Praktikum 9

Pengurutan (Sorting)

Quick Sort, Merge Sort

POKOK BAHASAN:

- ✓ Konsep pengurutan dengan *quick sort* dan *merge sort*
- ✓ Struktur data proses pengurutan
- ✓ Implementasi algoritma pengurutan *quick sort* dan *merge sort* dalam Bahasa C

TUJUAN BELAJAR:

Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

- ✓ Memahami karakteristik algoritma pengurutan *quick sort* dan *merge sort.*
- ✓ Mengimplementasikan algoritma *quick sort* dan *merge sort* dalam bentuk flowchart.
- ✓ Membuat diagram alir dan mengimplementasikan algoritma pada suatu permasalahan.

DASAR TEORI:

1. METODE QUICK SORT

Metode Quick sering disebut juga metode partisi (*partition exchange sort*). Metode ini mempunyai efektifitas yang tinggi dengan teknik menukarkan dua elemen dengan jarak yang cukup besar.

Metode pengurutan *quick sort* dapat diimplementasikan dalam bentuk non rekursif dan rekursif.

1.1 METODE QUICK SORT NON REKURSIF

Pada implementasi algoritma *quick sort* secara non rekursif memerlukan dua buah tumpukan (*stack*) yang digunakan yang digunakan untuk menyimpan batas-batas subbagian. Pada prosedur ini menggunakan tumpukan yang bertipe *record* (struktur) yang terdiri dari elemen kiri (untuk mencatat batas kiri) dan kanan (untuk mencatat batas kanan).

Ilustrasi dari langkah-langkah pengurutan dengan algoritma *quick sort* dapat dilihat pada tabel berikut :

Iterasi	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]
Awal	12	35	9	11	3	17	23	15	31	20
L=0;R=9	**12**	35	9	11	**3**	17	23	15	31	20
L=0;R=4	3	**35**	**9**	11	12	17	23	15	31	20
L=1;R=3	3	9	**35**	**11**	12	17	23	15	31	20
L=1;R=9	3	9	11	**35**	12	17	23	**15**	31	20
L=2;R=4	3	9	11	**15**	**12**	17	23	35	31	20
L=5;R=9	3	9	11	12	15	17	23	**35**	31	**20**
L=6;R=7	3	9	11	12	15	17	**23**	**20**	31	35
L=8;R=9	3	9	11	12	15	17	20	23	31	35
Akhir	3	9	11	12	15	17	20	23	31	35

Proses pengurutan metode *quick sort* non rekursif dapat dijelaskan sebagai berikut:

- L dan R menunjukkan data nilai tumpukan teratas pada tumpukan kiri dan kanan. tumpukan Mula-mula L=0 dan R=9 dan pivot adalah pada data ke-4 yaitu 3. Kita mencari data di sebelah kiri pivot yang lebih besar daripada 3, ternyata data ke-0 yaitu 12 lebih besar daripada 3. Untuk data di sebelah kanan pivot, ternyata tidak ada data yang lebih kecil daripada 3, yang berarti 3 adalah data terkecil, sehingga dilakukan pertukaran.
- Kemudian dibuat dua kumpulan data baru berdasarkan hasil ini. Kumpulan data pertama adalah data yang memiliki indeks 0 s/d 4-0=4 dan kumpulan data kedua adalah data yang memiliki indeks 0 + 1 = 1 s/d 9. Kumpulan data kedua ini belum bisa ditentukan sekarang karena masih tergantung dari hasil pengurutan kumpulan data pertama.

- Kembali ke kumpulan data pertama, dicari pivot kemudian menggunakan aturan yang serupa. Pivot pada kumpulan data ini adalah data ke-2 yaitu 9. Ternyata terdapat data yang lebih besar dari 9 di sebelah kiri yaitu 35 sehingga dilakukan pertukaran
- Langkah selanjutnya membuat dua kumpulan data lagi. Kumpulan pertama mempunyai indeks 0 sampai dengan 4-1=3. Pivot dari kumpulan data ini adalah 9. Perhatikan tidak ada data yang lebih besar dari 9 di sebelah kiri dan lebih kecil dari 9 di sebelah kanan. Kumpulan kedua dari indeks 0 s/d 4 adalah data ke 2 dan ke 4-1=3 yaitu 35 dan 11. Ternyata 35 lebih besar dari 11 sehingga kedua data ini ditukar.
- Kembali ke kumpulan data kedua yang memiliki indeks 1 s/d 9. Kumpulan ini dibagi menjadi dua yaitu kumpulan data berindeks 1 s/d 4 dan kumpulan data berindeks 5 s/d 9. Pivot dari data ini adalah data ke 5 yaitu 17. Data yang lebih besar dari 17 di sebelah kiri adalah 35 dan data yang lebih kecil dari 17 di sebelah kanan adalah 15, jadi kedua data ini ditukar.
- Dari hasil penukaran ini dilakukan pembagian menjadi 2 kumpulan data. Kumpulan pertama yaitu dari indeks 2 s/d 4, pivot pada data ke 3 dan terjadi penukaran data 15 dan 12. Kumpulan kedua yaitu dari indeks 4 s/d 5 tidak terjadi pertukaran data
- Kembali ke kumpulan kedua dari indeks 5 s/d 9. Pivot dari kumpulan ini adalah 35. Dicari data yang lebih besar dari 35 di sebelah kiri dan data yang lebih kecil dari 35 di sebelah kanan. Ternyata terjadi pertukaran data 35 dan 20.
- Dari hasil pertukaran ini dilakukan pembagian kumpulan data yaitu data yang mempunyai indeks 6 s/d 7 dan 8 s/d 9. Kumpulan data pertama terjadi pertukaran data 23 dan 20 sedangkan kumpulan data kedua tidak terjadi pertukaran data

1.2 METODE QUICK SORT REKURSIF

Implementasi algoritma pengurutan *quick sort* juga dapat menggunakan proses rekursif pada proses pengurutan bagian kiri dan kanan karena mempunyai proses yang sama. Dengan menggunakan proses rekursif, tidak diperlukan tumpukan untuk menyimpan batas kiri dan kanan.

2. METODE MERGE SORT

Metode penggabungan (*merge sort*) melakukan pengurutan dengan cara membagi data menjadi 2 kumpulan data dan masing-masing kumpulan diurutkan, kemudian setelah terurut dilakukan penggabungan dua kumpulan data tersebut dalam keadaan terurut.

Ilustrasi dari langkah-langkah pengurutan dengan algoritma *merge sort* dapat dilihat pada tabel berikut :

Iterasi	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]
Awal	**12**	**35**	9	11	3	17	23	15	31	20
L=0;M=0;R=1	**12**	35	**9**	11	3	17	23	15	31	20
L=0; M=1;R=2	9	12	35	**11**	**3**	17	23	15	31	20
L=3;M=3;R=4	**9**	12	35	3	**11**	17	23	15	31	20
L=0;M=2;R=4	3	9	11	12	35	**17**	**23**	15	31	20
L=5;M=5;R=6	3	9	11	12	35	**23**	17	**15**	31	20
L=5;M=6;R=7	3	9	11	12	35	15	17	23	**31**	**20**
L=8;M=8;R=9	3	9	11	12	35	**15**	17	23	20	**31**
L=5;M=7;R=9	**3**	9	11	12	35	15	17	20	23	**31**
L=0;M=4;R=9	3	9	11	12	15	17	20	23	31	35
Akhir	3	9	11	12	15	17	20	23	31	35

- Data yang akan diurutkan dibagi ke dalam dua kumpulan data yaitu kumpulan pertama terdiri dari `Data[0]` sampai dengan `Data[4]` dan kumpulan kedua terdiri dari `Data[5]` sampai dengan `Data[9].`
- Ditentukan batas kiri `L` mulai 0 dari kumpulan pertama dan dicari data terkecil setelah data ke 0 yaitu 9 pada batas kanan `M`=2 kemudian diurutkan.
- Nilai `L=M` dan `M` ditentukan dengan posisi data yang lebih kecil dari data pada posisi `L` kemudian diurutkan.
- Proses dilakukan sampai kumpulan pertama terurut.
- Hal yang sama dilakukan terhadap kumpulan kedua sampai data pada kumpulan kedua terurut
- Langkah terakhir mengurutkan keseluruhan data dari kumpulan pertama dan kedua yang sudah terurut.

TUGAS PENDAHULUAN:

1. Buatlah flowchart untuk operasi pengurutan bilangan bulat dengan algoritma *quick sort* non rekursif, *quick sort* rekursif dan *merge sort*.
2. Buatlah deklarasi data pegawai dan flowchart fungsi untuk mengurutkan data dengan metode *quick sort* non rekursif, *quick sort* rekursif dan *merge sort* sebagai berikut :
   - Data bertipe rekaman bernama Pegawai yang mempunyai 4 data yaitu :
     - `NIP`, bertipe bulat
     - `Nama`, bertipe string
     - `Alamat`, bertipe string
     - `Golongan`, bertipe char
   - Data diurutkan dengan urut naik berdasarkan NIP

PERCOBAAN:

1. Buatlah workspace menggunakan Visual C++.
2. Buatlah file *C source* untuk metode pengurutan *quick sort* non rekursif, *quick sort* rekursif dan *merge sort* pada project SORTING yang telah dibuat pada praktikum 7.
3. Cobalah untuk masing-masing percobaan di bawah dengan menambahkan menu pilihan metode pengurutan pada program utama.

Percobaan 1 : Implementasi pengurutan dengan metode *quick sort* non rekursif

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

#define MAX 10
#define MaxStack 10

int Data[MAX]; // = {12,35,9,11,3,17,23,15,31,20};

// Prosedur menukar data
void Tukar (int *a, int *b)
{
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```

```
// Prosedur pengurutan metode Quick Sort
void QuickSortNonRekursif()
{
	struct tump {
		int Kiri;
		int Kanan;
	}Tumpukan[MaxStack];

	int i, j, L, R, x, ujung = 1;
	Tumpukan[1].Kiri = 0;
	Tumpukan[1].Kanan = MAX-1;

	while (ujung!=0){
		L = Tumpukan[ujung].Kiri;
		R = Tumpukan[ujung].Kanan;
		ujung--;
		while(R > L){
			i = L;
			j = R;
			x = Data[(L+R)/2];
			while(i <= j){
				while(Data[i] < x)
					i++;
				while(x < Data[j])
					j--;
				if(i <= j){
					Tukar(&Data[i], &Data[j]);
					i++;
					j--;
				}
			}
			if(L < i){
				ujung++;
				Tumpukan[ujung].Kiri = i;
				Tumpukan[ujung].Kanan = R;
			}
			R = j;
		}
	}
}
```

```
void main()
{
   int i;
   srand(0);

// Membangkitkan bilangan acak
   printf("DATA SEBELUM TERURUT");
   for(i=0; i<MAX; i++)
   {
         Data[i] = (int) rand()/1000+1;
         printf("\nData ke %d : %d ", i, Data[i]);
   }

   QuickSortNonRekursif();

   // Data setelah terurut
   printf("\nDATA SETELAH TERURUT");
   for(i=0; i<MAX; i++)
   {
         printf("\nData ke %d : %d ", i, Data[i]);
   }
}
```

Percobaan 2 : Implementasi pengurutan dengan metode *quick sort* rekursif

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

#define MAX 10

int Data[MAX];

// Prosedur menukar data

void Tukar (int *a, int *b)
{
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```

```
// Prosedur pengurutan metode Quick Sort

void QuickSortRekursif(int L, int R)
{
      int i, j, x;
      x = Data[(L+R)/2];
      i = L;
      j = R;
      while (i <= j){
            while(Data[i] < x)
                  i++;
            while(Data[j] > x)
                  j--;
            if(i <= j){
                  Tukar(&Data[i], &Data[j]);
                  i++;
                  j--;
            }
      }
      if(L < j)
            QuickSortRekursif(L, j);
      if(i < R)
            QuickSortRekursif(i, R);
}

void main()
{
   int i;
   srand(0);

// Membangkitkan bilangan acak
   printf("DATA SEBELUM TERURUT");
   for(i=0; i<MAX; i++)
   {
         Data[i] = (int) rand()/1000+1;
         printf("\nData ke %d : %d ", i, Data[i]);
   }

   QuickSortRekursif(0, MAX-1);

   // Data setelah terurut
   printf("\nDATA SETELAH TERURUT");
   for(i=0; i<MAX; i++)
   {
         printf("\nData ke %d : %d ", i, Data[i]);
   }
}
```

Percobaan 3 : Implementasi pengurutan dengan metode *merge sort*

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

#define MAX 10

int Data[MAX];
int temp[MAX];

// Prosedur merge sort
void merge(int Data[], int temp[], int kiri, int tengah, int kanan)
{
   int i, left_end, num_elements, tmp_pos;

   left_end = tengah - 1;
   tmp_pos = kiri;
   num_elements = kanan - kiri + 1;

   while ((kiri <= left_end) && (tengah <= kanan))
   {
      if (Data[kiri] <= Data[tengah])
        {
         temp[tmp_pos] = Data[kiri];
         tmp_pos = tmp_pos + 1;
         kiri = kiri +1;
        }
      else
        {
         temp[tmp_pos] = Data[tengah];
         tmp_pos = tmp_pos + 1;
         tengah = tengah + 1;
        }
  }

  while (kiri <= left_end)
  {
      temp[tmp_pos] = Data[kiri];
      kiri = kiri + 1;
      tmp_pos = tmp_pos + 1;
  }
  while (tengah <= kanan)
  {
      temp[tmp_pos] = Data[tengah];
      tengah = tengah + 1;
      tmp_pos = tmp_pos + 1;
  }

  for (i=0; i <= num_elements; i++)
  {
      Data[kanan] = temp[kanan];
        kanan = kanan - 1;
  }
}
```

```
// Prosedur membuat kumpulan data
void m_sort(int Data[], int temp[], int kiri, int kanan)
{
  int tengah;

  if (kanan > kiri)
  {
      tengah = (kanan + kiri) / 2;

        m_sort(Data, temp, kiri, tengah);
      m_sort(Data, temp, tengah+1, kanan);

      merge(Data, temp, kiri, tengah+1, kanan);
  }
}

void mergeSort(int Data[], int temp[], int array_size)
{
    m_sort(Data, temp, 0, array_size - 1);
}

int main()
{
   int i;

   //pembangkit bilangan random
   srand(0);

   //membangkitkan bilangan integer random
   printf("\nDATA SEBELUM TERURUT : ");
   for (i = 0; i < MAX; i++)
   {
       Data[i] = rand()/1000+1;
         printf("%d  ", Data[i]);
   }

   mergeSort(Data, temp, MAX);

   printf("\nDATA SETELAH TERURUT : ");
   for (i = 0; i < MAX; i++)
      printf("%d  ", Data[i]);
   printf("\n");
}
```

LATIHAN:

1. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan *quick sort* dan *merge sort*.
2. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma *quick sort* dan *merge sort*.
3. Implementasikan pengurutan data Pegawai pada tugas pendahuluan dengan ketentuan :
    a. Metode pengurutan dapat dipilih.
    b. Pengurutan dapat dipilih secara urut naik atau turun.
    c. Pengurutan dapat dipilih berdasarkan NIP dan NAMA.
    d. Gunakan struktur data array.
4. Berikan kesimpulan dari percobaan dan latihan yang telah Anda lakukan.