

## Laporan Individu

### 1. Pengertian konsep MergeSort

Merge sort merupakan algoritma pengurutan dalam ilmu komputer yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data yang tidak memungkinkan untuk ditampung dalam memori komputer karena jumlahnya yang terlalu besar. Algoritma ini ditemukan oleh John von Neumann pada tahun 1945. Merge sort merupakan salah satu metode dari enam metode dalam melakukan pengurutan atau sorting

Algoritma Merge Sort adalah salah satu algoritma modern yang mirip seperti algoritma Quick Sort.

Keduanya sama-sama menggunakan metode Devide and Conquer. dimana sebuah list akan dipecah menggunakan fungsi rekursif

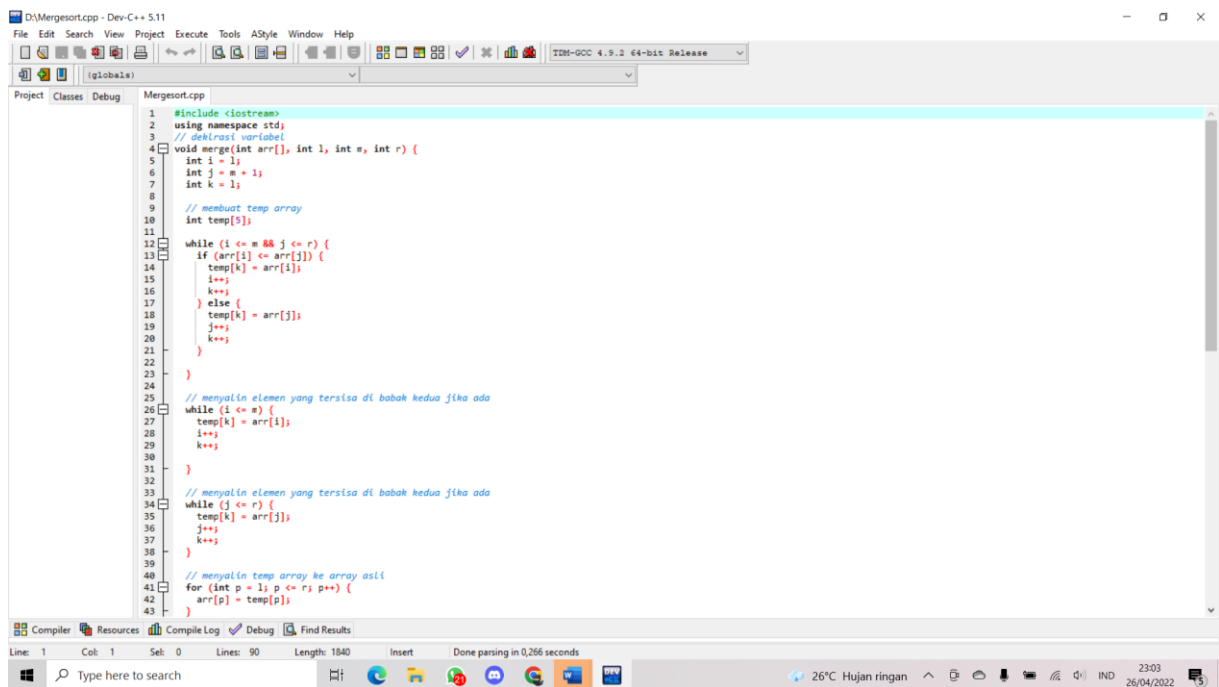
pertama kita akan membuat 2 blok dari sebuah list, dengan cara membelahnya menjadi 2 bagian sama rata, jika ternyata list tersebut jumlahnya ganjil, maka akan dibulatkan.

setelah menjadi 2 blok, masing masing blok akan di pecah kembali menggunakan fungsi rekursif hingga setiap blok hanya memiliki 1 index list.

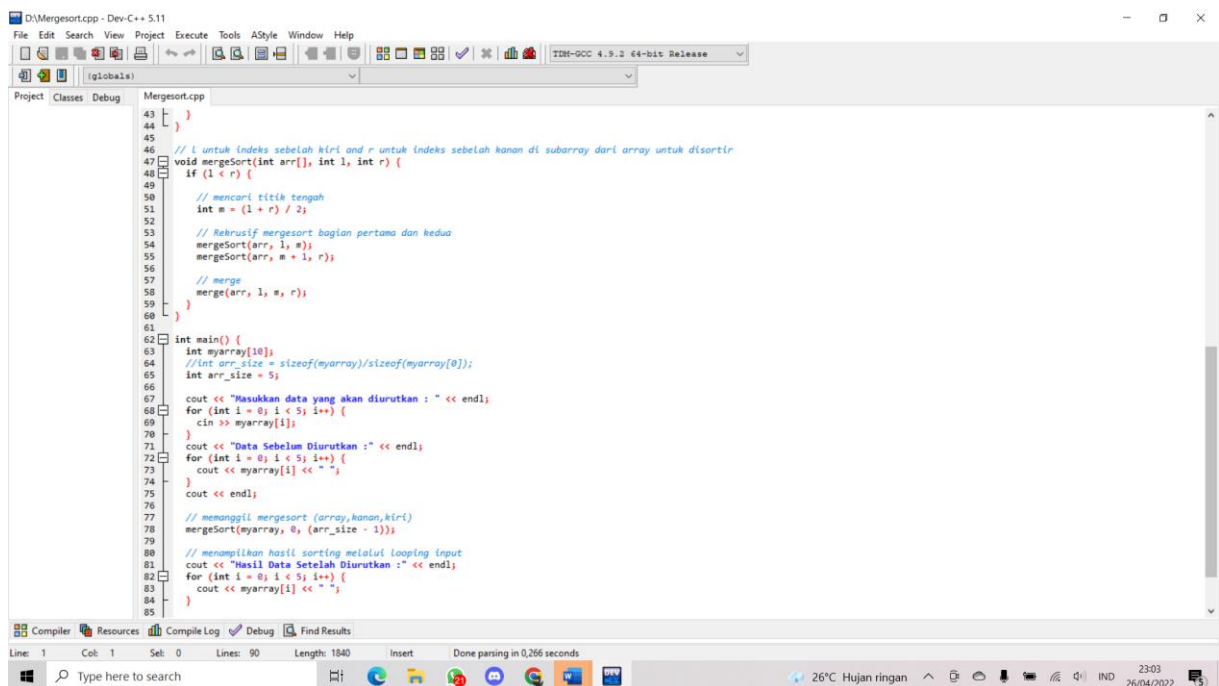
jika sudah, kita akan menyatukan 2 blok menjadi satu dan melakukan sorting antara 2 elemen blok tadi, jika ascending maka index dengan angka paling kecil akan berada disebelah kiri dan sebaliknya.

kita akan mengulang proses perbandingan tersebut hingga hanya tersisa 1 blok dengan index yang sudah tersorting.

## 2. Codongan MergeSort



```
1 #include <iostream>
2 using namespace std;
3 // deklarasi variabel
4 void merge(int arr[], int l, int m, int r) {
5     int i = l;
6     int j = m + 1;
7     int k = l;
8
9     // membuat temp array
10    int temp[5];
11
12    while (i <= m && j <= r) {
13        if (arr[i] <= arr[j]) {
14            temp[k] = arr[i];
15            i++;
16        } else {
17            temp[k] = arr[j];
18            j++;
19        }
20        k++;
21    }
22
23    // menyalin elemen yang tersisa di babak kedua jika ada
24    while (i <= m) {
25        temp[k] = arr[i];
26        i++;
27        k++;
28    }
29
30    // menyalin elemen yang tersisa di babak kedua jika ada
31    while (j <= r) {
32        temp[k] = arr[j];
33        j++;
34        k++;
35    }
36
37    // menyalin temp array ke array asli
38    for (int p = l; p <= r; p++) {
39        arr[p] = temp[p];
40    }
41 }
```



```
43 }
44 }
45
46 // l untuk indeks sebelah kiri dan r untuk indeks sebelah kanan di subarray dari array untuk disortir
47 void mergeSort(int arr[], int l, int r) {
48     if (l < r) {
49
50         // mencari titik tengah
51         int m = (l + r) / 2;
52
53         // Rekursif mergesort bagian pertama dan kedua
54         mergeSort(arr, l, m);
55         mergeSort(arr, m + 1, r);
56
57         // merge
58         merge(arr, l, m, r);
59     }
60 }
61
62 int main() {
63     int myarray[10];
64     //int arr_size = sizeof(myarray)/sizeof(myarray[0]);
65     int arr_size = 5;
66
67     cout << "Masukkan data yang akan diurutkan : " << endl;
68     for (int i = 0; i < 5; i++) {
69         cin >> myarray[i];
70     }
71     cout << "Data Sebelum Diurutkan : " << endl;
72     for (int i = 0; i < 5; i++) {
73         cout << myarray[i] << " ";
74     }
75     cout << endl;
76
77     // memanggil mergesort (array,kanan,kiri)
78     mergeSort(myarray, 0, (arr_size - 1));
79
80     // menampilkan hasil sorting melalui looping input
81     cout << "Hasil Data Setelah Diurutkan : " << endl;
82     for (int i = 0; i < 5; i++) {
83         cout << myarray[i] << " ";
84     }
85 }
```

Code fungsi mergesort

```

6 // Menggabungkan dua subarray L dan M menjadi arr
7 void merge(int arr[], int p, int q, int r) {
8
9     // Membuat salinan dari subarray L A[p..q] dan M A[q+1..r]
10    int n1 = q - p + 1;
11    int n2 = r - q;
12
13    int L[n1], M[n2];
14
15    for (int i = 0; i < n1; i++)
16        L[i] = arr[p + i];
17    for (int j = 0; j < n2; j++)
18        M[j] = arr[q + 1 + j];
19
20    // Pertahankan indeks sub-array dan larik utama saat ini
21    int i, j, k;
22    i = 0;
23    j = 0;
24    k = p;
25
26    // Sampai kita mencapai salah satu ujung L atau M, pilih yang lebih besar di antara
27    // elemen L dan M dan letakkan di posisi yang benar di A[p..r]
28    while (i < n1 && j < n2) {
29        if (L[i] <= M[j]) {
30            arr[k] = L[i];
31            i++;
32        } else {
33            arr[k] = M[j];
34            j++;
35        }
36        k++;
37    }
38
39    // Saat kita kehabisan elemen di L atau M,
40    // ambil elemen yang tersisa dan masukkan ke A[p..r]
41    while (i < n1) { // Kita keluar dari loop sebelumnya karena j < n2 tidak berlaku
42        arr[k] = L[i];
43        i++;
44        k++;
45    }
46
47    while (j < n2) { // Kita keluar dari loop sebelumnya karena i < n1 tidak berlaku
48        arr[k] = M[j];
49        j++;
50        k++;
51    }
52 }
53
54 // Bagilah array menjadi dua subarray, urutkan dan gabungkan
55 void mergeSort(int arr[], int l, int r) {
56     if (l < r) {
57         // m adalah titik di mana array dibagi menjadi dua subarray
58         int m = l + (r - l) / 2;
59
60         mergeSort(arr, l, m);
61         mergeSort(arr, m + 1, r);
62
63         // Gabungkan subarray yang diurutkan
64         merge(arr, l, m, r);
65     }
66 }

```

Penjelasan Algoritma Merge Sort:

Sebagai contoh kita memiliki list dengan urutan acak sebagai berikut :

4	20	2	9	13
---	----	---	---	----

Sesuai dengan konsep yang sudah di jelaskan algoritma ini meggunakan metode devide and conquer, jadi akan kita bagi menjadi 2 bagian.

4	20	2	9	13
---	----	---	---	----

4	20	3		9	13
---	----	---	--	---	----

Seperti yang kita lihat elemen ini belum menjadi elemen tunggal maka dari itu kita bagi lagi hingga menjadi elemen tunggal.

4	20	3		9	13
---	----	---	--	---	----

4	20		3	9	13
---	----	--	---	---	----

4	20	3		9	13
---	----	---	--	---	----

Nah, sekarang sudah menjadi elemen tunggal, selanjutnya kita akan melakukan sorting dengan kebalikan dari metode devide and conquer, dimana kita akan melakukan perbandingan antara 2 elemen dan menggabungkannya menjadi elemen baru

4	20	3		9	13
---	----	---	--	---	----

4	20		3	9	13
---	----	--	---	---	----

Selanjutnya kita akan membandingkan dua elemen Kembali hingga semua item dalam list tersusun ascending.

4	20	3		9	13
---	----	---	--	---	----

4	20		3	9	13
---	----	--	---	---	----

3	4	20		9	13
---	---	----	--	---	----

3	4	9		20	13
---	---	---	--	----	----

Dan ini hasil dari sortiran menggunakan merge sort

3	4	9	13	20
---	---	---	----	----

Langkah-langkah fungsi merge() =

```
1 void merge(int arr[], int p, int q, int r) {  
2     // Here, p = 0, q = 3, r = 5 (size of array)
```

Langkah 1: Buat salinan duplikat dari sub-array untuk diurutkan

```
// Create L A[p..q] and M A[q+1..r]  
int n1 = q - p + 1 = 2 - 0 + 1 = 3;  
int n2 = r - q = 5 - 3 = 2;  
  
int L[4], M[2];  
  
for (int i = 0; i < n1; i++)  
    L[i] = arr[p + i];  
    // L[0,1,2,3] = A[0,1,2,3] = [4,20,3]  
  
for (int j = 0; j < n2; j++)  
    M[j] = arr[q + 1 + j];  
    // M[0,1] = A[4,5] = [9,13]
```

Langkah 2: Pertahankan indeks sub-array dan larik utama saat ini

```
int i, j, k;  
i = 0;  
j = 0;  
k = p;
```

Langkah 3: Sampai kita mencapai ujung L atau M, ambil yang lebih besar di antara elemen L dan M dan letakkan di posisi yang benar di A[p..r]

```
while (i < n1 && j < n2) {  
    if (L[i] <= M[j]) {  
        arr[k] = L[i]; i++;  
    }  
    else {  
        arr[k] = M[j];  
        j++;  
    }  
    k++;  
}
```

Langkah 4: Saat kita kehabisan elemen di L atau M, ambil elemen yang tersisa dan masukkan ke A[p..r]

```
while (i < n1)  
{  
    arr[k] = L[i];  
    i++;  
    k++;  
}
```

```

while (j < n2)
{
    arr[k] = M[j];
    j++;
    k++;
}

```

Langkah ini diperlukan jika ukuran M lebih besar dari L.

Di akhir fungsi gabungan, subarray A[p..r] diurutkan.

Bukti hasil run

```

D:\Mergesort.exe
Masukkan data yang akan diurutkan :
4
20
3
9
13
Data Sebelum Diurutkan :
4 20 3 9 13
Hasil Data Setelah Diurutkan :
3 4 9 13 20

-----
Process exited after 23.48 seconds with return value 0
Press any key to continue . . .

```

### 3. Jenis Big O

#### MergeSort Complexity

Time Complexity	
Best	$O(n \cdot \log n)$
Worst	$O(n \cdot \log n)$
Average	$O(n \cdot \log n)$
Space Complexity	$O(n)$
Stability	Yes

#### Time Complexity

Best Case Complexity:  $O(n \cdot \log n)$

Worst Case Complexity:  $O(n \cdot \log n)$

Average Case Complexity:  $O(n \cdot \log n)$

#### Space Complexity

The space complexity of merge sort is  $O(n)$ .

MergeSort (A, l, r) = T(n).

untuk perintah operasi if (l < r) dan q = l adalah perintah operasi untuk membagi dua dari n data. Sehingga, perintah operasi setelahnya yaitu mergeSort(arr, l, q); mergeSort(arr, q + 1, r); menerima data sebanyak n/2.

Kompleksitas waktu dari kedua operasi perintah tersebut adalah T(n/2) sedangkan kompleksitas waktu dari operasi merge adalah O(n).

Jadi persamaan kompleksitas waktu untuk merge sort adalah

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = 2\left(T\left(\frac{n}{2}\right)\right) + O(n)$$

dengan  $n > 1; n \in \mathbb{Z}$

Persamaan diatas merupakan persamaan rekrusif sehingga kompleksitas waktu dapat di cari menggunakan Master theorem.

Diketahui :

a= 2 b= 2 f(n) = O(n) d = 1 maka  $a=b^d$  yang memenuhi syarat.

Sehingga, T(n) = O(n log n) jadi worst case dari Algoritma

Merge sort adala O (n log n).

N	1	5	10
O(n log n)	2	160	1.024

#### 4. Kelebihan dan kekurangan dari MergeSort.

Kelebihan MergeSort :

- Performa sangat bagus untuk list yang memiliki banyak index
- Memiliki waktu pengerjaan yang konsisten
- Dibanding dengan algoritma lain, merge sort ini termasuk algoritma yang sangat efisien dalam penggunaannya sebab setiap list selalu dibagi bagi menjadi list yang lebih kecil, kemudian digabungkan lagi sehingga tidak perlu melakukan banyak perbandingan.
- Cocok untuk sorting akses datanya lambat misalnya tape drive atau hard disk.
- Cocok untuk sorting data yang biasanya diakses secara sequentially (berurutan),
- misalnya linked list, tape drive, dan hard disk.

Kekurangan :

- Merge Sort membutuhkan lebih banyak ruang daripada jenis sorting lainnya
- Performa buruk untuk list dengan index sedikit dibandingkan algoritma sorting lainnya seperti bubble sort dan insertion sort
- Jika data sudah tersorting sejak awal, maka ia akan tetap melakukan sorting dari awal, maka ia akan tetap melakukan sorting dari awal
- Menggunakan memory yang lebih untuk melakukan split data