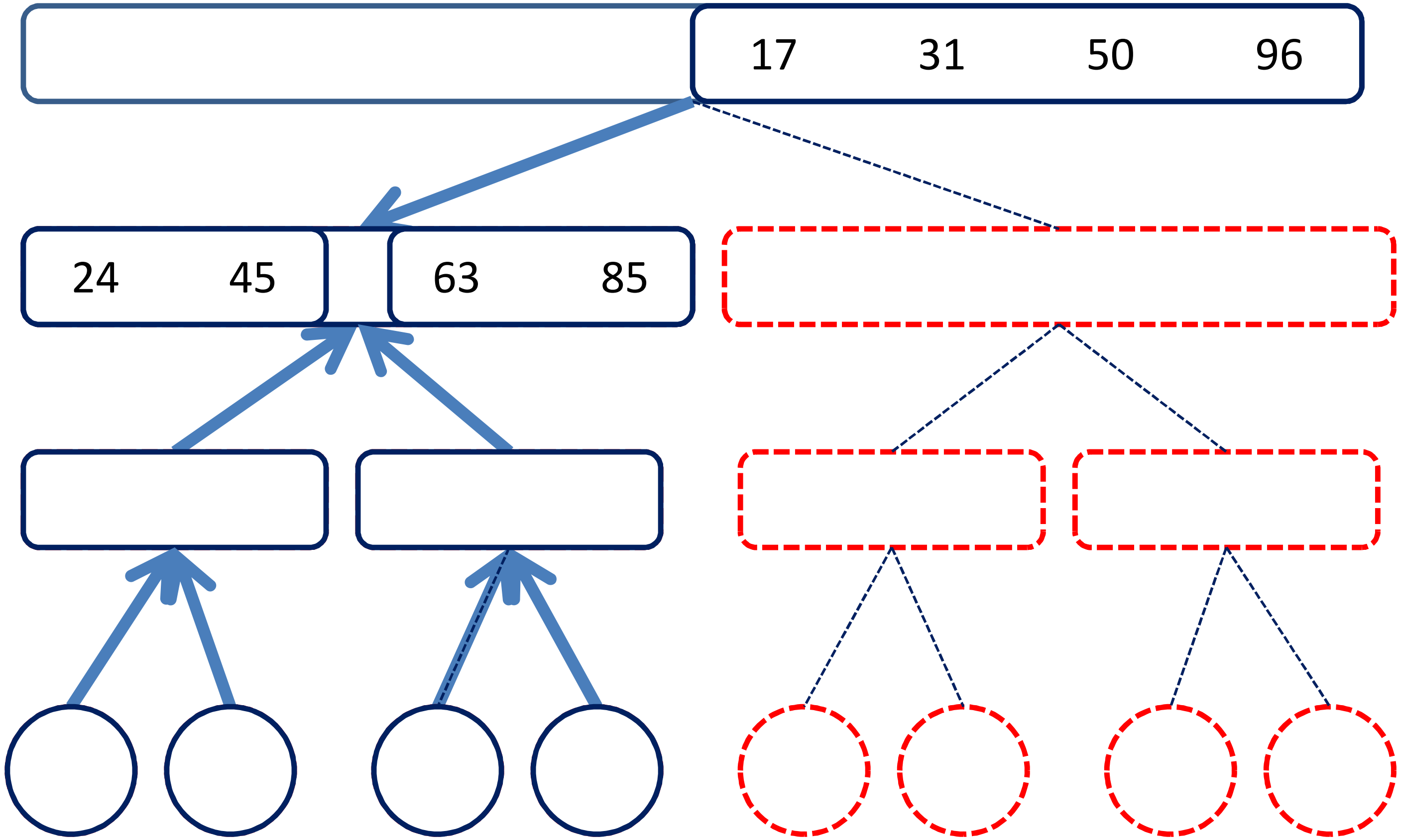
Running algoritma



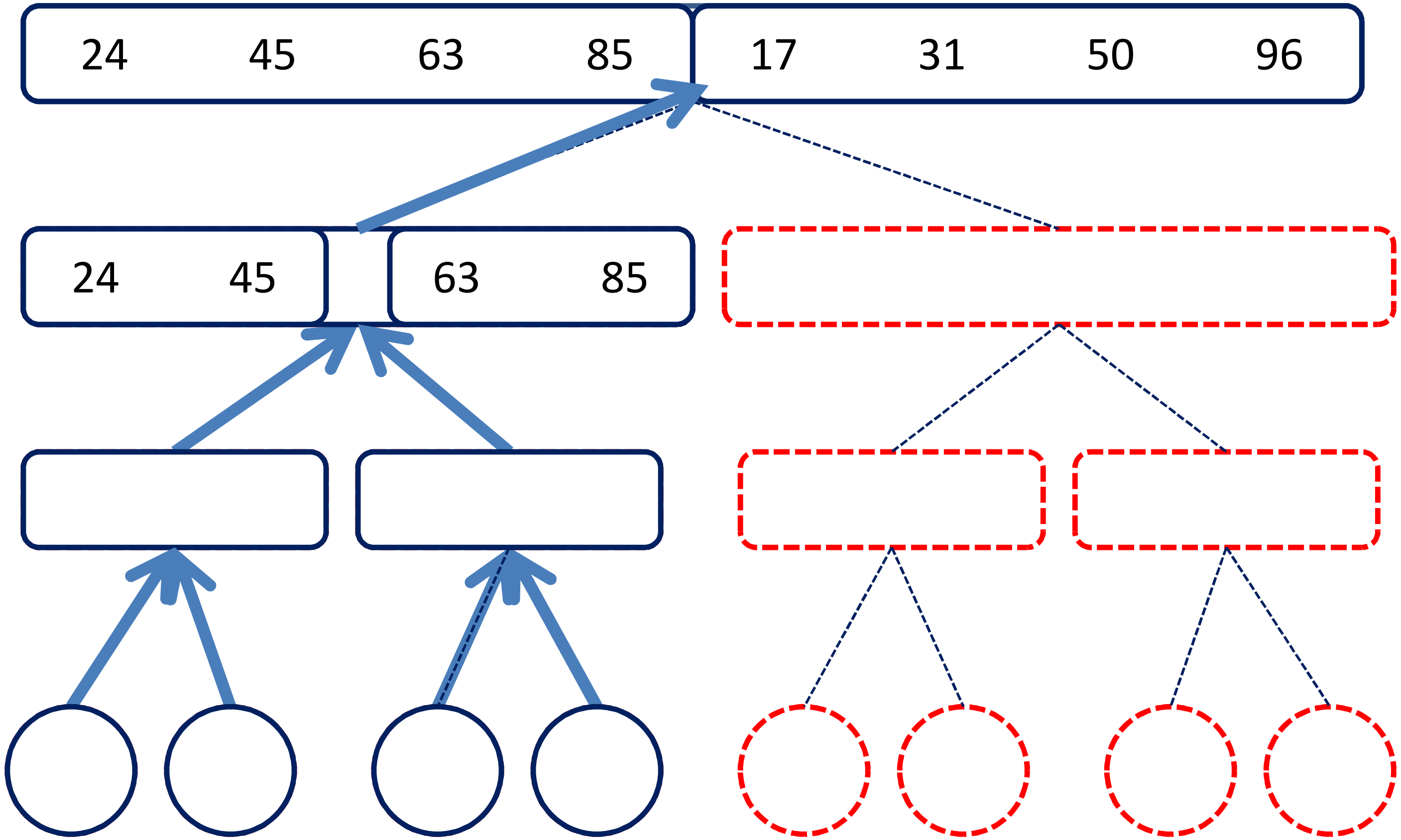
Running algoritma



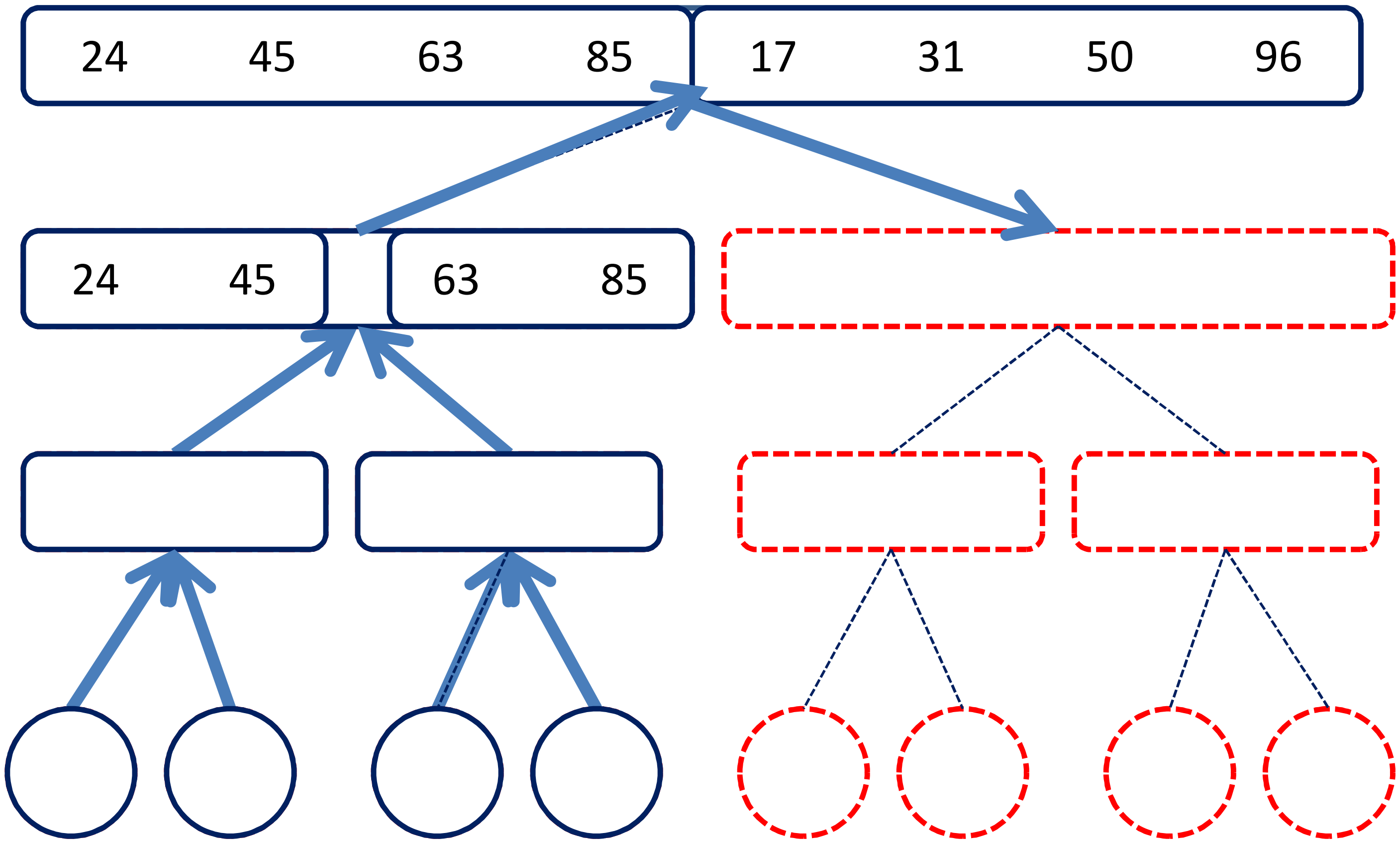
Running algoritma



Running algoritma

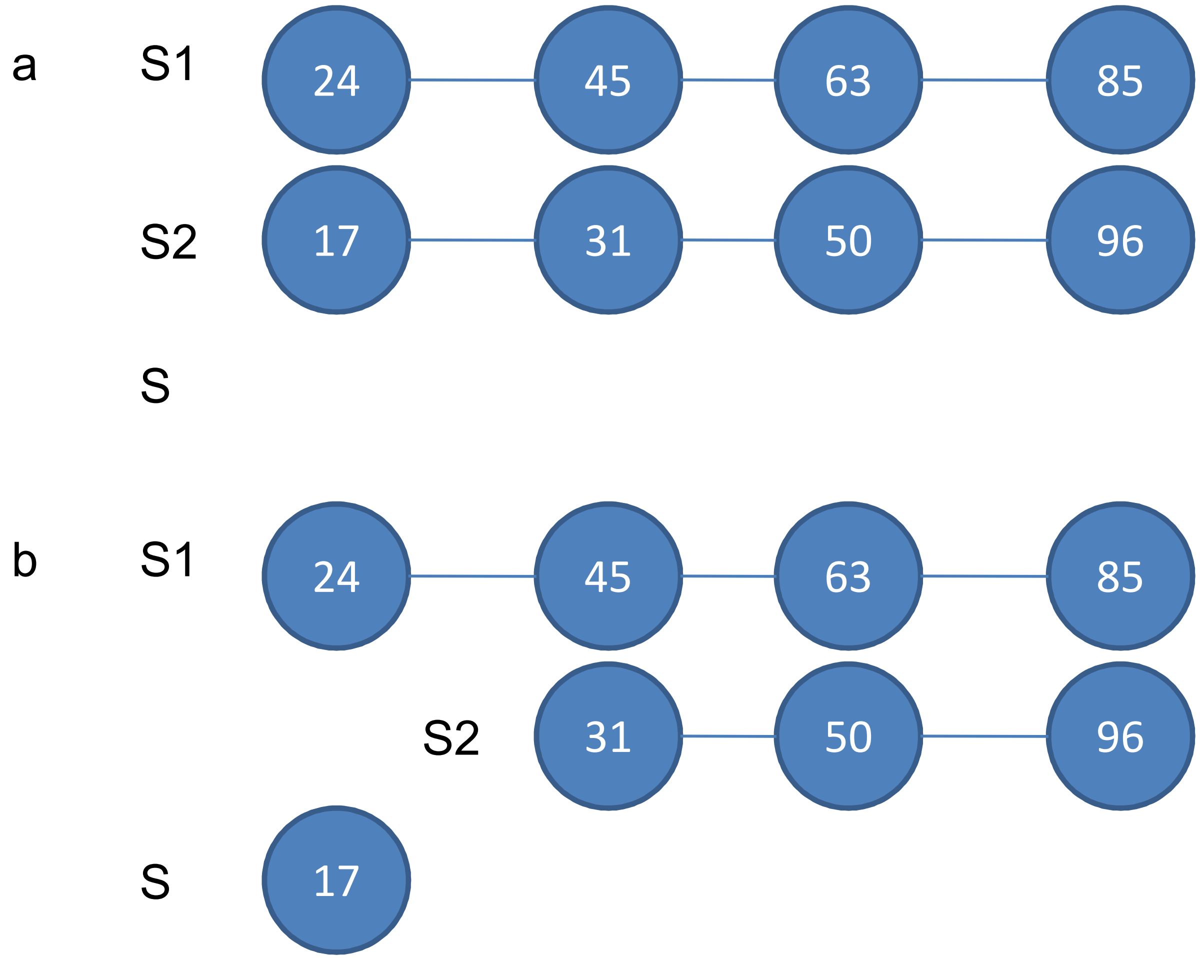


Running algoritma

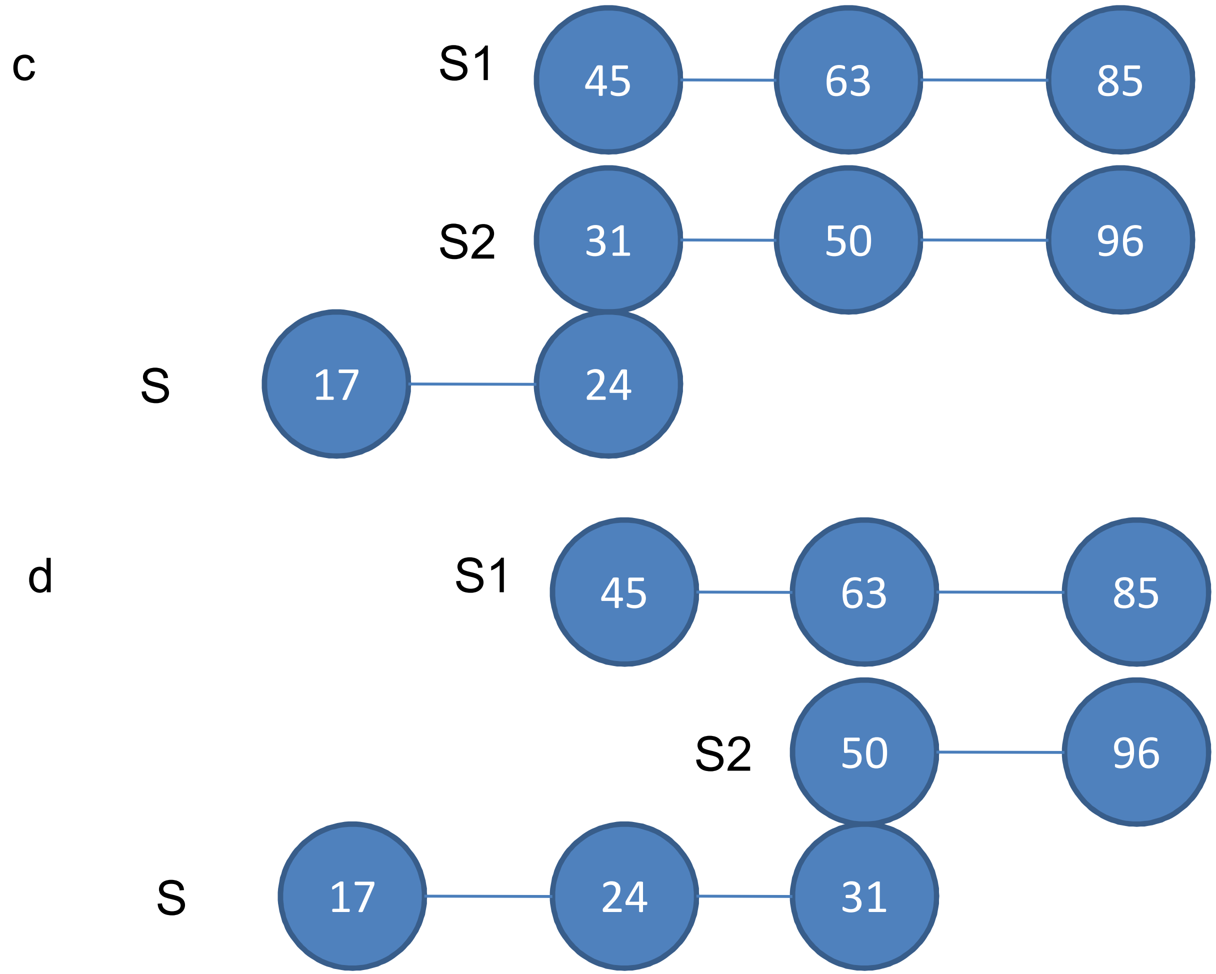


Langkah selanjutnya sama dengan bagian pertama

Merging 2 seq array

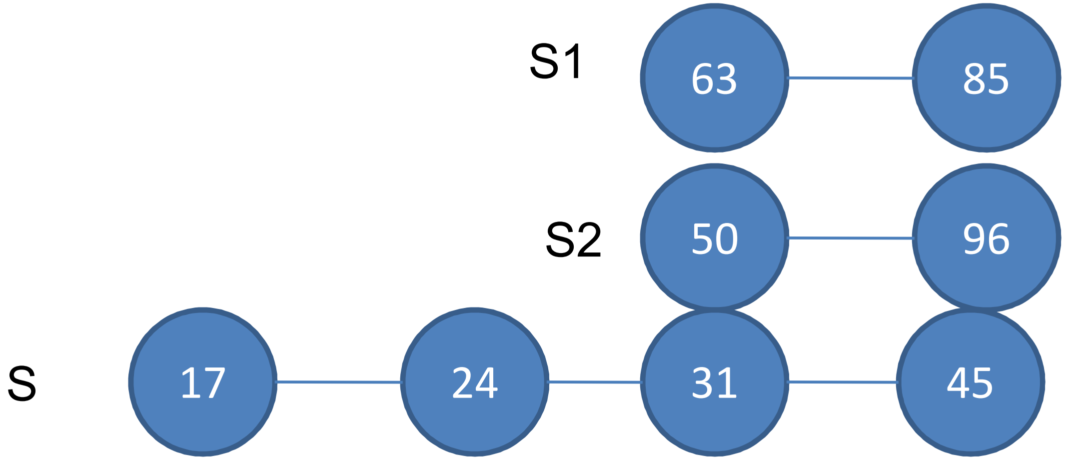


Merging 2 seq array

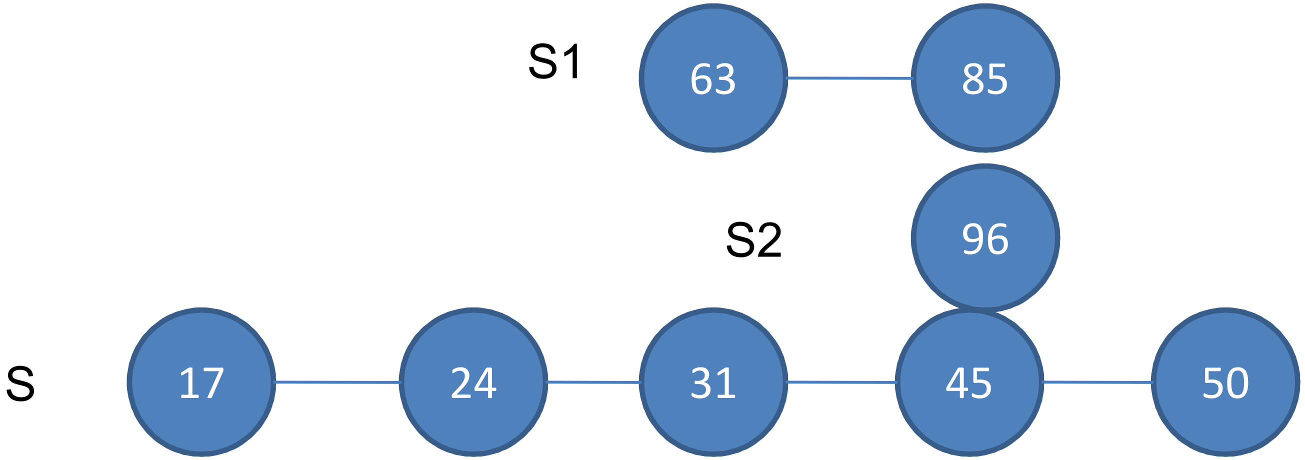


Merging 2 seq array

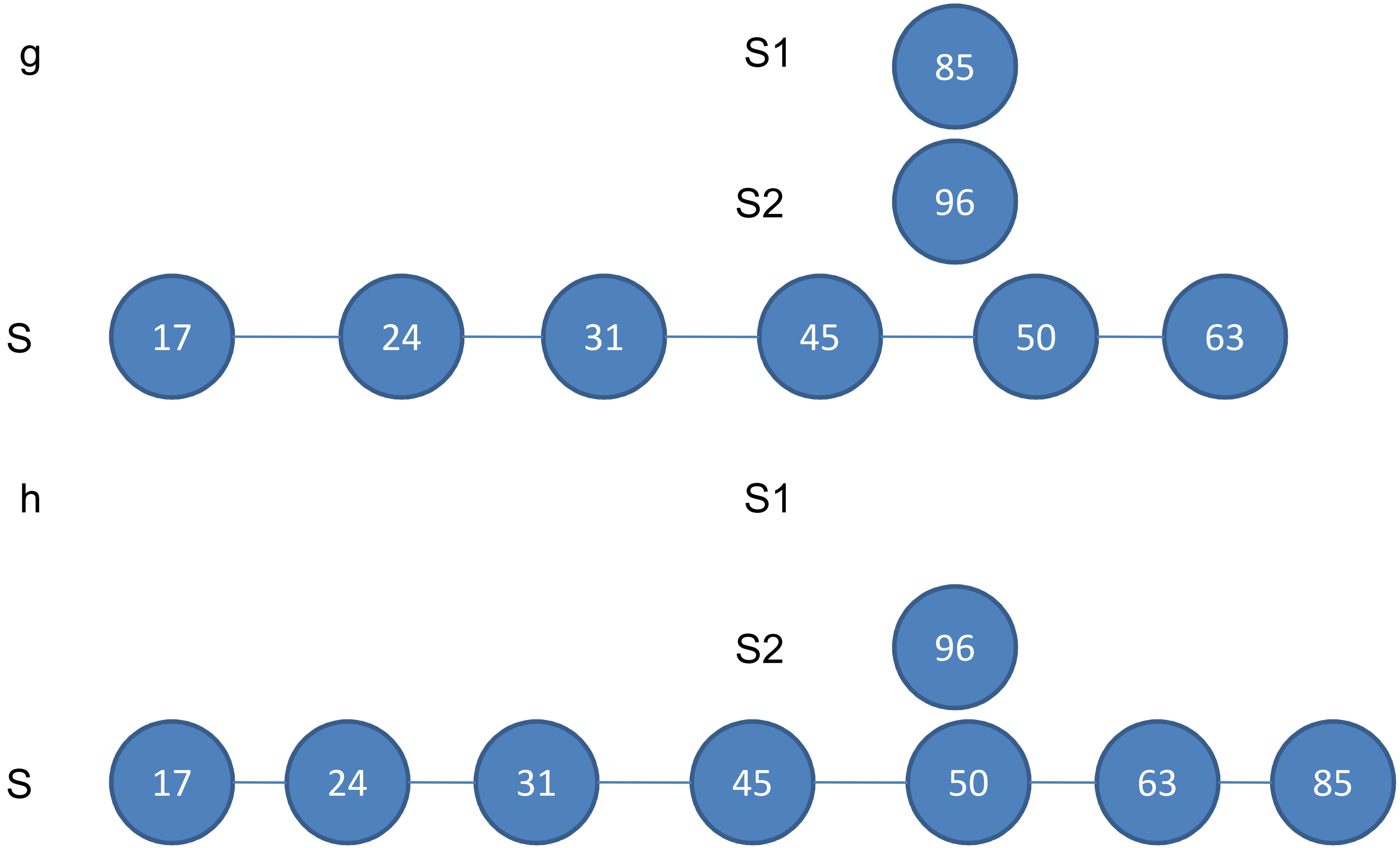
e



f



Merging 2 seq array



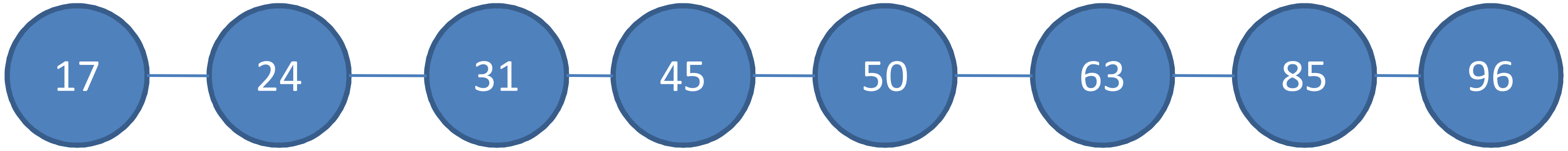
Merging 2 seq array

i

S1

S2

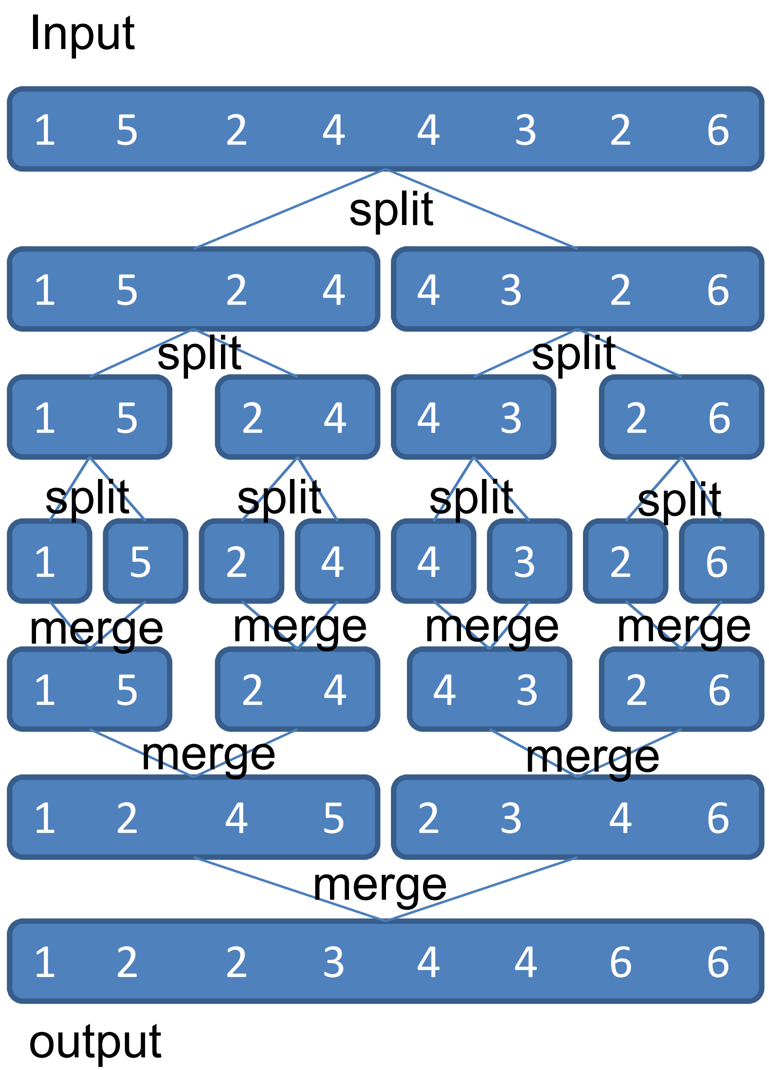
S



Merge revisi

- Untuk men-SORT sejumlah n
  - Jika n=1 selesai
  - Reccuren sort 2 list sejumlah $\lfloor n/2 \rfloor$ dan $\lceil n/2 \rceil$ elemen
  - Merge 2 list tsb dlm O(n)
- Strategi
  - Pecah masalah mjd subproblem yg mirip
  - Reccuren sub masalah
  - Combine solusi

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



C implementation

```
void mergesort(int numbers[], int temp[], int array_size){
     m_sort(numbers, temp, 0, array_size - 1);
}
 void m_sort(int numbers[], int temp[], int left, int right){
  int mid;
  if (right > left)
  {
    mid = (right + left) / 2;
    m_sort(numbers, temp, left, mid);
    m_sort(numbers, temp, mid+1, right);
    merge(numbers, temp, left, mid+1, right);
  }
}
```

C implementation

```
void merge(int numbers[], int temp[], int left, int mid, int right){
  int i, left_end, num_elements, tmp_pos;
  left_end = mid - 1;
  tmp_pos = left;
  num_elements = right - left + 1;
  while ((left <= left_end) && (mid <= right))  {
    if (numbers[left] <= numbers[mid])    {
      temp[tmp_pos] = numbers[left];
      tmp_pos = tmp_pos + 1;
      left = left +1;
    }
    else
```

C implementation

```
{temp[tmp_pos] = numbers[mid];
     tmp_pos = tmp_pos + 1;
     mid = mid + 1;    }
 }
 while (left <= left_end)  {
   temp[tmp_pos] = numbers[left];
   left = left + 1;
   tmp_pos = tmp_pos + 1;  }
 while (mid <= right)  {
   temp[tmp_pos] = numbers[mid];
   mid = mid + 1;
   tmp_pos = tmp_pos + 1;  }
 for (i=0; i <= num_elements; i++)  {
   numbers[right] = temp[right];
   right = right – 1;  }
}
```

Quick Sort

- Karakteristik
  - Sort dalam satu “tempat”, tidak perlu array tambahan
  - Sangat praktis, avarage case O(n log n), worst case O (n^2)

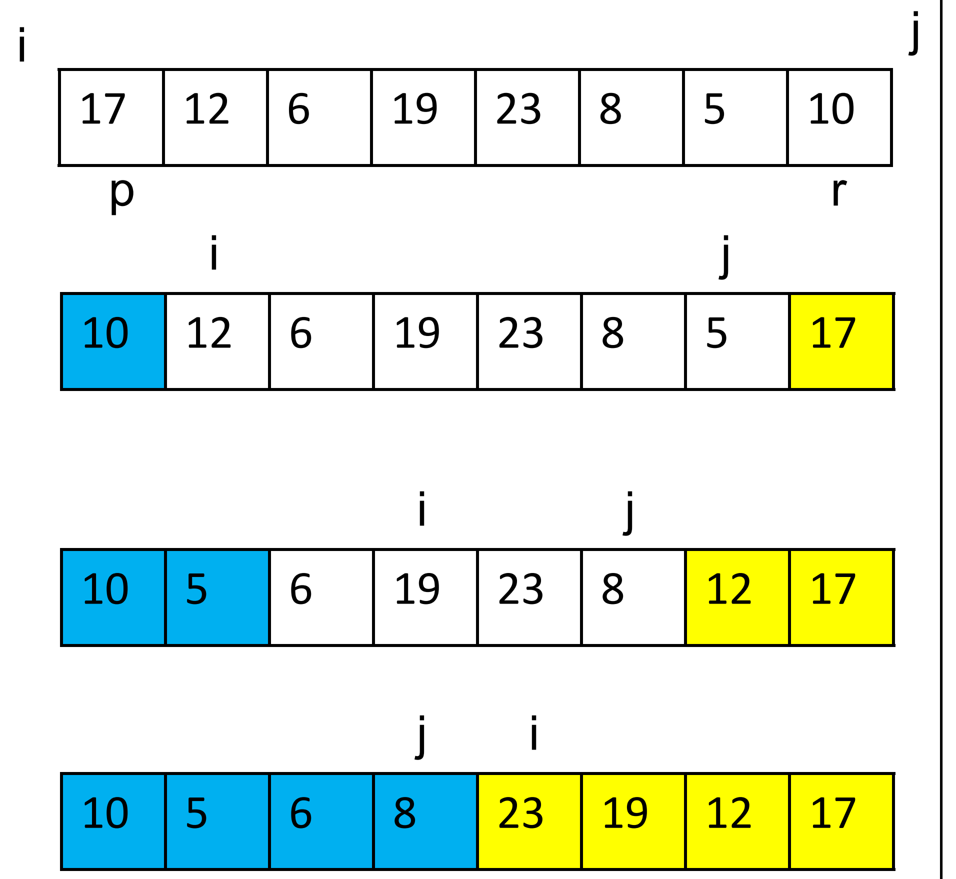
Prinsip Quick Sort

- Melihat deskripsi high level algorithm
- DANDC
  - Divide
    - Partisi array mjd 2 sub array s.r.s dalam bagian lower <= bagian higher
  - Conquer
    - Sort Recursive 2 sub array tadi
  - Combine
    - Sort selesai di "tempat" tadi

Partitioning (pembagian) wt=linear

```
Partition(A,p,r)
/*p & r batas array A*/
   x←A[p]
   i←p-1
   j←r+1
While TRUE
   REPEAT j←j-1
   UNTIL  A[j]>= x
   REPEAT i←i+1
   UNTIL A[i]<= x and i<=r
   if i>=j break;
   Tukar(A[i],A[j])

Tukar(A[p],A[j])
Return j
```

≤x=10≤



Mengembalikan posisi partisi dari array A

Algoritma Quick Sort

- Pemanggilan Quicksort(A,1,pjg[A])

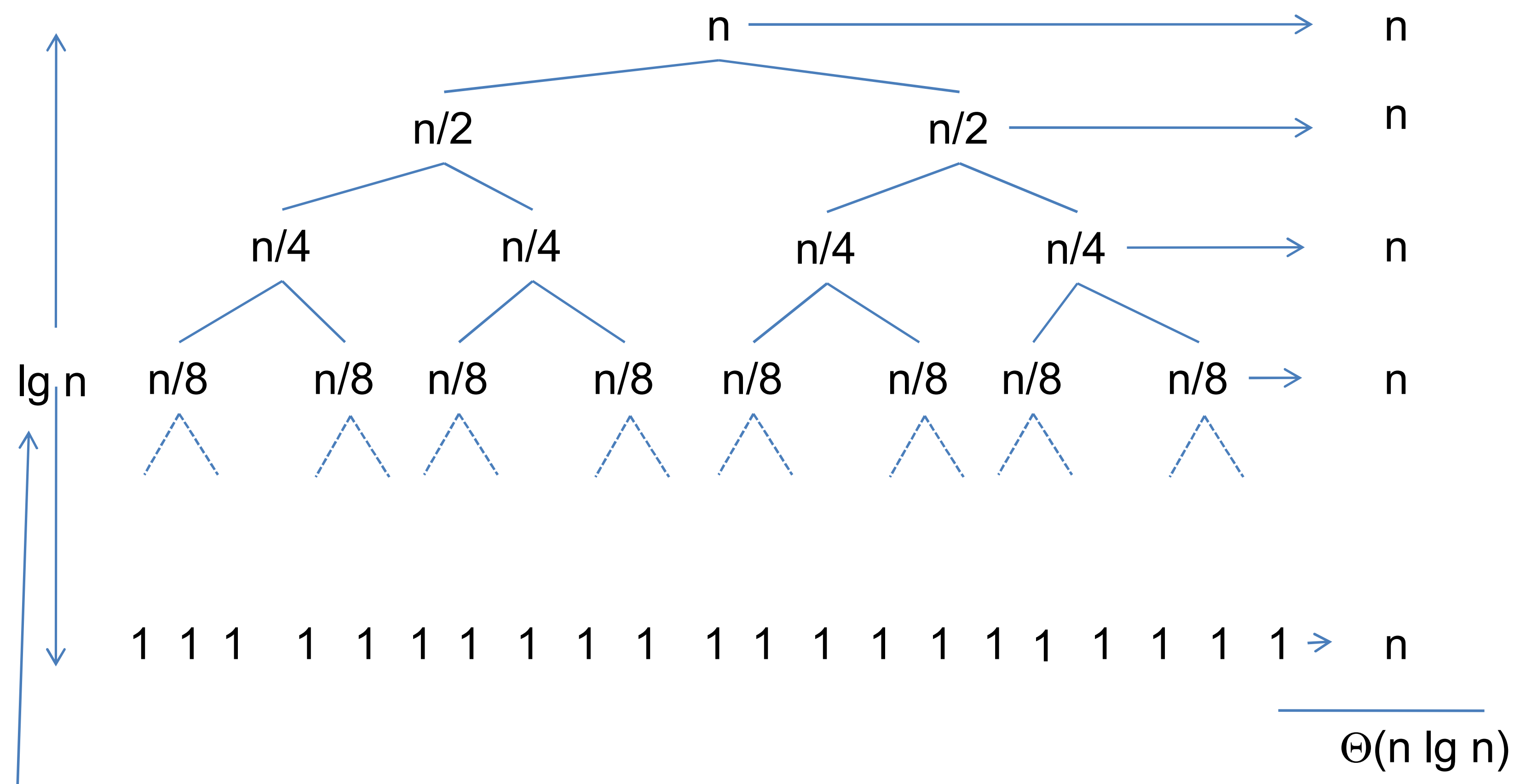
```
Quicksort(A,p,r)
   if p<r
        then q←Partition(A,p,r)
             Quicksort(A,p,q-1)
             Quicksort(A,q+1,r)
```

Partisi array A antara p dan r

Analisa Quicksort

- Asumsikan semua elemen berbeda
- Running time tergantung dari distribusi spliting array
- Spliting partisi perlu waktu $T(n) = 2T(n/2) + \Theta(n)$

Best case $T(n) = 2T(n/2) + \Theta(n)$



Running time tree adalah lg n, jadi total waktu tempuhnya adalah n lg n,
Karena setiap level pada tree adalah n,

Worst case = $\Theta(n^2)$

Randomize Quicksort

- Asumsi semua elemen berbeda
- Partisi di pilih secara random dalam elemen
- Konsekuensinya semua split (1:n-1,1:n-2,…, n-1:1) mirip dengan probabilitas 1/n
- Randomisasi merupakan alat generalisasi untuk meningkatkan algoritma dengan worst case yang jelek tetapi bagus untuk avarage case

Randomize Quicksort

```
RandomizePartition(A,p,r)
  i←Random(p,r)//generate angka
               //random antara p dan r
  Tukar(A[r],A[i])
  return Partition(A,p,r)
RandomizeQS(A,p,r)
  if p<r then
     q←RandomizePartition(A,p,r)
     RandomizeQS(A,p,r)
     RandomizeQS(A,q+1,r)
```

Analisis Randomize Quicksort

- Misal T(n), adalah waktu yang di butuhkan dalam perbandingan dalam quicksort sejumlah n
- Karena split butuh probabilitas 1/n, maka T(n) bernilai = T(i-1)+T(n-i)+n-1, jadi

$$T(n) = \frac{2}{n}\sum_{j=0}^{n-1} T(j) + n - 1$$

- Sehingga dg solving recc mjd O(n log n)

Analisis Randomize Quicksort

- Worstcase =$O(n^2)$
- Bestcase=$O(n \log n)$
- Expected running time quicksort $O(n \log n)$

Tugas

- Simulasikan Algoritma Radixsort dan Pigeon Hole
- Presentasikan