

# Praktikum 12

---

## Algoritma Pengurutan (Quick Sort dan Merge Sort)

---

### A. TUJUAN PEMBELAJARAN

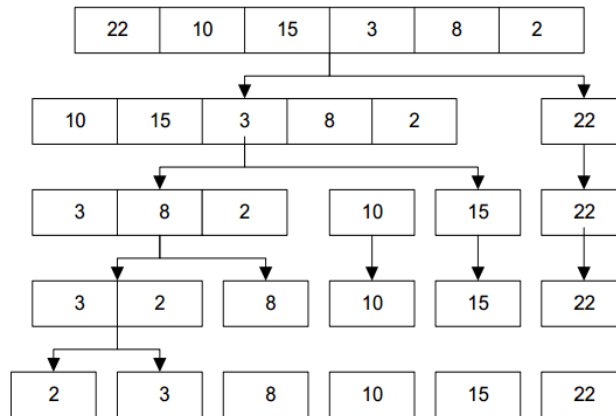
Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

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

### B. DASAR TEORI

#### B.1 Algoritma *Quick Sort*

Metode *quick sort* dikembangkan oleh C.A.R Hoare. Secara garis besar metode ini dijelaskan sebagai berikut. Misalnya kita ingin mengurutkan data A yang mempunyai N elemen. Kita pilih 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 kita sudah mempunyai dua sub data (kiri dan kanan). Langkah berikutnya diulang untuk setiap sub data.



**Gambar 11.1** Langkah-langkah Quick Sort

## B.2 Algoritma Merge Sort

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.

Algoritma *merge sort* 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 di bagi 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.

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

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

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

**CONTOH :**

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

- Larik tersebut dibagi menjadi dua bagian,  $\{3, 9, 4\}$  dan  $\{1, 5, 2\}$
- Kedua larik kemudian diurutkan secara terpisah sehingga menjadi  $\{3, 4, 9\}$  dan  $\{1, 2, 5\}$
- 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)
- langkah berikutnya adalah penggabungan dari masing-masing larik ke dalam larik baru yang dibuat sebelumnya.
  - a.  $\{1, 2\} \leftrightarrow \{3, 4, 9\}$  dan  $\{5\}$
  - b.  $\{1, 2, 3\} \leftrightarrow \{4, 9\}$  dan  $\{5\}$
  - c.  $\{1, 2, 3, 4\} \leftrightarrow \{9\}$  dan  $\{5\}$
  - d.  $\{1, 2, 3, 4, 5\} \leftrightarrow \{9\}$  dan  $\{null\}$
  - e.  $\{1, 2, 3, 4, 5, 9\} \leftrightarrow \{null\}$  dan  $\{null\}$

**C. TUGAS PENDAHULUAN**

Jawablah pertanyaan berikut ini :

1. Tuliskan algoritma pengurutan *quick sort* secara *ascending*.
2. Tuliskan algoritma pengurutan *merge sort* secara *ascending*.

**D. PERCOBAAN****Percobaan 1 : *Quick sort* secara *ascending*.**

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define MAX 20

int Partition (int arr[],int p,int r)
{
    int i, j;
    int pivot, temp;

    pivot = arr[p]; //pivot pada index 0
    i = p;
    j = r;
```

```

while (i <= j){
    while(pivot < arr[j])
        j--;
    while(pivot > arr[i])
        i++;
    if(i < j){
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        j--;
        i++;
    }
    else
        return j;
}
return j;
}

void QuickSort(int arr[],int p,int r)
{
    int q;
    if(p < r) {
        q = Partition(arr, p, r);
        QuickSort(arr, p, q);
        QuickSort(arr, q+1, r);
    }
}

void main()
{
    int data_awal[MAX], data_urut[MAX];
    int i;
    long k1, k2;

    printf("Sebelum pengurutan : \n");
    for(i=0; i<MAX; i++){
        srand(time(NULL) * (i+1));
        data_awal[i] = rand() % 100 + 1;
        printf("%d ", data_awal[i]);
    }
    printf("\nSetelah pengurutan : \n");
    for(i=0; i<MAX; i++)
        data_urut[i] = data_awal[i];

    time(&k1);
    QuickSort(data_urut,0,MAX-1);
    time(&k2);
    for(i=0; i<MAX; i++)
        printf("%d ", data_urut[i]);
    printf("\nWaktu = %ld\n", k2-k1);
}

```

### Percobaan 2 : Merge sort secara ascending

```

#include<stdio.h>
#include<stdlib.h>
#include<time.h>

```

```

#define MAX 20

void Merge(int arr[], int left, int median, int right)
{
    int temp[MAX];
    int kiril, kanan1, kiri2, kanan2, i, j;
    kiril = left;
    kanan1 = median;
    kiri2 = median+1;
    kanan2 = right;
    i = left;

    while ((kiril <= kanan1) && (kiri2 <= kanan2)) {
        if (arr[kiril] <= arr[kiri2]) {
            temp[i] = arr[kiril];
            kiril++;
        } else {
            temp[i] = arr[kiri2];
            kiri2++;
        }
        i++;
    }
    while(kiril <= kanan1){
        temp[i] = arr[kiril];
        kiril++;
        i++;
    }
    while(kiri2 <= kanan2){
        temp[i] = arr[kiri2];
        kiri2++;
        i++;
    }
    j = left;
    while(j <= right){
        arr[j] = temp[j];
        j++;
    }
}

void MergeSort(int arr[], int l, int r)
{
    int med;
    if(l < r){
        med = (l + r) / 2;
        MergeSort(arr, l, med);
        MergeSort(arr, med + 1, r);
        Merge(arr, l, med, r);
    }
}

void main()
{
    int data_awal[MAX], data_urut[MAX];
    int i;
    long k1, k2;

    printf("Sebelum pengurutan : \n");
    for(i=0; i<MAX; i++){
        srand(time(NULL) * (i+1));
        data_awal[i] = rand() % 100 + 1;
    }
}

```

```

        printf("%d ", data_awal[i]);
    }
    printf("\nSetelah pengurutan : \n");
    for(i=0; i<MAX; i++)
        data_urut[i] = data_awal[i];

    time(&k1);
    MergeSort(data_urut, 0, MAX-1);
    time(&k2);
    for(i=0; i<MAX; i++)
        printf("%d ", data_urut[i]);
    printf("\nWaktu = %ld\n", k2-k1);
}

```

### E. LATIHAN

1. Dari percobaan 1 tambahkan fungsi untuk melakukan pengurutan *quick sort* secara *descending*.
2. Dari percobaan 2 tambahkan fungsi untuk melakukan pengurutan *merge sort* dengan *flag* secara *descending*.
3. Buatlah struktur Mahasiswa dengan variable `nrp` dan `nama` yang memiliki tipe `String` dan kelas yang bertipe `int`. Buatlah fungsi pengurutan dengan *shell sort* dan *merge sort* berdasarkan `nrp`.

```

struct Mahasiswa {
    char nrp[10];
    char nama[20];
    int kelas;
};

```

4. Buatlah menu untuk memilih jenis algoritma pengurutan dan menu ascending atau descending seperti di bawah ini

```

ALGORITMA SORTING
1. Insertion
2. Selection
3. Bubble
4. Bubble dengan Flag
5. Shell
6. Quick
7. Merge
Pilihan : _1
1. Ascending
2. Descending
Pilihan : _1
Waktu : 4

```

Gunakan data acak dari array of Integer dengan ukuran 100.

5. Lakukan percobaan untuk melihat waktu running untuk setiap algoritma sorting dengan data acak yang sama seperti table di bawah ini dan gambarkan dalam bentuk grafik (boleh menggunakan Excel atau Matlab).

Jumlah data	Waktu Proses (milidetik)						
	Insertion	Selection	Bubble	Bubble Flag	Shell	Quick	Merge
50000							
100000							
150000							
200000							
250000							
300000							
350000							
400000							
450000							
500000							

**F. LAPORAN RESMI**

1. Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.
2. Tuliskan kesimpulan dari percobaan dan latihan yang telah anda lakukan.