TUGAS
ANALISIS ALGORITMA

Disusun Oleh:

Rafa Azka Ulinnuha 140810200033

PROGRAM STUDI S-1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

JATINANGOR

2022

Soal Sorting

1. Insertion Sort

```
PROGRAM INSERTION SORT
{ membandingkan dan mengurutkan dua data pertama pada array, kemudian
membandingkan data para array berikutnya apakah sudah berada di
tempat semestinya }
DEKLARASI
int arr[n];
int key;
DEFINISI
for (i = 1; i < n; i++)
      key ← arr[i];
      j ← i - 1;
      while (j >= 0 && arr[j] > key)
            arr[j + 1] ← arr[j];
            j ← j - 1;
      endwhile
      arr[j + 1] ← key;
endfor
```

Analisis Kompleksitas Waktu

1	`for (i = 0; i < n; i++)`	1 + n + n - 1
2	`key ← arr[i];`	n – 1
3	`j ← i – 1;`	n – 1
4	`while (j >= 0 && arr[j] > key)`	(x + 1) + 2x
5	`arr[j + 1] ← arr[j];`	3x
6	`j ← j – 1;`	x
7	`endwhile`	
8	`arr[j + 1] ← key;`	2(n – 1)
9	`endfor`	

$x = 1 + 2 + 3 + \ldots + (n - 1)$

$= \frac{n-1\,(n-1+1)}{2}$

$= \frac{n\,(n-1)}{2}$

$T(n) = 1 + n + n - 1 + (n - 1) + (n - 1) + (x + 1) + 2x + 3x + x + 2(n - 1)$

$= 7x + 6n - 3$

$= 7(\frac{n\,(n-1)}{2}) + 6n - 3$

$= 3{,}5n^2 - 3{,}5n + 6n - 3$

$= 3{,}5n^2 + 2{,}5n - 3$

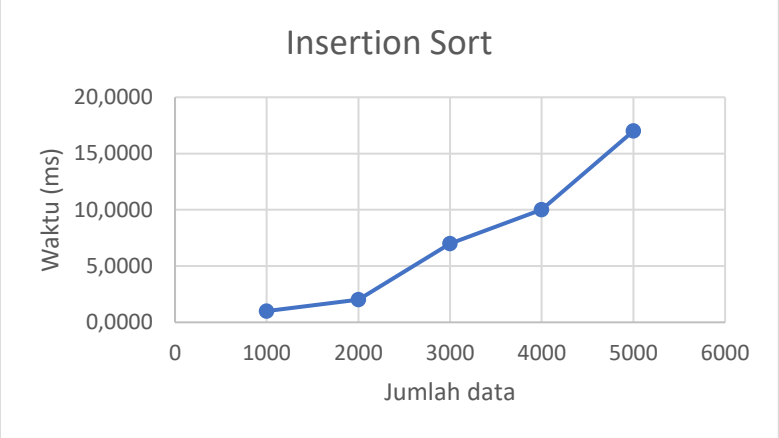
$= O(n^2)$

Maka, waktu kompleksitas Insertion Sort adalah $O(n^2)$.

Tabel waktu eksekusi algoritma Insertion Sort, yaitu sebagai berikut:

Jumlah data	Waktu (ms)
1000	1,0004
2000	2,0009
3000	7,0022
4000	10,0089
5000	17,0025

Kompleksitas algoritma Insertion Sort dapat di gambarkan seperti grafik dibawah ini:

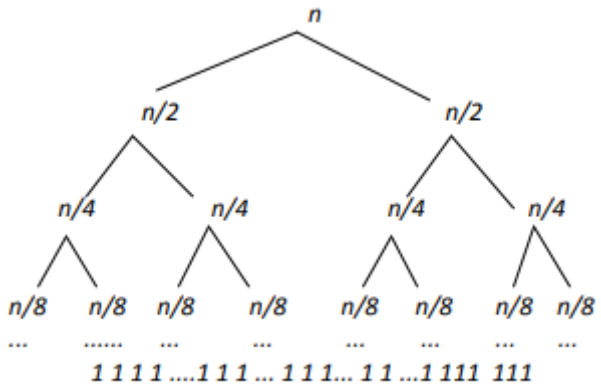


2. Quick Sort

Analisis Kompleksitas Waktu

a) Best Case

Kasus terbaik terjadi bila pivot adalah elemen median sedemikian sehingga kedua upa-tabel berukuran relatif sama setiap kali pempartisian.



Kompleksitas waktu pengurutan dihitung dari jumlah perbandingan elemen-elemen tabel:

$T_{min}(n)$ = waktu partisi + waktu pemanggilan rekurens

Kompleksitas prosedur partisi adalah t(n) = cn = O(n), sehingga kompleksitas algoritma quick sort menjadi

$$T(n) = \begin{cases} a & ,n=1 \\ 2T\left(\frac{n}{2}\right) + cn & ,n>1 \end{cases}$$

Penyelesaian persamaan rekurens:

T(n) = 2T(n/2) + cn

= 2(2T(n/4) + cn/2) + cn = 4(T(n/4) + 2cn

= 4(2(T(n/8) + cn/4) + 2cn = 8T(n/8) + 3cn

= ..

= 2^k(T(n/2^k) + kcn

Persamaan terakhir dapat diselesaikan karena basis rekursif adalah ketika ukuran tabel = 1,

$n/2^k = 1 \rightarrow k = {}^2\log n$

sehingga

T(n) = nT(1) + $cn^2 \log n$

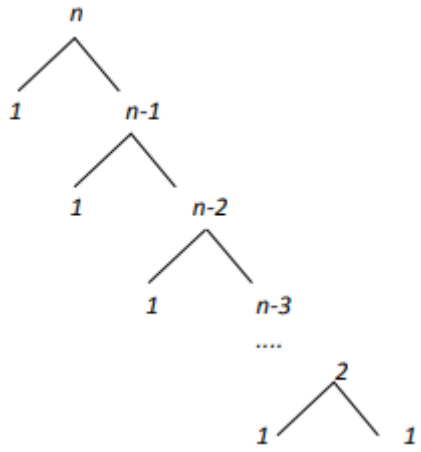
= na + $cn^2 \log n$

= O($n^2 \log n$)

Maka, waktu kompleksitas Quick Sort dengan keadaan best case adalah O($n^2 \log n$).

b) Worst Case

Kasus ini terjadi bila pada setiap partisi pivot selalu elemen masksimum atau elemen minimum sebuah tabel sehingga pembagian menghasilkan upatabel kiri atau kanan berukuran satu elemen dan upatabel kanan atau kiri berukuran n – 1 elemen. Keadaan kedua upa tabel ini digambarkan sebagai berikut:



Kompleksitas waktu pengurutan :

$$T(n) = \begin{cases} a & ,n=1 \\ T(n-1)+cn & ,n>1 \end{cases}$$

T(n) = cn + T(n-1)

= bn + { b.(n-1) + T(n-2) }

= bn + b(n-1) + { b(n-2) + T(n-3) }

= ...

= b(n+(n-1)+(n-2)…+2) +a

= b{(n-1)(n+2)/2} + a

= $bn^2/2$ + bn/2 + ((a-b)

= $O(n^2)$

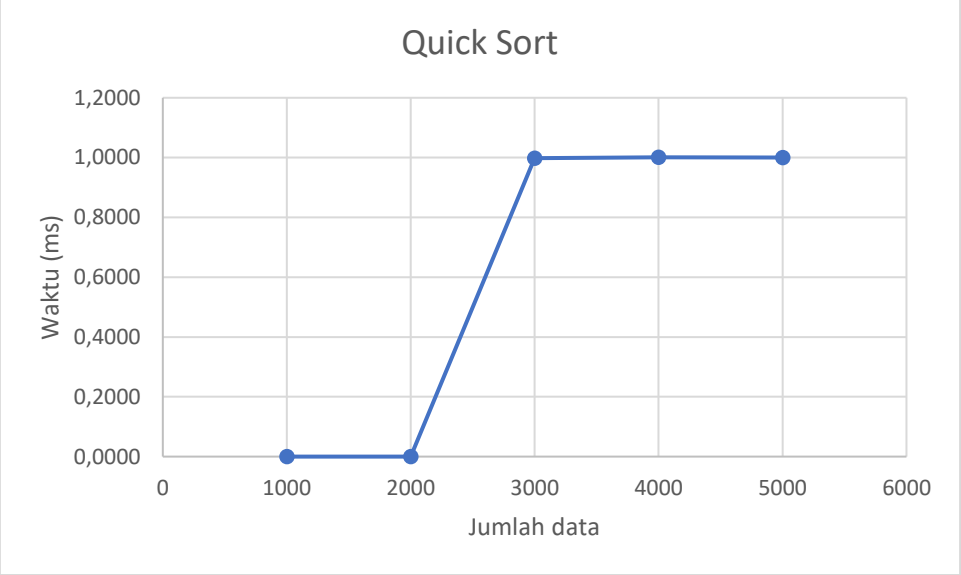
Maka, waktu kompleksitas Quick Sort dengan keadaan worst case adalah $O(n^2)$.

Tabel waktu eksekusi algoritma Quick Sort, yaitu sebagai berikut:

Jumlah data	Waktu (ms)
1000	0,0000
2000	0,0000
3000	0,9985

4000	1,0009
5000	0,9998

Kompleksitas algoritma Quick Sort dapat di gambarkan seperti grafik dibawah ini:



c) Merge Sort

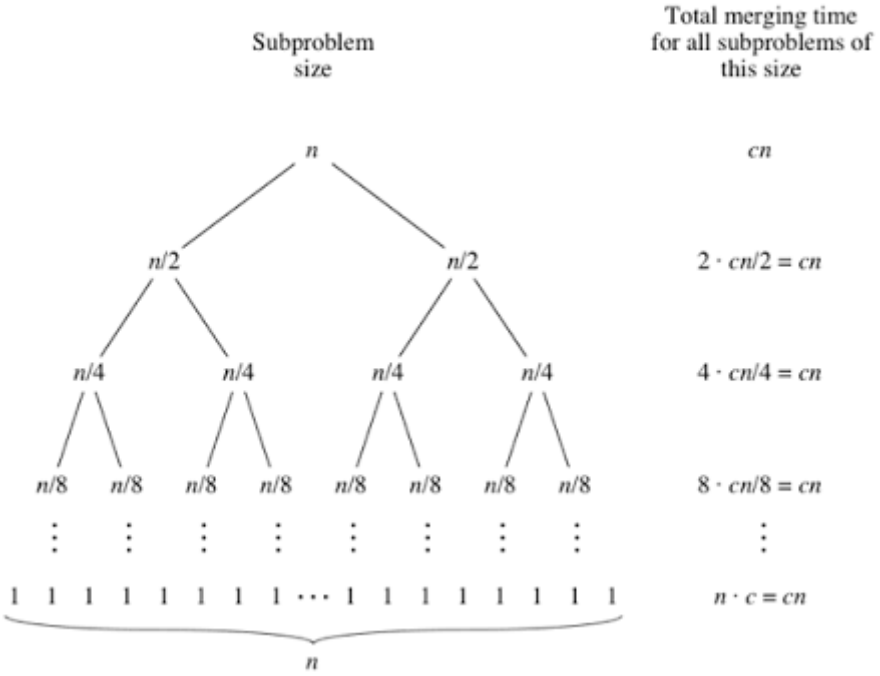
Analisis Kompleksitas Waktu

Misalkan, terdapat array A berukuran n. Merge Sort membagi array ini menjadi dua bagian yang sama dan mengurutkannya satu per satu. Bagi simpul A[L, R] menjadi dua simpul A[L, M] dan A[M+1,R] dengan M = (L+R)/2.

Pemisahan simpul A[L, R] menjadi dua simpul membutuhkan waktu R−L+1 dan kemudian menggabungkan dua simpul A[L,M] dan A[M+1,R] lagi membutuhkan waktu A[R−L+1]. Jadi untuk setiap simpul, jumlah operasi yang dilakukan algoritma sama dengan dua kali ukuran array yang sesuai dengan simpul tersebut.

Jadi pada level tertentu jika kita memiliki larik berukuran k, pemisahan dan penggabungan larik dapat dilakukan dalam operasi k + 2 × k/2 = 2k.

Perhatikan bahwa kita terus membelah array sampai kita memiliki array berukuran 1 karena kita tidak dapat membaginya lebih jauh.



Waktu komputasi yang dihabiskan pada masing-masing simpul ini hanya dua kali ukuran arra. Oleh karena itu total waktu berjalan S dari Merge Sort hanyalah jumlah dari semua ukuran array yang sesuai dengan setiap simpul dari pohon di atas.

$$S = 2\sum_{i=0}^{k} 2^i \frac{n}{2^i}$$

Bagaimana mendapatkan jumlah seperti itu? Terdapat 2^i simpul dengan ukuran $n/2^i$ dalam pohon dan membutuhkan 2k waktu untu menyelesaikan komputasi pada array dengan ukuran k.

Perhatikan bahwa Ketika i = k, $n/2^k = 1 \rightarrow n = 2^k \rightarrow k = \log n$, maka S tereduksi menjadi

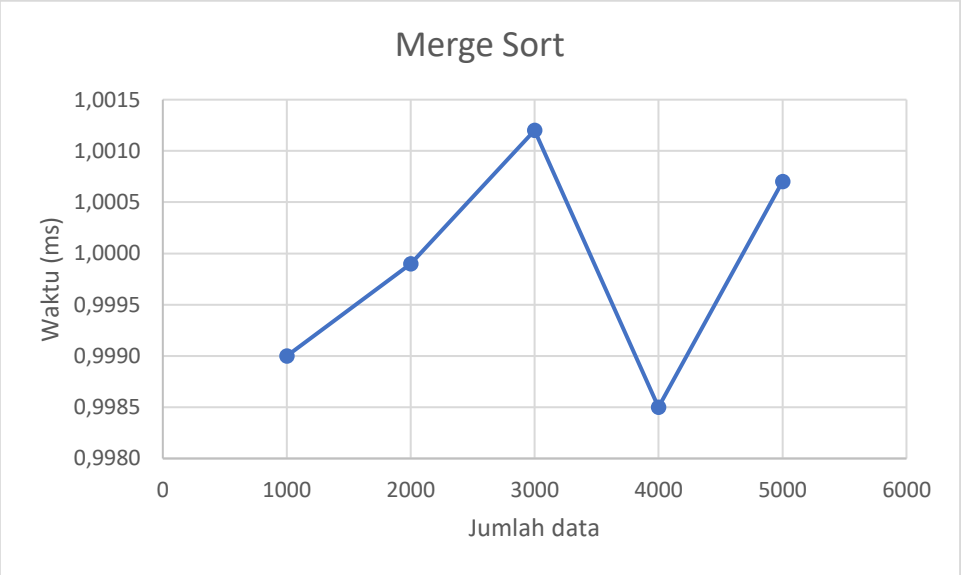
$$2\sum_{i=0}^{\lceil \log n \rceil} n = 2n\lceil \log n \rceil = \mathcal{O}(n \log n)$$

Tabel waktu eksekusi algoritma Merge Sort, yaitu sebagai berikut:

Jumlah data	Waktu (ms)
1000	0,9990
2000	0,9999

3000	1,0012
4000	0,9985
5000	1,0007

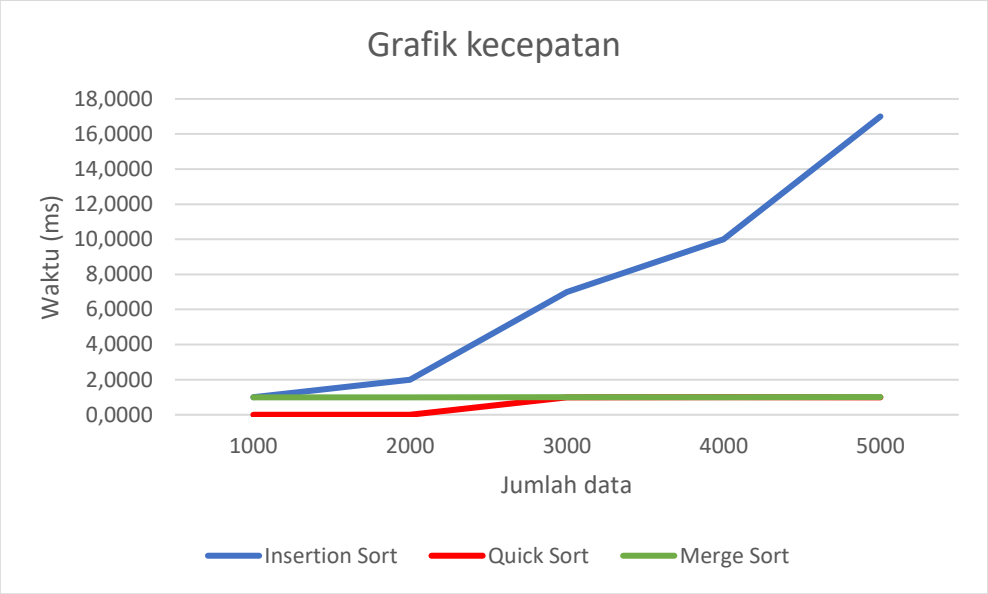
Kompleksitas algoritma Merge Sort dapat di gambarkan seperti grafik dibawah ini:



Tabel waktu eksekusi program dengan jumlah data yang berbeda

No	Jumlah Data	Insertion Sort	Quick Sort	Merge Sort
		Waktu Eksekusi Program (s)		
1	1000	0.0010004	0	0.000999
2	2000	0.0020009	0	0.0009999
3	3000	0.0070022	0.0009985	0.0010012
4	4000	0.0100089	0.0010009	0.0009985
5	5000	0.0170025	0.0009998	0.0010007

No	Jumlah Data	Insertion Sort	Quick Sort	Merge Sort
		Waktu Eksekusi Program (ms)		
1	1000	1,0004	0,0000	0,9990
2	2000	2,0009	0,0000	0,9999
3	3000	7,0022	0,9985	1,0012
4	4000	10,0089	1,0009	0,9985
5	5000	17,0025	0,9998	1,0007



Dari grafik diatas, terlihat jelas bahwa waktu eksekusi setiap algoritma sorting dengan menggunakan data yang sama berbeda-beda. Hal ini disebabkan oleh kompleksitas waktu yang dimiliki oleh setiap algoritma sorting. Tingkat kompleksitas suatu algoritma memiliki urutan sebagai berikut:

$$O(1)<O(\log n)<O(n)<O(n \log n)<O(n^2)<\ldots<O(2^n)<O(n!)$$

Jika kompleksitasnya semakin kecil, maka waktu runningnya akan semakin cepat. Oleh karena itu, dapat dilihat bahwa merge menjadi algoritma tercepat dan insertion sort menjadi algoritma terlambat dari ketiga algoritma sorting.

Output Program

```
Insertion Sort
Durasi hasil running    : 0.0010004s
Durasi hasil running    : 0.0020009s
Durasi hasil running    : 0.0070022s
Durasi hasil running    : 0.0100089s
Durasi hasil running    : 0.0170025s
```

```
Quick Sort
Durasi hasil running    : 0s
Durasi hasil running    : 0s
Durasi hasil running    : 0.0009985s
Durasi hasil running    : 0.0010009s
Durasi hasil running    : 0.0009998s
```

```
Merge Sort
Durasi hasil running    : 0.000999s
Durasi hasil running    : 0.0009999s
Durasi hasil running    : 0.0010012s
Durasi hasil running    : 0.0009985s
Durasi hasil running    : 0.0010007s
```