MODUL KULIAH KE → 12

ALGORITMA dan STRUKTUR DATA

MATERI KULIAH :

Comparison networks, prinsip Zero-one, *Bitonic Sorting Networks* yang terdiri dari *half cleaner* dan *bitonic sorter*, *Merging Networks*; bagian terakhir dijelaskan tentang bahasan utama yaitu : *Sorting Networks*.

POKOK BAHASAN :

Sorting Networks

Oleh : Endang Sri Rahayu

Sorting Network :

- ⇒ Memanfaatkan *comparison network* (*comparison* dilakukan paralel)
- ⇒ Kompleksitas waktu *sorting* dengan *comparison*, batas bawah n log n (n banyaknya unsur yg di*sort*)
- ⇒ *Sorting network*, kompleksitas waktu kurang dari n log n

Sorting network yang dapat mengurutkan sembarang *input sequence* akan dikonstruksi melalui :

1. *Half Cleaner* : *comparison network* dengan kedalaman 1
2. *Bitonic Sorter* : *network* yang dapat mengurutkan *Bitonic Sequence*
   - ⇒ *Bitonic Sequence* : suatu sequence yang monoton naik, kemudian monoton turun, atau sebaliknya.

     Contoh : < 1, 4, 6, 7, 3, 2 >

     < 9, 8, 3, 2, 4, 6 >

3. *Merging Network* : diperoleh dengan memodifikasi *bitonic sorter*. *Network* ini akan menggabungkan 2 *input sequence* yang masing-masing sudah terurut menjadi satu *output sequene* yang juga terurut.

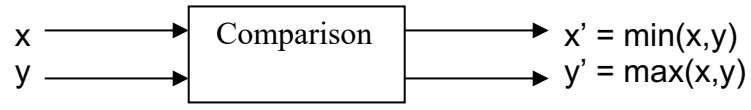
1. Comparison networks

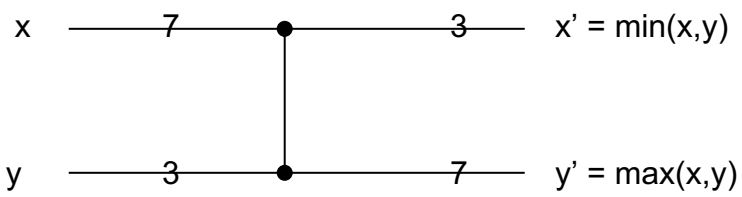
- Telah dibahas algoritma *sorting* (pengurutan) yang memanfaatkan *comparison* dengan batas bawah kompleksitas waktu pengurutan sebesar n log n, jika n= banyaknya unsur yang akan diurutkan.

- Pada *network* tersebut hanya *comparison* yang dapat dilakukan

- ***Sorting network*** **adalah** ***comparison network*** **yang selalu mengurutkan input yang diberikan.**

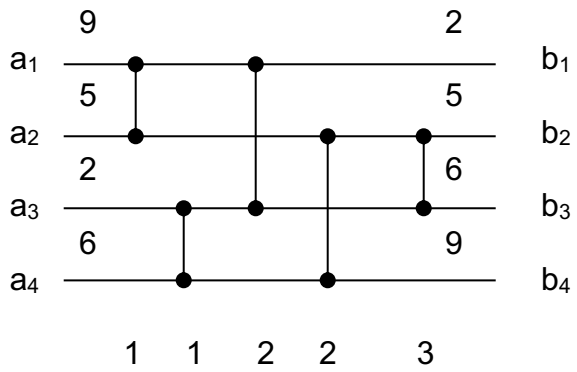
- *Comparison network* terdiri dari kabel penghubung dan *comparator*.





- *Comparator* dapat dihubungkan satu dengan yang lainnya asalkan dalam sirkuit yang terbentuk tidak ada *cycle*

Comparison network dengan 4 *input* dan 4 *output*,



- *Running time* dari *comparison network* = kedalaman maksimum dari *comparator* yang ada di *network* tersebut.

- *Comparison* dapat dilakukan secara paralel, ini berarti kompleksitas waktu pengurutan dimungkinkan untuk dapat dibuat lebih baik dari n log n.

- *Sorting network* memanfaatkan *comparison network* sedemikian sehingga *comparison* dapat dilakukan secara paralel.

2. Prinsip *Zero - one*

Prinsip *Zero-one* mengatakan bahwa jika *sorting network* bekerja secara benar ketika setiap *input* dinyatakan dalam set {0, 1}, maka dia juga akan bekerja dengan baik pada sebarang angka-angka *input*.

Lemma :

Jika suatu *comparison network* memproses input a = $<a_1, a_2, .., a_n>$ menjadi b = $<b_1, b_2, \ldots, b_n>$ maka untuk sembarang fungsi f yang monoton naik, *network* tsb. memproses f(a) menjadi f(b).

Teorema :

Jika suatu *comparison network* dengan *input* sebanyak n (terdiri dari 0 atau 1) dapat mengurutkan 2^n kombinasi dari *input* yang mungkin secara benar, maka hal ini juga akan berlaku untuk sembarang bilangan yang merupakan *input* dari *network* tsb.

Pembuktian (memakai kontradiksi)

- Network memproses bilangan biner dengan benar
- Untuk sembarang bilangan yang merupakan input, $< a_1, a_2, \ldots, a_n >$, terdapat unsur $a_i < a_j$ dimana *network* memproses sehingga a_j ada sebelum a_i pada *output*

- Definisikan

$$f(x) = \begin{cases} 0 & x <= a_i \\ 1 & x > a_j \end{cases}$$

- a_j diletakkan sebelum a_i pada *output*, menurut LEMMA, $f(a_j)$ juga akan diletakkan sebelum $f(a_i)$
- Sedangkan dipunyai $f(a_j)=1$ dan $f(a_i)=0$ → 1 muncul sebelum 0 pada output berarti bilangan biner tidak diproses dengan benar. (KONTRADIKSI !!!)

3. Bitonic Sorting Networks

Tahap pertama dalam konstruksi *sorting* yang efisien adalah untuk membentuk *comparison network* yang dapat mengurutkan beberapa *bitonic sequence*.

Contoh : bitonic sequence : {1, 4, 6, 8, 3, 2}

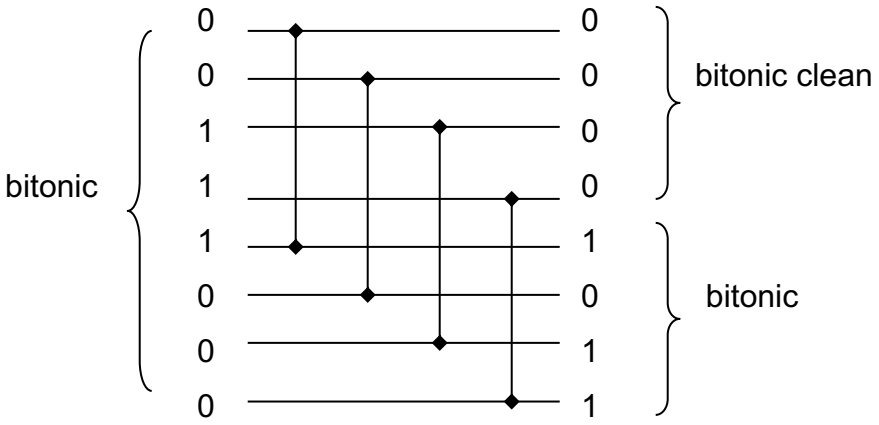
{9, 8, 3, 2, 4, 6}

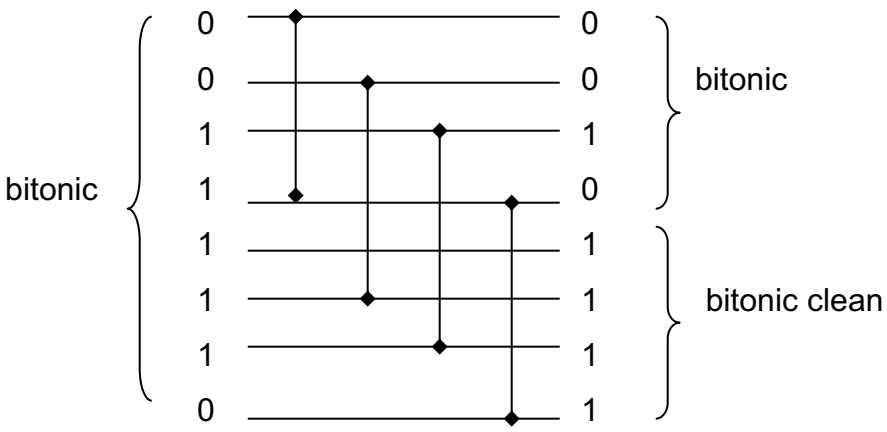
urutan yang terjadi pertama naik kemudian turun, atau pertama turun, kemudian naik.

3.1. Half Cleaner

Bitonic Sorter dibandingkan dari beberapa tahap, setiap tahap disebut *half cleaner*. Setiap *half cleaner* adalah *comparison network* dengan kedalaman (*depth*) 1 dimana input baris i dibandingkan dengan baris i + n/2 untuk setiap i=1, 2, 3, …, n/2 (diasumsikan n genap)

Berikut ini ditunjukkan *Half Cleaner[8],* yaitu *half cleaner* dengan 8 *input* dan 8 *output*. 2 contoh berbeda dengan *input zero-one* dan nilai *output* seperti ditunjukkan pada gambar berikut :





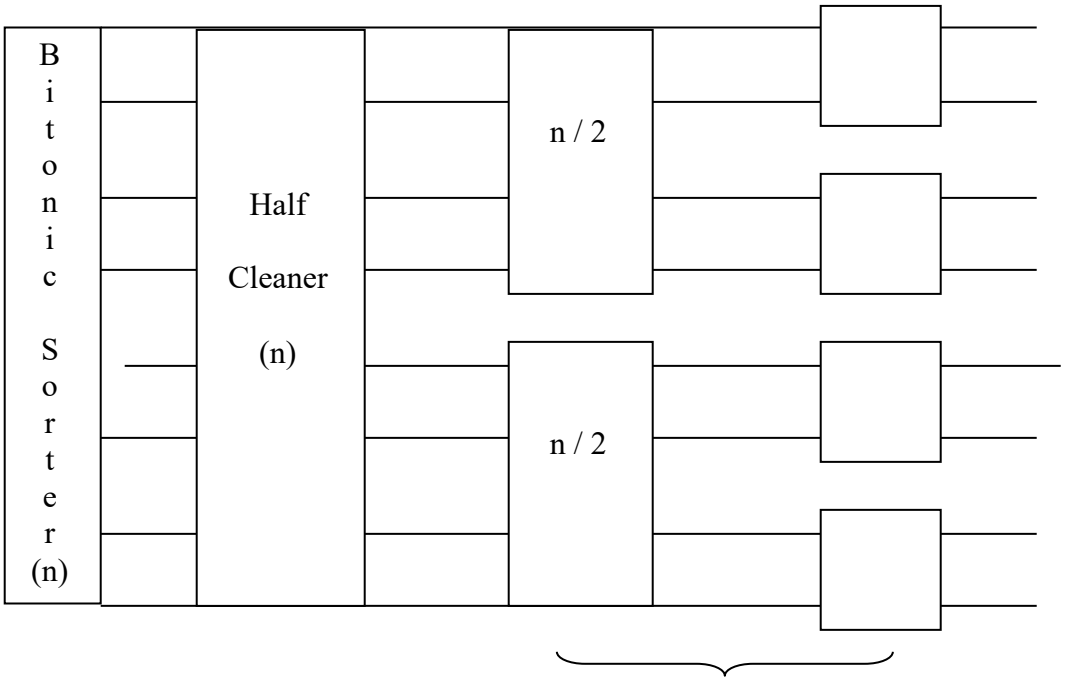
Input pada kabel/*line* ke i dibandingkan dengan input pada kabel/*line* ke – (i+n/2), untuk i =1,2, …, n/2

Perhatikan : separuh dari *output sequence* adalah *bitonic clean* dan nilai kecil ada di paruh bagian atas dan nilai besar diparuh bagian bawah.

3.2. *Bitonic Sorter*

- Dengan penggabungan *half cleaner* secara rekursif, seperti ditunjukkan pada gambar berikut kita dapat membentuk *bitonic sorter*, dimana jaringan mengurutkan *bitonic sequences*.

- Tahap pertama dari *Bitonic Sorter[n]* terdiri dari *Half Cleaner[n],* yang akan menghasilkan 2 *bitonic sequences* dari setengah ukuran



BITONIC SORTER

0	0	0	0
0	0	0	0
1	0	0	0
1	0	0	0
1	1	1	0
0	0	0	1
0	1	1	1
0	1	1	1

HALF CLEANER

- Jumlah *comparator* minimal = n log n
- Kedalaman = log n
- Tiap *stage* ada n/2 *comparator*
- Total = ½ n (log n)
- Kompleksitas ➔ O(n log n)

- Kedalaman dari *bitonic sorter* (n) dapat dinyatakan sebagai persamaan *recurrence* sebagai berikut :

$$D(n) = \begin{cases} 0 & n = 1 \\ D(n/2)+1 & n=2^k,\ k>=1 \end{cases}$$

Solusinya adalah D(n) = log(n)

- *Bitonic sorter* dengan kedalaman log n dapat dipakai untuk mengurutkan *bitonic sequence* yang ukurannya n (dengan input berupa bilangan biner)

- Dengan menggunakan *zero-one principle*, sembarang *bitonic sequence* dapat juga diurutkan dengan *bitonic sorter*

4. Merging Network

Merging Network :

- *Network* yang dipakai untuk menggabungkan dua *input sequence* yang sudah terurut menjadi satu *sequence* yang juga terurut
- Dibentuk dengan memodifikasi *bitonic sorter*

<u>Sebagai ilustrasi, perhatikan hal-hal berikut :</u>

⇒ *Bitonic Sorter* mengurutkan *bitonic sequence*

⇒ Apabila terdapat 2 *sequence* yang masing-masing sudah terurut :

⇒ Dua *sequence* tsb. dapat dibuat menjadi *bitonic sequence* jika salah satu *sequence* dibalik urutannya

Contoh

Sequence pertama → < 0 0 0 1 1 1 1 1 >, terurut

Sequence kedua → < 0 0 1 1 1 1 1 1 >, terurut dan kemudian dibalik menjadi : < 1 1 1 1 1 1 0 0 >

Setelah itu digabungkan kembali menjadi :

< 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 >

yang merupakan bitonic sequence yang dapat diurutkan dengan menggunakan BITONIC SORTER

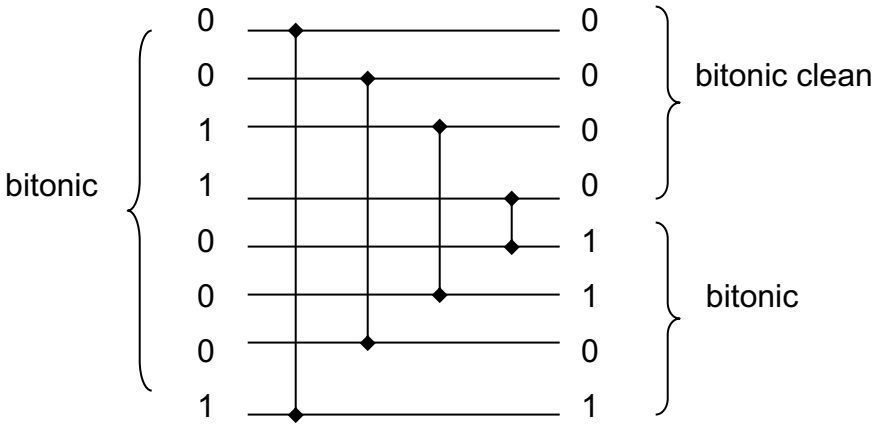
Cara mengkonstruksi *Merging Network* :

Pendekatan sederhana

- Balik urutan dari salah satu *sequence*
- Gabungkan kedua sequence sehingga menjadi *bitonic sequence*
- Gunakan BITONIC SORTER

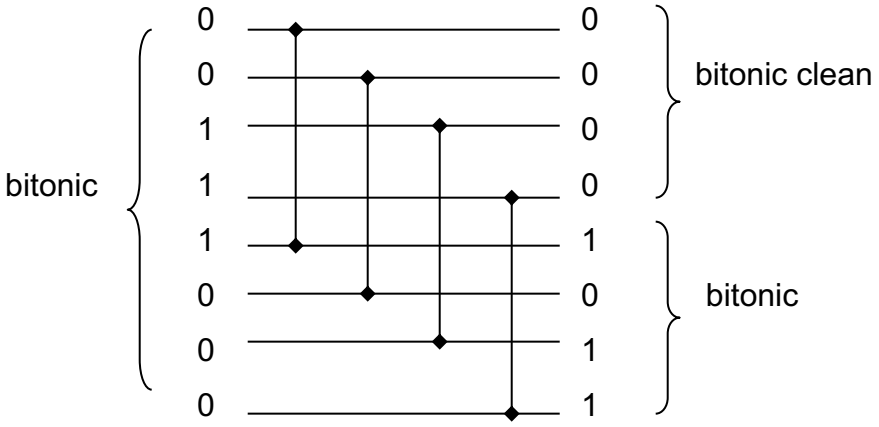
Alternatif lain

- Melakukan modifikasi pada *bitonic sorter* sehingga proses membalik urutan dari salah satu *input sequence*nya dapat dilakukan secara implisit.



Modified

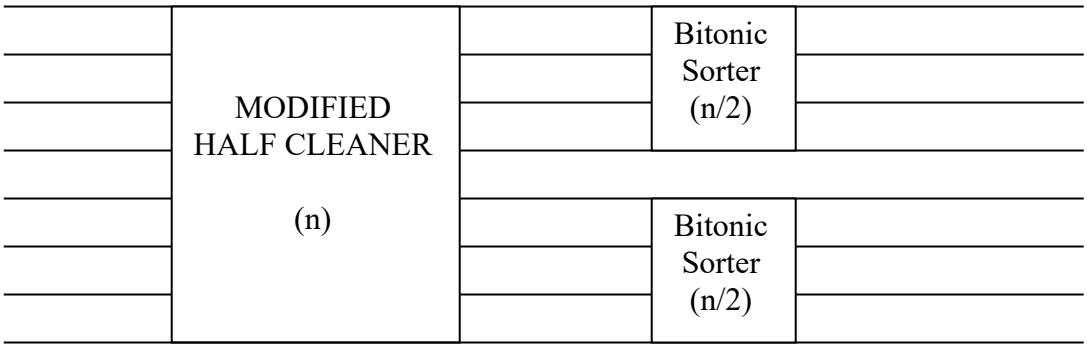
- *Input*nya dua *sequence* yang terurut
- *Output*nya dua *bitonic sequence* (*bitonic clean*, *bitonic sequence*)

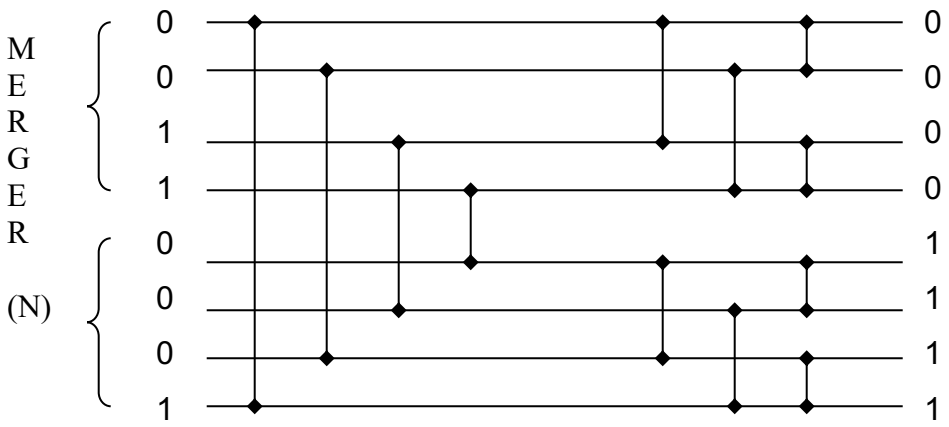


Half Cleaner

- *Input*nya *bitonic sequence*
- *Output*nya dua *bitonic sequence*

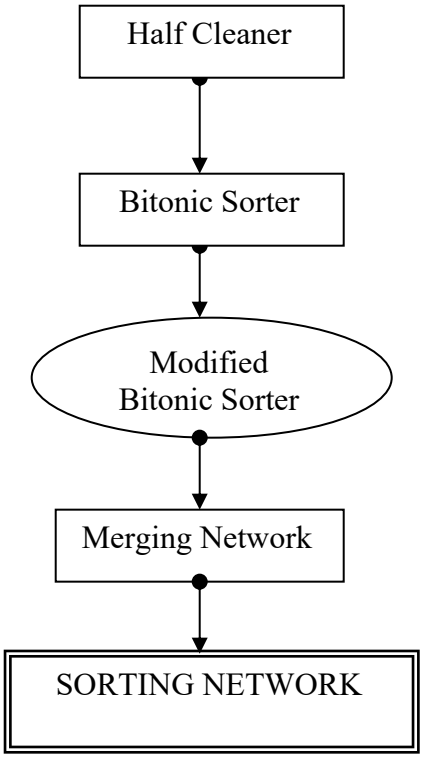
Dengan demikian bentuk *MERGING NETWORK*



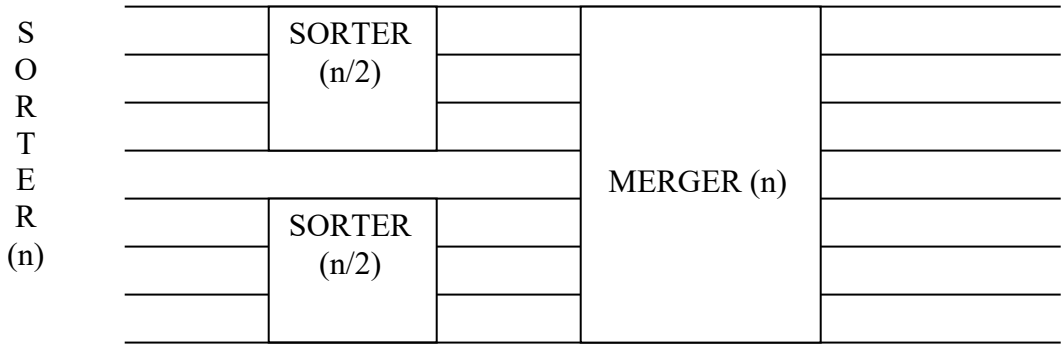


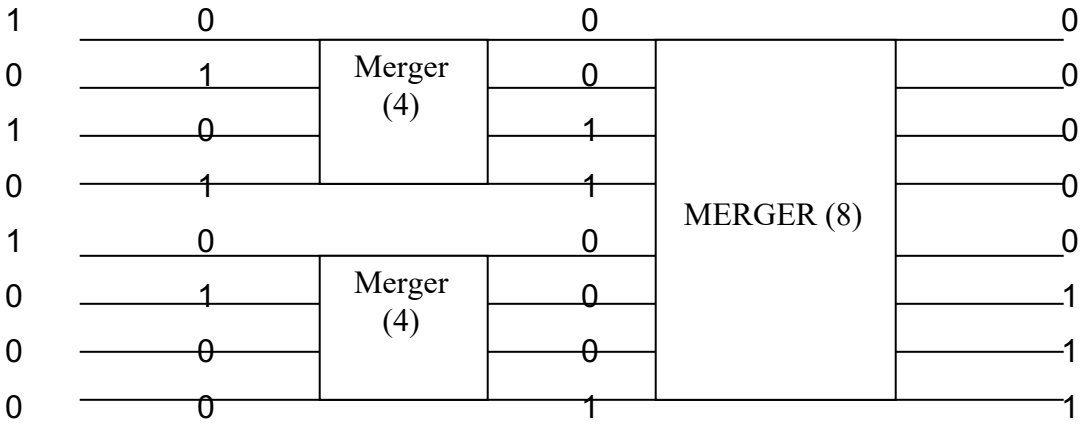
5. Sorting Networks

- *Sorting network* adalah *comparison network* dimana *output*nya monoton naik ($b_1 <= b_2 <= b_3 <= \ldots <= b_n$) untuk setiap input (yang banyaknya n) dari *network* tersebut.
- Perhatikan bahwa banyaknya *comparator* yang dipakai (=ukuran dari *network*) tergantung dari besarnya *input* dan *output* yang diproses oleh *network* tersebut.



Sorting network memanfaatkan *merging network* untuk melakukan *merge sort* secara paralel





- Kompleksitas Waktu dari *Sorting network* = kedalaman dari *network* tsb.
- D(n) = kedalaman dari *sorter(*n)
- D(n) terdiri dari D(n/2) + kedalaman *MERGER*(n)

$$D(n) = \begin{cases} 0 & n=1 \\ D(n/2) + \log(n) & n=2^k,\ k>=1 \end{cases}$$

Solusinya diperoleh → $D(n) = \Theta(\log^2(n))$

Referensi :

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, **Introduction to Algoritms**, Mc Graw-Hill, 1990