

BAB 2

SORTING (LANJUTAN)

2.1. TUJUAN PEMBELAJARAN

Tujuan dari pembelajaran pada pertemuan ini adalah mahasiswa diharapkan mampu:

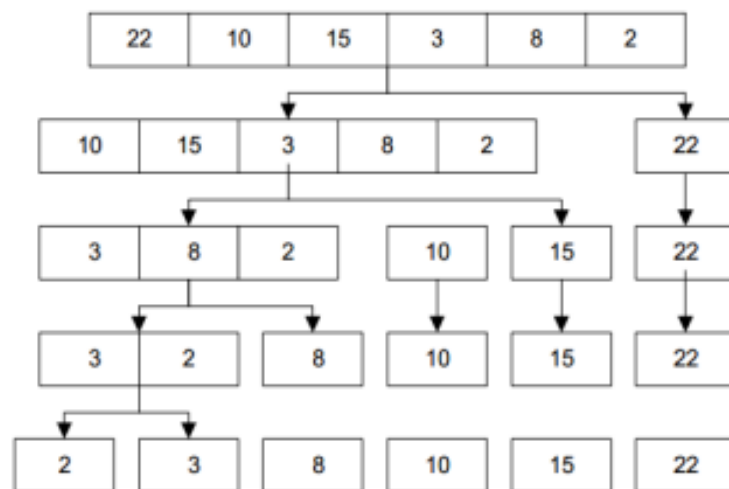
1. Memahami algoritma pengurutan *quick sort* dan *merge sort*.
2. Mampu mengimplementasikan algoritma pengurutan *quick sort* dan *merge sort* secara *ascending* dan *descending*.

2.2. DASAR TEORI

2.2.1. Algoritma Quick Sort

Algoritma QuickSort merupakan algoritma untuk mengurutkan data dengan pendekatan rekursif. Metode *quick sort* ini dikembangkan oleh C.A.R Hoare. Proses pengurutan dilakukan dengan memecah kumpulan data menjadi dua bagian berdasarkan nilai pivot yang dipilih. Pada prinsipnya nilai pivot yang dipilih ini akan ditempatkan pada posisinya di setiap akhir proses partisi.

Sebagai contoh, misalkan terdapat sebuah program untuk mengurutkan data A yang mempunyai N elemen. Program akan memilih sembarang elemen dari data tersebut, biasanya elemen pertama, misalnya X. kemudian semua elemen tersebut disusun dengan menempatkan X pada posisi J sedemikian rupa sehingga elemen ke 1 sampai ke J-1 mempunyai nilai lebih kecil dari X dan elemen J+1 sampai ke N mempunyai nilai lebih besar dari X. Sampai saat ini, program sudah mempunyai dua sub data (kiri dan kanan). Langkah berikutnya diulang untuk setiap sub data.



Gambar 2.1 Langkah – langkah *quick sort*

2.2.2. Algoritma Merge Sort

Merge sort merupakan algoritma pengurutan yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data dengan jumlah yang besar dan tidak memungkinkan untuk ditampung dalam memori komputer. Berdasarkan kegunaannya, algoritma merge ini disesuaikan untuk mesin drive tape. Penggunaannya dalam akses memori acak besar yang terkait telah menurun, karena banyak aplikasi algoritma merge yang mempunyai alternatif lebih cepat ketika kamu memiliki akses memori acak yang menjaga semua data mu. Hal ini disebabkan algoritma ini membutuhkan setidaknya ruang atau memori dua kali lebih besar karena dilakukan secara rekursif dan memakai dua tabel.

Seperti QuickSort, Merge Sort adalah algoritma *Divide and Conquer*. Ini membagi array input dalam dua bagian, memanggil dirinya sendiri untuk dua bagian dan kemudian menggabungkan dua bagian yang diurutkan. Berikut adalah cara kerja algoritma merge sort, langkah pertamanya adalah dengan membagi tabel menjadi dua tabel yang sama besar. Masing-masing tabel diurutkan secara rekursif, dan kemudian digabungkan kembali untuk membentuk tabel yang terurut. Implementasi dasar dari algoritma *merge sort* memakai tiga buah tabel, dua untuk menyimpan elemen dari tabel yang telah dibagi dua dan satu untuk menyimpan elemen yang telah terurut. Namun algoritma ini dapat juga dilakukan langsung pada dua tabel, sehingga menghemat ruang atau memori yang dibutuhkan.

Pada umumnya, Algoritma Merge memiliki satu set pointer $p_{0..n}$ yang menunjuk suatu posisi di dalam satu set daftar $L_{0..n}$. Pada awalnya terdapat item yang pertama yang ditunjuk pada setiap daftar. Algoritmanya sebagai berikut:

Selama $p_{0..n}$ masih menunjuk data yang di dalam sebagai pengganti, maka pada akhirnya:

1. Melakukan sesuatu dengan data item yang menunjuk daftar mereka masing-masing.
2. Menemukan pointers points untuk item dengan kunci yang paling rendah; membantu salah satu pointer untuk item yang berikutnya dalam daftar.

Contohnya dapat diilustrasikan dari penerapan atas sebuah larik sebagai data sumber yang akan diurutkan $\{3, 9, 4, 1, 5, 2\}$ adalah sebagai berikut:

- a. Pertama, larik tersebut dibagi menjadi dua bagian, $\{3, 9, 4\}$ dan $\{1, 5, 2\}$
- b. Kedua, larik kemudian diurutkan secara terpisah sehingga menjadi $\{3, 4, 9\}$ dan $\{1, 2, 5\}$

- c. Sebuah larik baru dibentuk yang sebagai penggabungan dari kedua larik tersebut {1}, sementara nilai-nilai dalam masing larik {3, 4, 9} dan {2, 5} (nilai 1 dalam elemen larik ke dua telah dipindahkan ke larik baru)
- d. Langkah terakhir adalah penggabungan dari masing-masing larik ke dalam larik baru yang dibuat sebelumnya.
 - 1) {1, 2} <-> {3, 4, 9} dan {5}
 - 2) {1, 2, 3} <-> {4, 9} dan {5}
 - 3) {1, 2, 3, 4} <-> {9} dan {5}
 - 4) {1, 2, 3, 4, 5} <-> {9} dan {null}
 - 5) {1, 2, 3, 4, 5, 9} <-> {null} dan {null}

2.3. PERCOBAAN

1. Buatlah workspace menggunakan Replit.
2. Buatlah project baru SORTING2 yang berisi file C++ source untuk metode *quick_sort* dan *merge_sort*
3. Cobalah untuk masing-masing percobaan di bawah.

Percobaan 1 : Implementasi *quick sort* secara *ascending*

```
#include<iostream>;
using namespace std;

void swap(int arr[] , int pos1, int pos2){
    int temp;
    temp = arr[pos1];
    arr[pos1] = arr[pos2];
    arr[pos2] = temp;
}

int partition(int arr[], int low, int high, int pivot){
    int i = low;
    int j = low;
    while( i <= high){
        if(arr[i] > pivot){
            i++;
        }
    }
}
```

```

        else{
            swap(arr,i,j);
            i++;
            j++;
        }
    }
    return j-1;
}

void quickSort(int arr[], int low, int high){
    if(low < high){
        int pivot = arr[high];
        int pos = partition(arr, low, high, pivot);

        quickSort(arr, low, pos-1);
        quickSort(arr, pos+1, high);
    }
}

int main()
{
    int n ;
    cout << "Tentukan panjang array : ";
    cin>>n;
    int arr[n];
    for( int i = 0 ; i < n; i++){
        cin>> arr[i];
    }
    quickSort(arr, 0 , n-1);
    cout<<"Berikut adalah array yang telah di sortir: ";
    for( int i = 0 ; i < n; i++){
        cout<< arr[i]<<"\t";
    }
}

```

```
}
```

OUTPUT

Tentukan panjang array : 5

3

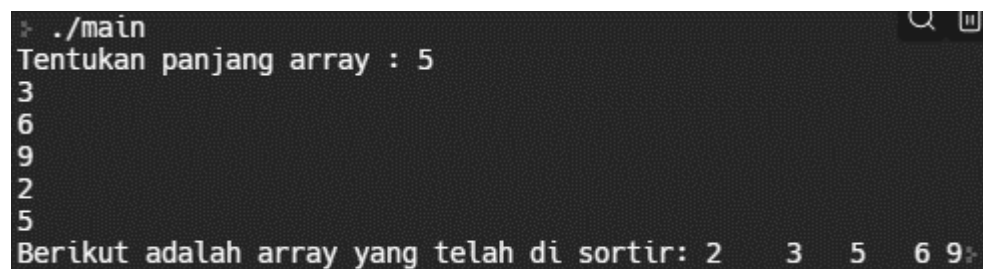
6

9

2

5

Berikut adalah array yang telah di sortir: 2 3 5 6 9



```
./main
Tentukan panjang array : 5
3
6
9
2
5
Berikut adalah array yang telah di sortir: 2 3 5 6 9
```

Percobaan 2 : Implementasi *merge sort*

```
#include <iostream>
using namespace std;

void merge(int arr[], int l, int m, int r)
{
    int x, y, z;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (x = 0; x < n1; x++)
        L[x] = arr[l + x];
    for (y = 0; y < n2; y++)
        R[y] = arr[m + 1 + y];
```

```

x = 0;
y = 0;
z = 1;
while (x < n1 && y < n2)
{
    if (L[x] <= R[y])
    {
        arr[z] = L[x];
        x++;
    }
    else
    {
        arr[z] = R[y];
        y++;
    }
    z++;
}

while (x < n1)
{
    arr[z] = L[x];
    x++;
    z++;
}

while (y < n2)
{
    arr[z] = R[y];
    y++;
    z++;
}
}

void mergeSort(int arr[], int l, int r)

```

```

{
    if (l < r)
    {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

void show(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << A[i] << " ";
}

int main()
{
    int size;
    cout << "\nMasukan Banyak Data : ";

    cin >> size;

    int arr[size];

    for (int i = 0; i < size; ++i)
    {
        cout << "\nMasukan Data array ke "<<i<<" :";
        cin >> arr[i];
    }
}

```

```
mergeSort(arr, 0, size);

cout << "Hasil\n";
show(arr, size);
return 0;
}
```

OUTPUT

Masukan Banyak Data : 5

Masukan Data array ke 0 :9

Masukan Data array ke 1 :8

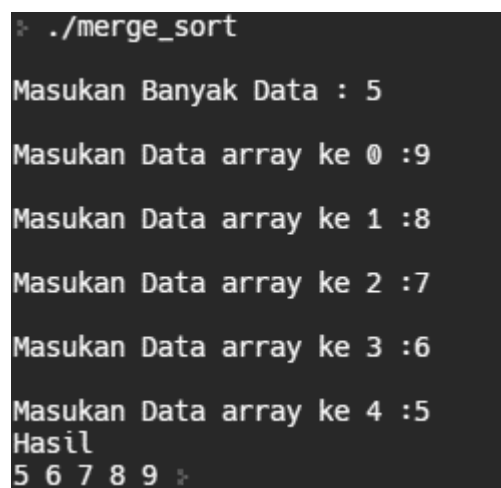
Masukan Data array ke 2 :7

Masukan Data array ke 3 :6

Masukan Data array ke 4 :5

Hasil

5 6 7 8 9

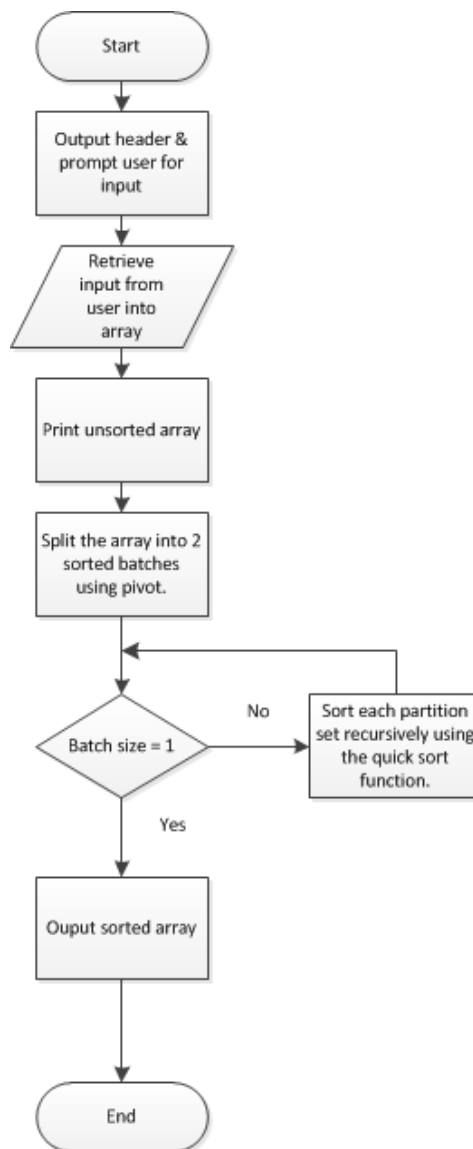


```
❖ ./merge_sort
Masukan Banyak Data : 5
Masukan Data array ke 0 :9
Masukan Data array ke 1 :8
Masukan Data array ke 2 :7
Masukan Data array ke 3 :6
Masukan Data array ke 4 :5
Hasil
5 6 7 8 9 ❖
```

2.4. TUGAS DAN LATIHAN

1. Buatlah *comment* atau penjelasan program pada contoh 1 sub bab **C.PERCobaan**

2. Buatlah program quick sort berdasarkan flowchart berikut!



Gambar 2.2 Soal no 2