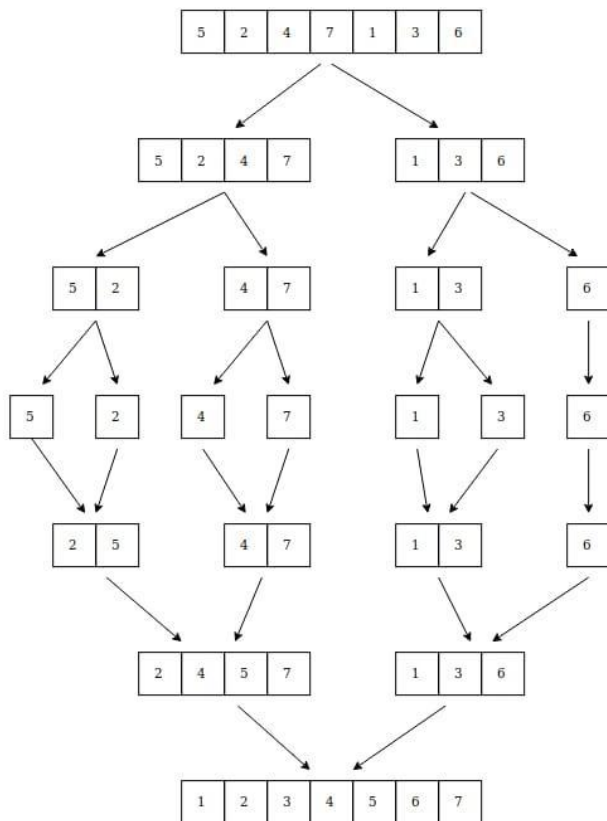


## Merge Sort

### 1. Definisi dan Cara Kerja Algoritma Merge Sort

Mergin sort adalah algoritma pengurutan, yang umum digunakan oleh komputer. Mergin sort merupakan metode pengurutan yang menggunakan pola divide and conquer. Merge Sort melakukan pengurutan dengan memecah array menjadi 2 bagian secara terus menerus (rekursif) hingga hanya berisi 1 element, kemudian item tersebut diurutkan dan digabungkan kembali membentuk 1 array dengan element yang terurut. Jadi Merge Sort pertama-tama memecah array menjadi bagian yang sama dan kemudian menggabungkannya dengan cara yang diurutkan

### 2. Contoh Ilustrasi Algoritma



Merge Sort

### 3. Code

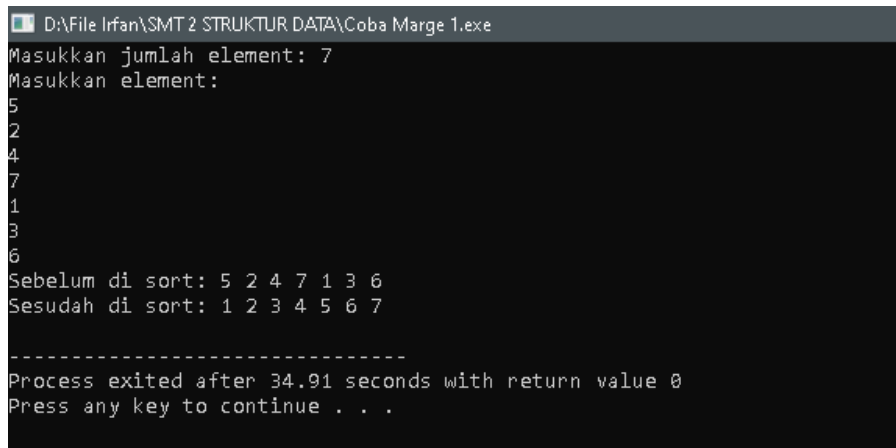
```
4. //Merge Sort c++
5.
6. #include<iostream>
7. using namespace std;
8. void swapping(int &a, int &b) {      //swap a dan b
9.     int temp;
10.    temp = a;
11.    a = b;
12.    b = temp;
13.}
14.void display(int *array, int size) {
15.    for(int i = 0; i<size; i++)
16.        cout << array[i] << " ";
17.    cout << endl;
18.}
19.void merge(int *array, int l, int m, int r) {
20.    int i, j, k, nl, nr;
21.    //ukuran/size sub-arrays kanan dan kiri
22.    nl = m-1+1; nr = r-m;
23.    int larr[nl], rarr[nr];
24.    //isi sub-arrays kanan dn kiri
25.    for(i = 0; i<nl; i++)
26.        larr[i] = array[l+i];
27.    for(j = 0; j<nr; j++)
28.        rarr[j] = array[m+1+j];
29.    i = 0; j = 0; k = l;
30.
31.    while(i < nl && j<nr) {
32.        if(larr[i] <= rarr[j]) {
33.            array[k] = larr[i];
34.            i++;
35.        }else{
36.            array[k] = rarr[j];
37.            j++;
38.        }
39.        k++;
40.    }
41.    while(i<nl) {      //element tambahan di kiri
42.        array[k] = larr[i];
43.        i++; k++;
44.    }
45.    while(j<nr) {      //element tambahan di kanan
46.        array[k] = rarr[j];
47.        j++; k++;
48.    }
49.}
50.void mergeSort(int *array, int l, int r) {
51.    int m;
```

```

52.     if(l < r) {
53.         int m = l+(r-l)/2;
54.         // urutan array pertama dan kedua
55.         mergeSort(array, l, m);
56.         mergeSort(array, m+1, r);
57.         merge(array, l, m, r);
58.     }
59.}
60.int main() {
61.    int n;
62.    cout << "Masukkan jumlah element: ";
63.    cin >> n;
64.    int arr[n];    //buat array dengan jumlah elemen yang
                    diinputkan
65.    cout << "Masukkan element:" << endl;
66.    for(int i = 0; i<n; i++) {
67.        cin >> arr[i];
68.    }
69.    cout << "Sebelum di sort: ";
70.    display(arr, n);
71.    mergeSort(arr, 0, n-1);    //(n-1) untuk indeks terakhir
72.    cout << "Sesudah di sort: ";
73.    display(arr, n);
74.}

```

Output :



```

D:\File Ifan\SMT 2 STRUKTUR DATA\Coba Marge 1.exe
Masukkan jumlah element: 7
Masukkan element:
5
2
4
7
1
3
6
Sebelum di sort: 5 2 4 7 1 3 6
Sesudah di sort: 1 2 3 4 5 6 7

-----
Process exited after 34.91 seconds with return value 0
Press any key to continue . . .

```

#### 4. Kompleksitas Waktu

Kompleksitas waktu untuk semua kasus Algoritma merge sort adalah  $O(n \log n)$ . Kita bisa menerapkan pembagian general dan teorema Conquer untuk menghitung kompleksitas dari algoritma merge sort. Kita ketahui bahwa

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^k)$$

n merupakan jumlah masalah utama  
a merupakan jumlah dari sub-masalah  
n/b merupakan jumlah dari sub-masalah  
dan  $O(n^k)$  adalah kompleksitas dari pembagian dan operasi merging.

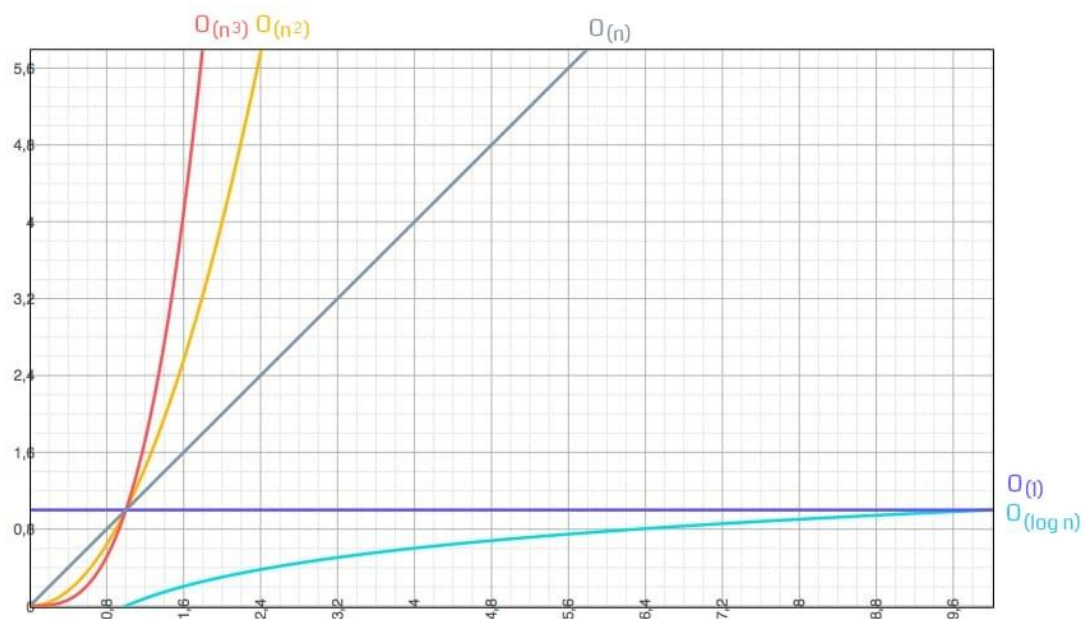
$$T(n) = O(n^{\log_b a}) \text{ jika } a > b^k$$

$$T(n) = O(n^k \log n) \text{ jika } a = b^k$$

$$T(n) = O(n^k) \text{ jika } a < b^k$$

Pada merge sort, masalah utama dibagi kedalam 2 sub-masalah, jadi  $a = 2$ . Jumlah dari masalah utama dibagi dengan 2, jadi  $b = 2$ . Kompleksitas untuk pembagian adalah  $O(1)$  dan kompleksitas waktu untuk merging adalah  $O(n)$ . Total waktu untuk pembagian dan merging adalah  $O(1) + O(n) = O(n) = O(n^1)$ . Jadi,  $k = 1$ .

Karena  $a = b^k$  maka kompleksitas dari merge sort adalah  $O(n^k \log n) = O(n \log n)$ .



N	1	5	10
$O(n \log n)$	2	11	33

## 5. Jelaskan Kelebihan dan Kekurangan Merge Sort

Kelebihan ini jauh lebih cepat daripada algoritma pengurutan sederhana seperti bubble sort ketika berhadapan dengan kumpulan data besar. Tidak seperti beberapa algoritma, kompleksitas waktu terbaik, rata-rata dan terburuk adalah  $O(n \log n)$ . Tidak seperti bubble sort, ini sekali lagi cukup konsisten.

Kekurangan Merge sort ini tidak sesederhana algoritma pengurutan yang tidak terlalu rumit. Misalkan Anda berurusan dengan array 20 item. Perbedaan antara kompleksitas waktu akan sangat nominal. Juga, tidak seperti algoritma pengurutan yang lebih sederhana, kompleksitas ruangnya adalah  $O(n)$ , di mana kompleksitas ruang Bubble Sort adalah  $O(1)$ .