MINGGU 12
Sorting

DESKRIPSI TEMA

Sorting adalah kegiatan mengurutkan data dengan format yang diinginkan. Sedangkan algoritma *sorting* adalah aturan atau langkah untuk mengurutkan data dengan urutan tertentu. *Sorting* menjadi penting dalam pencarian data dan memudahkan untuk pembacaan data. Jenis-jenis *sorting* yang dipelajari pada pertemuan minggu ini adalah: Bubble Sort, Selection Sort, Insertion Sort, Radix Sort, Merge Sort, Shell Sort, Heap Sort, dan Quick Sort

Bubble Sort adalah salah satu *sorting algorithm* yang berulang kali berjalan melalui *list* yang akan diurutkan, membandingkan setiap pasangan item yang berdekatan, dan menukar mereka jika mereka berada di urutan yang salah. Nilai-nilai yang lebih kecil secara bertahap berjalan ke bagian atas *array* seperti gelembung yang naik dalam air, sedangkan nilai-nilai yang lebih besar tenggelam ke bagian bawah *array*.

Selection Sort adalah salah satu *sorting algorithm* yang bersifat *in-place* (tidak memerlukan memori tambahan) dan menggunakan konsep perbandingan antar dua elemen. Pada algoritma ini, *list* dibagi menjadi dua bagian *sorted-part* (kiri) dan *unsorted part* (kanan). (Pada kondisi awal, *sorted-part* masih kosong dan *unsorted-part* mencakup keseluruhan *list*.) Langkah yang dilakukan dalam algoritma ini adalah memilih elemen paling kecil pada *unsorted-part* (dengan melakukan perbandingan antar elemen), lalu meletakkannya pada bagian paling kanan dari *sorted-part*. Hal ini dilakukan hingga *unsorted-part* kosong. Algoritma ini kurang tepat digunakan untuk data dalam jumlah banyak karena kompleksitasnya tergantung jumlah elemen dalam *list*.

Insertion Sort adalah salah satu *sorting algorithm* yang bersifat *in-place* (tidak memerlukan memori tambahan) dan menggunakan konsep perbandingan antar dua elemen. Pada algoritma ini, *list* dibagi menjadi dua bagian *sorted-part* (kiri) dan *unsorted-part* (kanan). (Pada kondisi awal, *sorted-part* berisi satu elemen paling kiri dan *unsorted-part* berisi sisa elemen dalam list.) Langkah yang dilakukan dalam algoritma ini adalah menunjuk elemen paling kiri pada unsorted-part. Elemen ini akan dibandingkan dengan elemen-elemen pada *sorted-part* secara berurut dan dimasukkan (‘*insert’ed*) pada posisi yang benar. Hal ini dilakukan hingga *unsorted-part* kosong. Algoritma ini kurang tepat digunakan untuk data dalam jumlah banyak karena kompleksitasnya tergantung jumlah elemen dalam *list*.

Radix Sort adalah salah satu *sorting algorithm* yang bersifat *not-in-place* (memerlukan memori tambahan) dan menggunakan konsep perbandingan *least significant digits*. Pada algoritma ini, kita perlu menyediakan *array of linked list* sebagai penampung sementara (*bucket*).

Langkah yang dilakukan dalam algoritma ini adalah memasukkan tiap elemen dalam *list* - elemen ke-i sebagai digit ke-n nya ke dalam array[i], di mana n merupakan penentu yang dimulai dari *least significant digit*. Iterasi dilakukan sebanyak x kali, di mana x merupakan jumlah digit paling besar sebuah elemen dalam *list*.

Quick Sort adalah algoritma pengurutan yang menggunakan metode *divide and conquer*. Algoritma ini sering di implementasikan untuk data dengan jumlah yang banyak. Konsep utama algoritma ini adalah menentukan sebuah titik (pivot) kemudian membagi elemen dalam *list*, menjadi dua partisi yaitu: partisi yang

lebih kecil dari pivot dan partisi yang lebih besar dari pivot. Kemudian tiap partisi akan di *quick sort* (rekursif) hingga elemen sudah terurut.

Merge Sort adalah algoritma pengurutan yang menggunakan metode *divide and conquer*. Algoritma ini membagi sebuah *array* menjadi 2 bagian, dan tiap bagian tersebut akan terus dibagi 2 hingga akhirnya hanya terdiri dari 1 elemen saja.

Shell Sort merupakan variasi dari Insertion Sort. Dalam Insertion Sort, kita hanya memindahkan elemen ke 1 posisi ke depan, ketika sebuah elemen harus dipindahkan jauh ke depan, maka akan ada banyak pemindahan yang dilakukan. Sedangkan Shell Sort, dapat menukarkan suatu elemen dengan elemen lainnya yang jauh.

Sebelum mempelajari tentang heap sort, ada beberapa terminologi dalam *heap* yang perlu dimengerti terlebih dahulu, yaitu Heap Tree, Heapify, dan Heap Sort itu sendiri.

Heap Tree merupakan *complete binary tree* yang dapat dikategorikan menjadi 2 berdasarkan pengurutan nilainya.

- Max Heap Tree, yaitu *heap tree* yang nilai *parent*-nya selalu lebih besar dari nilai *child*-nya.
- Min Heap Tree, yaitu *heap tree* yang nilai *parent*-nya selalu lebih kecil dari nilai *child*-nya.

Heapify secara sederhana merupakan proses untuk menciptakan struktur *heap* yang benar. Heap tree dapat diimplementasikan untuk melakukan proses pengurutan data, proses ini biasa disebut dengan heap sort. Dalam proses heap sort dibutuhkan struktur *heap* yang benar. Proses inilah yang disebut dengan *heapify*.

Heap sort merupakan sebuah proses pengurutan data dengan memanfaatkan struktur *heap*.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

(Capaian pembelajaran pada minggu ini)

PENUNJANG PRAKTIKUM

1. CodeBlocks
2. Dev-C++ (alternatif)

LANGKAH-LANGKAH PRAKTIKUM

A. Tutorial 1.1 – Bubble Sort
   - Buatlah sebuah *file* dengan nama W12_NIM_ BubbleSort.c
   - Salinlah potongan *code* berikut.
   - Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

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

void bubbleSort(int *bil, int n){
  int i, j;

  for(i=1; i<n; i++){
    for(j=n-1; j>=1; j--){
      if(bil[j] < bil[j-1]){
        swap(&bil[j], &bil[j-1]);
      }
    }
  }
}

int main(){
  int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  bubbleSort(bil, n);

  printf("Hasil bubble sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

B. Tutorial 2.1 – Selection Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ SelectionSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

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

void selectionSort(int *bil, int n){
  int i, j, temp;
  for(i=0; i<n; i++){
    // Menetapkan elemen di indeks ke-i sebagai nilai minimum
    temp = i;

    // Cek nilai minimum terhadap elemen selanjutnya
    for(j=i+1; j<n; j++){
      if(bil[j] < bil[temp]){
        temp = j;
      }
    }

    // Tukar nilai
    if(temp != i){
      swap(&bil[temp], &bil[i]);
    }
  }
}
```

```c
int main(){
  int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  selectionSort(bil, n);

  printf("Hasil selection sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

C. Tutorial 3.1 – Insertion Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ InsertionSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

void insertionSort(int *bil, int n){
  int i, j, temp;
  for(i=0; i<n; i++){
    // Menetapkan elemen di indeks ke-i sebagai nilai minimum
    temp = bil[i];

    // Cek nilai minimum terhadap elemen selanjutnya
    for(j=i-1; j>=0 && bil[j] > temp; j--){
      bil[j+1] = bil[j];
    }

    bil[j+1] = temp;
  }
}

int main(){
int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  insertionSort(bil, n);

  printf("Hasil insertion sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

D. Tutorial 4.1 – Radix Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ RadixSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

typedef struct LinkedList{
  int value;
  struct LinkedList *next;
} LinkedList;

// Fungsi untuk membuat node baru dan
// insert ke belakang LinkedList
void insertNode(int val, LinkedList **head){
  // **head merupakan bucket[i]

  // Buat node baru
  LinkedList *newNode = (LinkedList*) malloc(sizeof(LinkedList));
  newNode->value = val;
  newNode->next = NULL;

  // Jika bucket[i] kosong
  if (*head == NULL){
    *head = newNode;
  } else { // Kalau bucket[i] udah ada isi
    LinkedList *temp = *head;
    while(temp->next != NULL){
      temp = temp->next;
    }
    temp->next = newNode;
  }
}
```

```c
int main(){
  int i, j, n, data[100], maks;

  // *bucket sebagai *head pada Radix Sort
  // Karena menginginkan sort integet 0-9, jadi terdapat 10 head
  // a.k.a. array of Linked List
  LinkedList *bucket[10];
  for(i=0; i<10; i++){
    bucket[i] = NULL;
  }

  // Input data
  printf("Masukkan jumlah data yang akan di sort: ");
  scanf("%d", &n);
  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &data[i]);
    if (i==0 || maks < data[i]) {
      maks = data[i];
    }
  }

  // Menentukan jumlah iterasi
  int totalIterasi = 0;
  while(maks > 0){
    maks /= 10;
    totalIterasi++;
  }
  printf("\nTotal iterasi yang akan dilakukan adalah: %d\n", totalIterasi);
```

```
int pembagi = 1; // Iterasi sebanyak totalIterasi
for(i=0; i<totalIterasi; i++){
  // Masukkan data dari array ke bucket yang sesuai
  for(j=0; j<n; j++){
    int digit = (data[j] / pembagi) % 10;
    insertNode(data[j], &bucket[digit]);
  }
  pembagi *= 10;

  // Output isi sementara bucket
  printf("\n====\nIterasi %d\n", i+1);
  for(j=0; j<10; j++){
    printf("Bucket[%d]: ", j);
    LinkedList *temp = bucket[j];
    while(temp != NULL){
      printf("%d ", temp->value);
      temp = temp->next;
    }
    printf("\n");
  }

  // Pindahkan kembali data dari bucket ke dalam array
  int idx = 0;
  for(j=0; j<10; j++){
    LinkedList *temp = bucket[j];
    while(temp != NULL){
      data[idx] = temp->value;
      idx++;
      temp = temp->next;
    }
  }

  // Output isi sementara array
  printf("Isi sementara array\n");
  for(j=0; j<n; j++){
    printf("%d ", data[j]);
  }
  printf("\n");

  // Kosongkan bucket
  for(j=0; j<10; j++){
    LinkedList *trash, *temp = bucket[i];
    while(temp != NULL){
      trash = temp;
      temp = temp->next;
      free(trash);
    }
    bucket[j] = NULL;
  }
}
```

```
111     // Output hasil sorting
112     printf("\nHasil setelah di sort\n");
113     for(j=0; j<n; j++){
114       printf("%d ", data[j]);
115     }
116
117     return 0;
118   }
```

E. Tutorial 5.1 – Quick Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ QuickSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

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

/*
  PARTITIONING
  Elemen terakhir dalam list dijadikan pivot
  Elemen dengan nilai lebih kecil dari pivot di swap ke bagian kiri pivot
  Elemen dengan nilai lebih besar dari pivot di swap ke bagian kanan pivot
*/
int partition(int *bil, int l, int r){
  // Menjadikan elemen terakhir list sebagai pivot
  int pivot = bil[r];
  // Tembok yang menjadi pemisah partisi kiri dan kanan
  // Pertama diletakkan di kiri elemen pertama
  int i = l - 1;

  // Looping dari elemen paling kiri hingga elemen sebelum pivot
  for(int j=l; j<=r-1; j++){
    // Jika elemen lebih kecil sama dengan pivot
    if (bil[j] <= pivot){
      // Geser tembok
      i++;
      // Sekarang bil[i] = elemen paling kanan di partisi kiri
      // bil[i] belum tenti lebih kecil dari pivot

      // Tukar bil[i] dengan bil[j] yang sudah dicek
      swap(&bil[i], &bil[j]);
    }
  }

  // Tukar elemen paling kiri partisi kanan dengan pivot
  swap(&bil[i+1], &bil[r]);

  // Mengembalikan index dari pivot yang sudah benar
  return (i+1);
}
```

```c
void quickSort(int *bil, int l, int r){
  if(l<r){
    // pi adalah partitioning index
    int pi = partition(bil, l, r);
    // Setelah fungsi dijalankan
    // bil[pi] (sebelumnya pivot) sudah berada pada posisi yang benar

    // Secara terpisah dan rekursif
    // Laukan quicksort untuk tiap partisi kiri dan kanan
    quickSort(bil, l, pi-1);
    quickSort(bil, pi+1, r);
  }
}

int main(){
  int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  quickSort(bil, 0, n-1);

  printf("Hasil quick sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

F. Tutorial 6.1 – Merge Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ MergeSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

void merge(int *bil, int l, int m, int r);

void mergeSort(int *bil, int l, int r){
  if (l < r){
    int m = (l+r)/2;

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

    // Proses ketika sudah menelusuri bagian kiri dan kanan
    merge(bil, l, m, r);
  }
}

int main(){
  int *bil, i, n;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  mergeSort(bil, 0, n-1);

  printf("Hasil merge sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

```
40  void merge(int *bil, int l, int m, int r){
41    int i, j, k;
42
43    // Banyak data di sisi kiri
44    int n1 = m - l + 1;
45
46    // Banyak data di sisi kanan
47    int n2 = r - m;
48
49    // Membuat array sementara untuk bilangan di sisi kiri
50    int L[n1];
51
52    // Membuat array sementara untuk bilangan di sisi kanan
53    int R[n2];
54
55    // Menyalin bilangan di sisi kiri ke array L[]
56    for(i=0; i<n1; i++)
57      L[i] = bil[l + i];
58
59    // Menyalin bilangan di sisi kanan ke array R[]
60    for(j=0; j<n2; j++)
61      R[j] = bil[m + 1 + j];
62
63    // Proses menggabungkan kembali bilangan di sisi kiri dan kanan
64    // menjadi bil[l..r]
65    i = 0; // Iterator untuk bilangan di sisi kiri
66    j = 0; // Iterator untuk bilangan di sisi kanan
67
68    // Iterator untuk menggabungkan kembali
69    // bilangan yang terpisah di sisi kiri dan kanan
70    k = l;
71
72    while(i<n1 && j<n2){
73      if(L[i] <= R[j]){
74        bil[k] = L[i];
75        i++;
76      } else {
77        bil[k] = R[j];
78        j++;
79      }
80      k++;
81    }
82
83    // Menyalin jika ada elemen yang tersisa dari L[]
84    while(i < n1){
85      bil[k] = L[i];
86      i++;
87      k++;
88    }
89
90    // Menyalin jika ada elemen yang tersisa dari R[]
91    while(j < n2){
92      bil[k] = R[j];
93      j++;
94      k++;
95    }
96  }
```

G. Tutorial 1.1 – Shell Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ ShellSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

void shellSort(int *bil, int n){
  int i, j, gap;

  // Mulai dari gap besar (n/2), lalu kurangi gapnya (gap /= 2)
  // NOTE: Tidak harus menggunakan angka 2 sebagai pembagi
  // Bisa gunakan angka yang lebih besar untuk perpindahan yang lebih jauh
  for (gap=n/2; gap>0; gap/=2){
    // Lakukan insertion sort untuk gap size ini.
    for(i=gap; i<n; i++){
      // Simpan bil[i] ke temp
      int temp = bil[i];

      // Pindahkan nilai yang besar ke tempat yang seharusnya
      for(j=i; j>=gap && bil[j - gap] > temp; j -= gap){
        bil[j] = bil[j - gap];
      }

      // Taruh temp ke lokasi yang benar
      bil[j] = temp;
    }
  }
}

int main(){
  int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  shellSort(bil, n);

  printf("Hasil shell sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

H. Tutorial 1.1 – Heap Sort

- Buatlah sebuah *file* dengan nama W12_NIM_ HeapSort.c
- Salinlah potongan *code* berikut.
- Perhatikan *indentation* dan *comment* dalam *code* untuk membantu menjelaskan

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

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

int heapify(int *bil, int n, int i){
  // int i akan mejadi indeks root sementara

  int maks = i;    // Mengambil indeks root sebagai nilai maks
  int l = 2*i + 1; // Rumus untuk mengambil indeks array akan kiri
  int r = 2*i + 2; // Rumus untuk mengambil indeks array akan kanan

  // Proses pengecekan, apakah nilai dari anak kiri > maks ?
  if (l<n && bil[l] > bil[maks]){
    maks = l;
  }

  // Proses pengecekan, apakah nilai dari anak kanan > maks ?
  if (r<n && bil[r] > bil[maks]){
    maks = r;
  }

  // Proses pengecekan, apakah maks masih sama dengan root
  if (maks != i){
    swap(&bil[i], &bil[maks]);

    // Melakukan heapify kembali terhadap sub-tree yang terkena dampak swap
    heapify(bil, n, maks);
  }
}
```

```c
int heapSort(int *bil, int n){
  int i;

  // Melakukan heapify dimulai dari parent yang berada pada indeks terbesar
  for(i=n/2 - 1; i>=0; i--){
    heapify(bil, n, i);
  }

  // Menukarkan elemen pada indeks[0] dengan indeks[(n--) - 1]
  // Satu persatu menghilangkan elemen dari heap
  // sehingga tidak lagi terlibat dalam proses heapify
  for(i=n-1; i>=0; i--){
    // Menukarkan elemen pada indeks[0] ke index [(n--) - 1]
    swap(&bil[0], &bil[i]);

    // Melakukan proses heapify
    heapify(bil, i, 0);
  }
}

int main(){
  int i, n, *bil;

  printf("Banyak bilangan: "); scanf("%d", &n);
  bil = malloc(sizeof(int) * n);

  for(i=0; i<n; i++){
    printf("Input bulangan ke-%d: ", i+1); scanf("%d", &bil[i]);
  }

  heapSort(bil, n);

  printf("Hasil heap sort:\n");
  for(i=0; i<n; i++){
    printf("%d ", bil[i]);
  }

  free(bil);

  return 0;
}
```

I. Tugas
   a) Buatlah versi terbalik dari masing-masing sort (buat descending jika contoh modul adalah ascending dan begitu juga sebaliknya)
   b) Gabungkan semua sorting (setelah dibuat versi terbaliknya) ke dalam 1 program dan beri nama W12_NIM_Tugas.c

REFERENSI