Priority Queue for Food Serving Based on the Estimated Time Using Minimum Heap Tree

LAPORAN PROYEK AKHIR

MATA KULIAH COMP6362004 – DATA STRUCTURES

KELAS BC20

Oleh:

2502012546 – Keyla Azzahra

2502020390 – Theodorus Austin Valenzio

2501966143 – Vanessa Danuwijaya

Semester Genap 2021/2022

MALANG

LEMBAR PERSETUJUAN PROYEK AKHIR

Priority Queue for Food Serving Based on the Estimated Time Using Minimum Heap Tree

MATA KULIAH COMP6362004 – DATA STRUCTURES

KELAS BC20

Semester Genap 2021/2022

Laporan akhir proyek ini adalah benar karya kami:

Keyla Azzahra	Theodorus Austin Valenzio	Vanessa Danuwijaya
2502012546	2502020390	2501966143

Malang, Senin 20 Juni 2022

(Elizabeth Paskahlia Gunawan, S. Kom., M.Cs.)

D5707

DAFTAR ISI

BAB 1

LATAR BELAKANG

1.1 Team's Planning

Program yang dibuat adalah sebuah *priority queue* yang bertujuan untuk mengurutkan sekumpulan data. Program ini terdiri atas beberapa menu, seperti untuk menambah data, menghapus data, menampilkan data, serta mengurutkan data berdasarkan syarat tertentu. Atribut yang digunakan dalam program ini adalah pesanan, nomor meja, dan waktu perkiraan.

1.2 Reason of Choosing the Topic

Topik ini dipilih karena *heap tree* dibutuhkan untuk menghemat waktu serta mempermudah untuk mengakses data terkecil/terbesar karena data yang ada dalam formasi yang diurutkan. Pada topik ini, jenis *heap tree* yang digunakan adalah *minimum heap tree*, yang berarti *tree* akan menyimpan data dari yang terkecil sebagai *root*. *Min-heap* mempunyai bisa diimplementasikan ke dalam berbagai kasus, contohnya adalah *priority queue*, dimana *binary sort* dalam *min-heap* dikombinasikan dengan konsep queue.

Kami memilih tema prioritas waktu di restoran untuk menentukan urutan pesanan yang dibuat terlebih dahulu untuk mendapatkan efektivitas waktu yang seoptimal mungkin. Dalam hal ini, meja yang memiliki estimasi waktu pembuatan yang tercepat dilayani terlebih dahulu dibandingkan dengan waktu kedatangan *customers*, dengan tujuan untuk menurunkan waktu tunggu pelanggan. Ketika pesanan untuk meja *root* atau antrian pertama sudah selesai, maka *root* akan dihapus dan digantikan dengan node terkecil selanjutnya.

1.3 Overview of Program Definition

Program yang dibuat memiliki beberapa fitur umum, seperti *add data, delete, dan display data*. Beberapa fitur lain juga ditambahkan ke dalam program, antara lain *display table* dan *display sorted table* untuk menampilkan data secara lebih terperinci. Selain itu, program juga bisa menampilkan sebuah visualisasi dari *heap tree* yang akan menampilkan data-data yang ada dalam bentuk *tree.* Node *root* dan anak-anaknya akan ditampilkan pada tampilan awal aplikasi untuk mempermudah *user* melihat estimasi waktu yang dibutuhkan.

BAB 2

TINJAUAN PUSTAKA

2.1 Teori Berhubungan dengan Struktur Data

Array adalah tipe data terstruktur yang berguna untuk menyimpan sejumlah data dengan tipe data yang sama. Misalkan, array A berupa kumpulan variabel bertipe integer, dimana masing-masing elemen memiliki nilai indeks. *Array* dibedakan menjadi dua, yaitu *One-Dimensional Array* dan *Two-Dimensional Array.*

Struct adalah sebuah cara untuk mengelompokkan bermacam-macam variabel yang berkaitan satu sama lain di dalam satu tempat. Berbeda dengan *array*, *struct* dapat terdiri atas variabel dengan macam-macam tipe data yang berbeda (int, string, float, dll).

Queue adalah data structure bertipe linear yang berfungsi untuk menyimpan dan memanipulasi elemen pada data. *Queue* memiliki konsep *First-In-First-Out (FIFO),* dimana data yang pertama masuk akan keluar pertama. Data baru masuk di ujung satu atau disebut dengan *rear*, sementara data keluar di ujung lainnya disebut dengan *front.*

Switch-Case adalah sebuah *statement* yang berfungsi untuk mengeksekusi baris-baris code. Konsep *Switch-Case* mirip dengan konsep *if-else-if-else ladder,* hanya saja *Switch-Case* lebih mudah dipahami. Apabila *switch* yang dipilih 1, maka akan menampilkan baris coding *case* 1. *Switch-Case* ini umum digunakan untuk membuat sebuah menu jika dikombinasikan dengan *do-while loop.*

Umumnya, operasi dalam *min-heap* adalah sebagai berikut.

1. Insertion
2. Swap
3. Deletion
4. Display

2.2 Teori Umum

Priority Queue adalah sebuah jenis *queue* dimana elemen-elemen di dalamnya diasosiasikan dengan sebuah *priority value* atau nilai prioritas. Jadi, elemen di dalam *queue* diurutkan berdasarkan prioritas mereka, dimana elemen dengan prioritas yang lebih tinggi akan didahulukan. Sedikit berbeda dengan *queue,* elemen yang dihapus terlebih dahulu dalam *priority queue* didasarkan pada prioritas

mereka, sehingga jenis *queue* ini tidak menggunakan aturan FIFO (*first-in-first out)* seperti pada *queue* biasa.

Dalam implementasinya, *priority queue* dapat dibuat menggunakan beberapa metode, seperti *array, linked list, heap,* atau *binary search tree.* Dari beberapa metode tersebut, *heap* merupakan metode yang efisien untuk *priority queue* karena efisiensi waktu *heap* adalah 0(1) untuk operasi *peek* dan 0(log n) untuk operasi *insert* dan *delete*.

Heap adalah sebuah struktur data yang terorganisir dan terurut. *Minimum Heap Tree (Min-Heap)* merupakan salah satu implementasi dari *binary tree* dimana *value* dari *children node* lebih besar dibandingkan dengan *parent node. Min-Heap* adalah bentuk dari *complete binary tree. Complete binary tree* adalah sebuah bentuk tree dimana seluruh level, kecuali pada level terakhir, terisi penuh. Oleh karena *heap* merupakan *complete binary tree*, maka elemen-elemen pada *heap* dapat disimpan di dalam sebuah *array. Min-Heap* merupakan salah satu metode yang digunakan untuk mengurutkan data dari nilai terkecil sebagai *root* dan *descendants' root* memiliki nilai lebih besar dibandingkan *root,* yang berarti nilai terkecil berada di *level order* 0.

Nilai atau *key* yang dibandingkan bisa berupa tipe data *integer* yang berisi angka, ataupun *string* yang berisi karakter huruf. Dikarenakan *heap sort* menggunakan *binary sort* untuk mengurutkan array dengan efisiensi waktu 0(nLogn), menggunakan *Linked List* dalam operasi *heap* bisa dibilang menghilangkan seluruh efisiensi tersebut. Contoh dari *heap* yang biasa digunakan adalah Binomial Heap dan Fibonacci Heap.

BAB 3

GAMBARAN UMUM PROGRAM

3. 1 Pseudocode dan Flowchart

3.1.1 Pseudocode

Priority Queue using Heap

PSEUDOCODE

Using C library:

- stdio.h
- string.h
- stdlib.h

//Display and choose menu

STEP 1: DISPLAY the following menu

1. Add data
2. Display root
3. Display table
4. Display sorted table
5. Display minimum heap flow
6. Display heap tree
0. Exit

STEP 2: Choose menu

STEP 3: While case (menu) is not 0

IF chosen menu is 1

Input table number, order, and estimated time.

Go to function insert

Go to function upheap

Display (“--- Element successfully added. ---“)

Break

IF chosen menu is 2

Go to function delRoot

Break

```
        IF chosen menu is 3
                Go to function displayTable
                Display ("--- Min successfully deleted. ---")
                Break

        IF chosen menu is 4
                Go to function displaySort
                Break

        IF chosen menu is 5
                Go to function displayTime
                Break

        IF chosen menu is 6
                Go to function displayTree
                Break

        IF chosen menu is 0
                Display ("--- Thank you. —")
                Break
        Return 0
//function insert
//Insertion to an empty min heap
STEP 1: IF min heap is empty
                Insert new node as root
                Increment size by 1
           [End of if]


//Insertion into a min heap
STEP 1: Increment size by 1
STEP 2: Store data in struct data
STEP 3: Go to function upheap
```

//sorting and comparing the new node to parent

//function upheap

STEP 1: declare parent index as i/2

STEP 2: IF i is smaller than size AND array index i is bigger than the current i

Go to function swap

Go to function swapChar

Go to function upheap

[End of IF]

STEP 3: Repeat step 2 until the heap order property is restored.

//Deletion from a min heap

//function delRoot

STEP 1: Find the minimum node.

STEP 2: Replace the minimum node (root) with the last leaf node in level order

STEP 3: Compare the replacement against its children. If one of the children is smaller, swap the replacement with the smallest child.

IF child[ ] < replacement

Swap

[End of if]

STEP 4: Repeat step 3 until the heap order property is restored.

//Display Tree in table form

//function displayTable

STEP 1: Display table head

STEP 2: Display root and descendants

STEP 3: FOR LOOP i is 1, as long as i is less or equal to size, increment i by 1

Display (“num, number of table, order, and estimated time”)

[End of Loop]

STEP 4: Repeat step 2 until all the data is displayed.

//Display sorted table, table is sorted by time

//function displaySort

STEP 1: go to function bubblesort

STEP 2: Display table head

STEP 3: FOR LOOP i is 1, as long as i is less or equal to size, increment i by 1

Display (“num, number of table, order, and estimated time”)

[End of Loop]

STEP 4: Repeat step 2 until all the data is displayed.

//sort data by time using bubblesort

//function bubbleSort

//to sort data by time, from quickest to slowest estimated time

STEP 1: FOR LOOP i is 1, as long as i is less or equal to size, increment i by 1

FOR LOOP j is 1, as long as j is less or equal to size, increment j by 1

Compare time[j] with time[j+1]

IF time[j] is more than time[j+1]

Swap time[j] with time[j+1]

Swapchar string[j] with string[j+1]

Swap tablenum[j] with tablenum[j+1]

[End of If]

[End of Loop]

[End of Loop]

STEP 2: Go to the next data

STEP 3: Repeat step 1 and two until the data is sorted.

//Display heap flow

//function displayTime

STEP 1: FOR LOOP i is 1, as long as i is less or equal to size, increment i by 1

Display (“-> time”)

Then go to the next node

[End of Loop]

STEP 2: Repeat step 1 until all the nodes are displayed.

//Display heap tree, tree is sideways

//function displayTree

STEP 1: Declare variable count = 10

STEP 2: IF data index is 0

Return to menu

[End of If]

STEP 3: Declare space = space + count

STEP 4: Call the function displayTree, with the variable being space and the index is index*2 + 1 for right child

STEP 5: Newline

STEP 6: FOR LOOP i is count, as long as i is less than space, increment i by 1

Print “ “

[End of Loop]

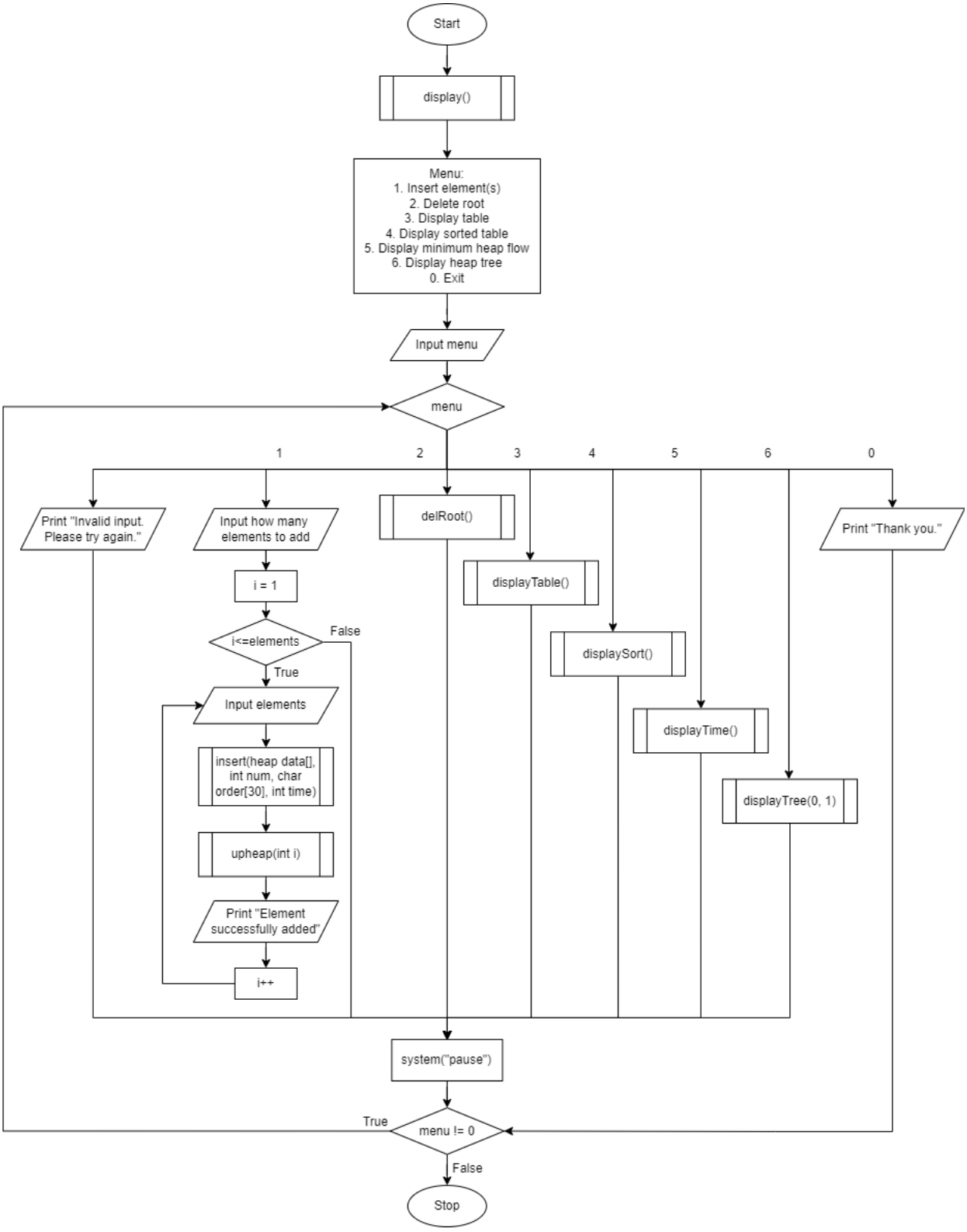
STEP 7: IF data index is not equal to 999999

Print data time

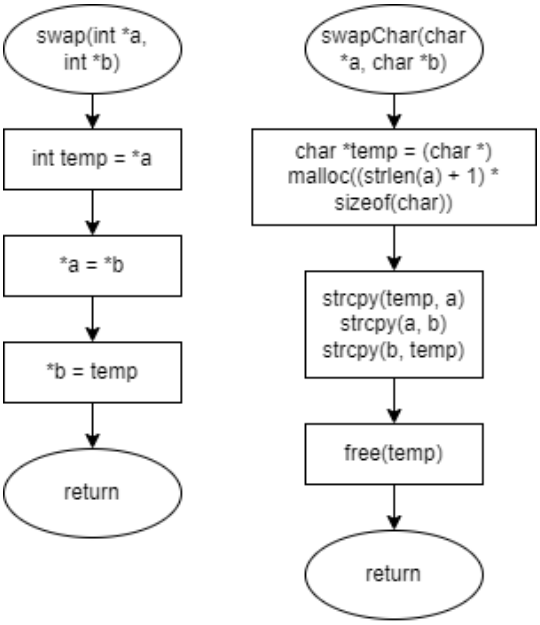
[End of If]

STEP 8: Call the function displayTree, with the variable being space and the index is index*2 for left child

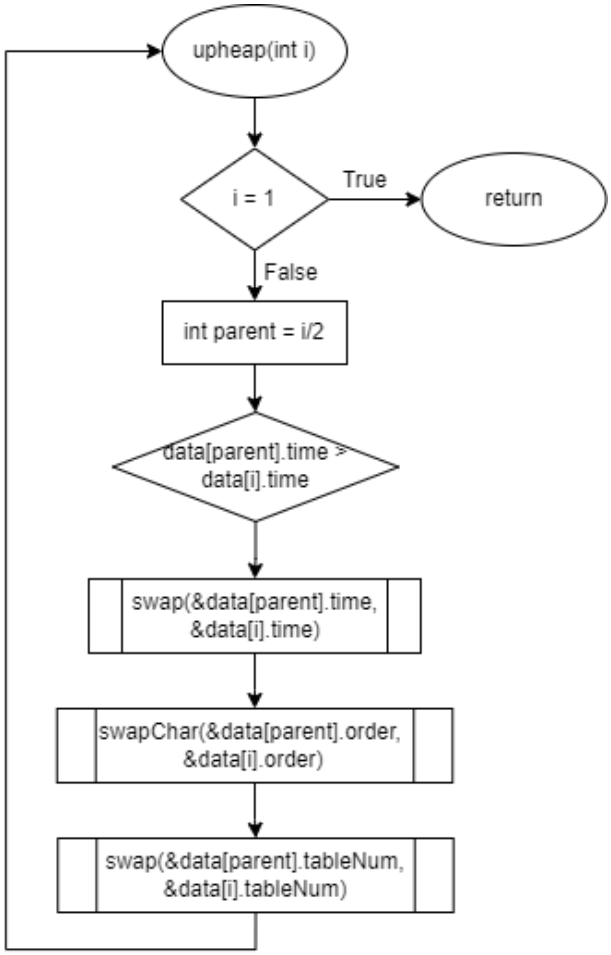
3.1.2 Flowchart



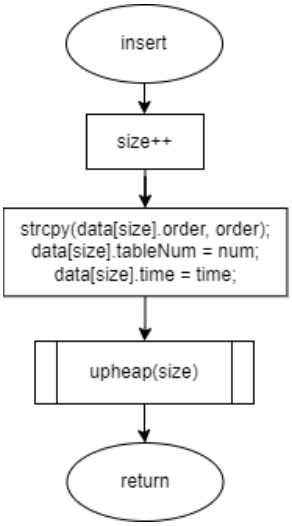
Gambar 3.1.1 Flowchart fungsi main



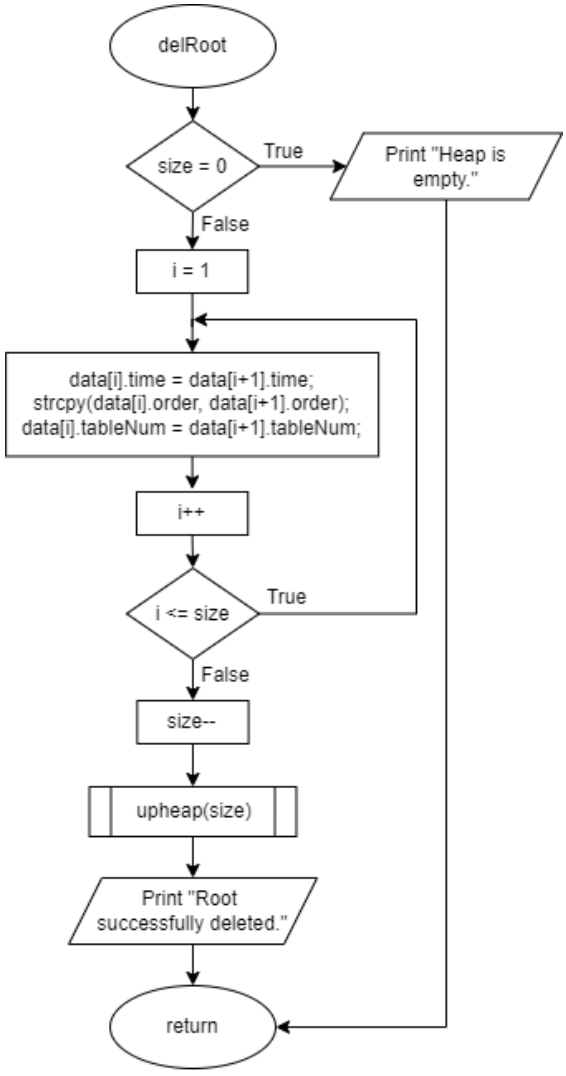
Gambar 3.1.2 Flowchart fungsi swap dan swapChar



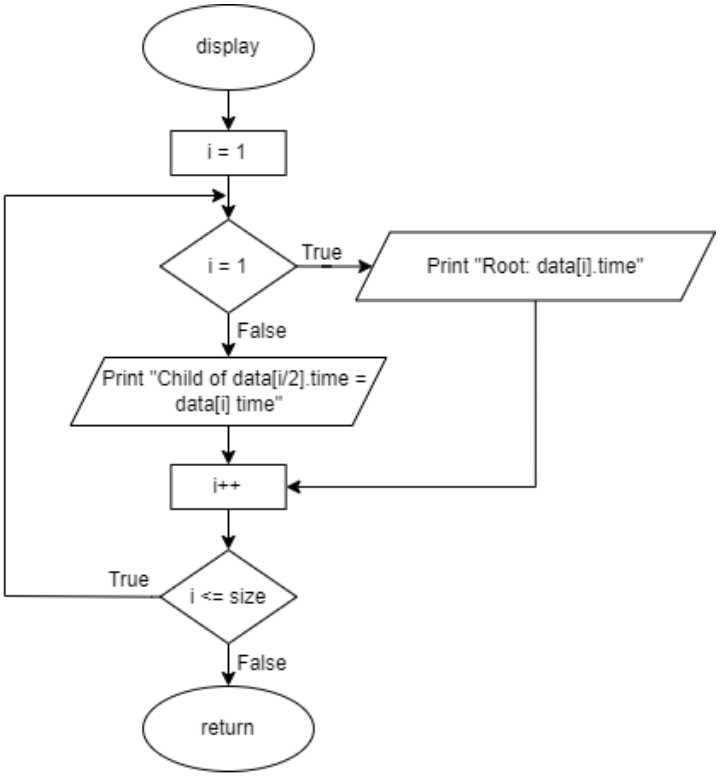
Gambar 3.1.3 Flowchart fungsi upheap



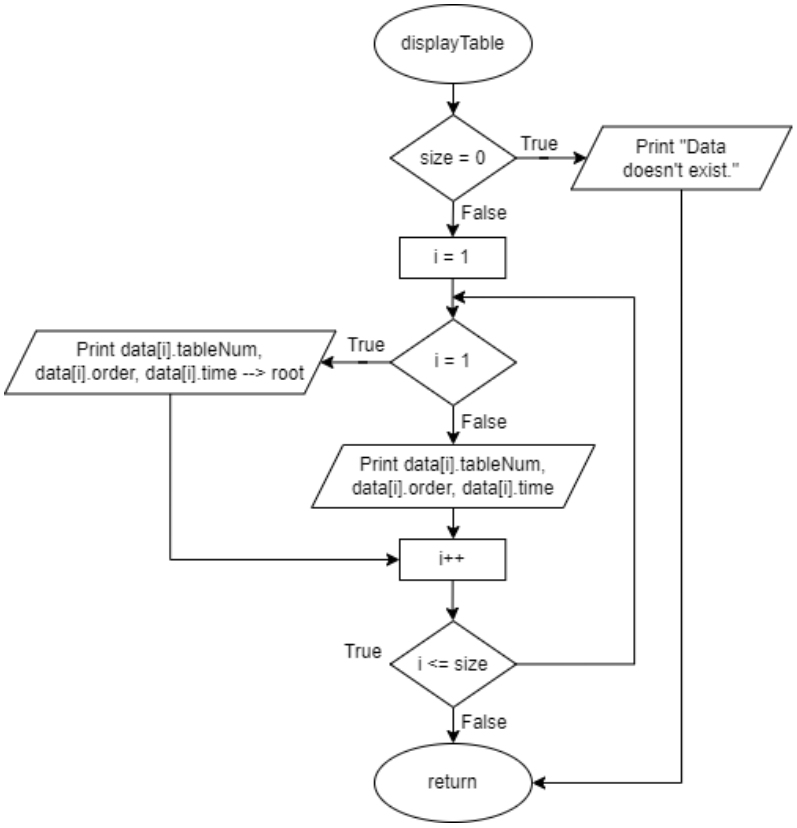
Gambar 3.1.4 Flowchart fungsi insert



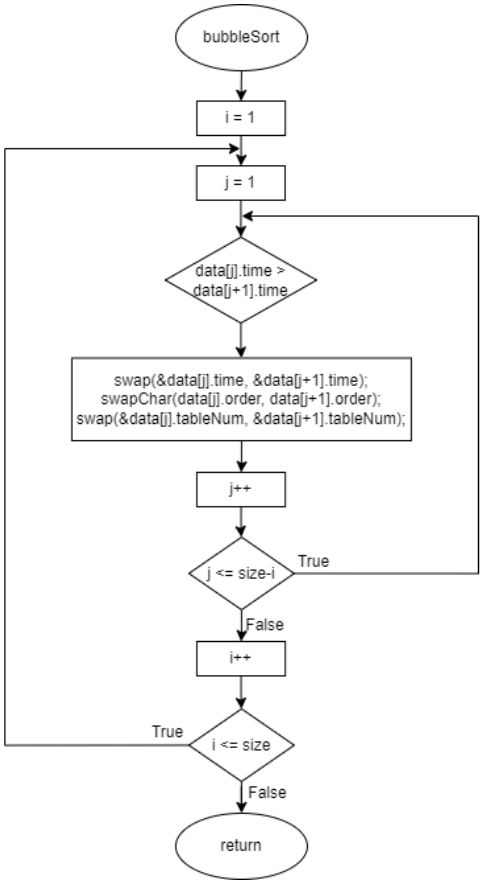
Gambar 3.1.5 Flowchart fungsi delRoot



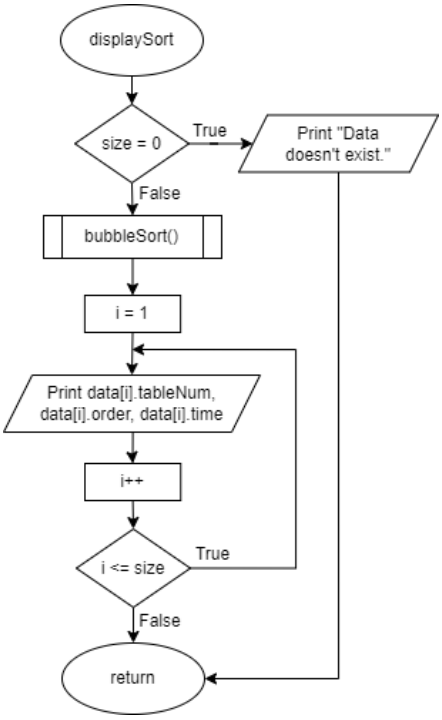
Gambar 3.1.6 Flowchart fungsi display



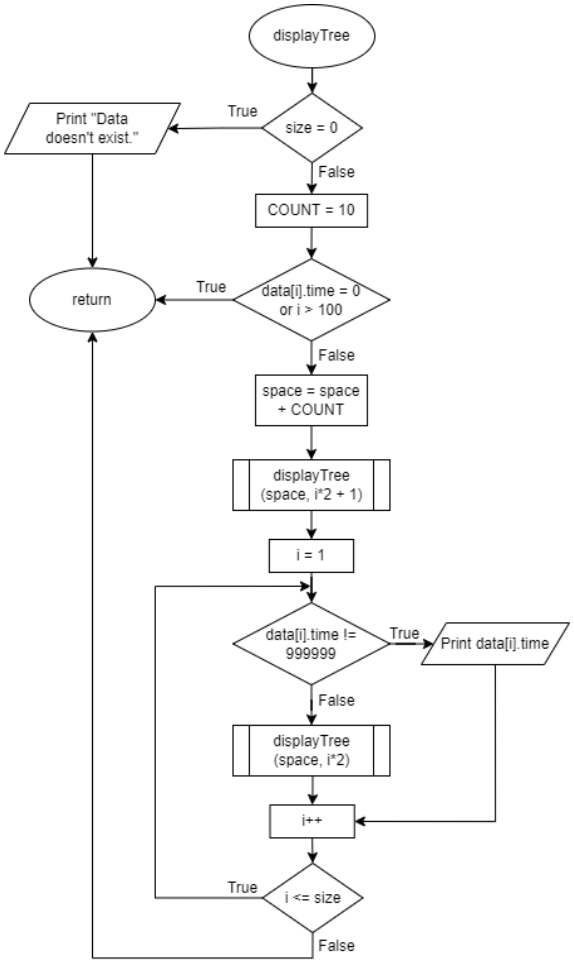
Gambar 3.1.7 Flowchart fungsi displayTable



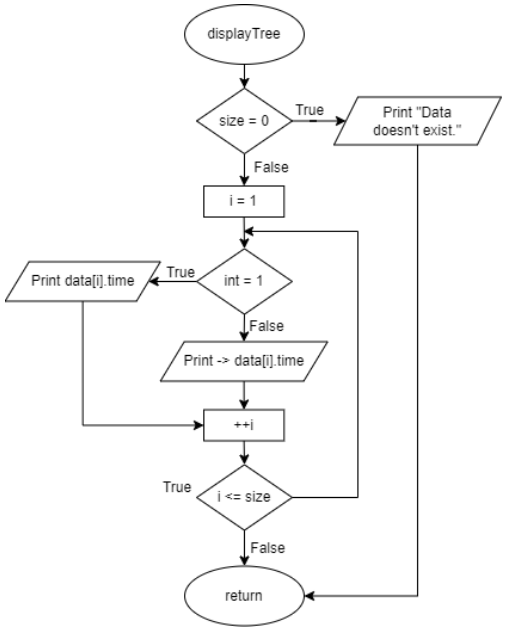
Gambar 3.1.8 Flowchart fungsi bubbleSort



Gambar 3.1.9 Flowchart fungsi displaySort



Gambar 3.1.10 Flowchart fungsi displayTree



Gambar 3.1.11 Flowchart fungsi displayTree

3.2 Program Overview

3.2.1 Deskripsi Program

Program yang dibuat terdiri atas 275 baris yang terdiri atas 11 *function void* dan 1 *function main*.

Function tersebut antara lain adalah sebagai berikut.

a. Void swap

Function ini berguna untuk menukar posisi data *key* dengan tipe data *integer*, atau variabel *pointer* a dengan *pointer* b dibantu dengan *variable temp*.

b. Void swapChar

Sama seperti *void swap, void swapChar* berguna untuk menukar posisi data. Perbedaannya, *void swapChar* ini menukar variabel *string.*

c. Void upheap

Function ini berfungsi untuk mengurutkan data-data di dalam *array* dengan cara membandingkan nilai nilai *child* dengan nilai *parent.* Apabila nilai *child* lebih kecil, maka kedua nilai akan ditukar dengan memanggil fungsi *swap* dan *swapChar.*

d. Void insert

Function ini akan mencatat ulang data baru yang dicatat dari *input* user pada function *main,* kemudian dimasukkan ke dalam data dalam *array.* Selanjutnya, function akan memanggil function *upheap* untuk memposisikan data pada *heap.*

e. Void delRoot

Function ini berfungsi untuk menghapus *root* dari *heap* atau *priority queue.* Function ini menghapus data dengan cara mengisi setiap data dalam *array* dengan data indeks *array* selanjutnya. Setelah itu, data kembali diposisikan dengan cara memanggil function *upheap.*

f. Void displayTable

Function ini menampilkan data yang ada di dalam *array* dalam bentuk tabel. Beberapa data yang ditampilkan pada tabel antara lain nomor, nomor meja, pesanan, dan estimasi waktu pembuatan.

g. Void display

Function ini menampilkan data dalam *array* dengan menampilkan *root* dan *child(ren)* dari root tersebut.

h. Void bubbleSort

Function ini bertugas untuk mengurutkan data yang ada di dalam array menggunakan metode *bubble sort.* Function ini kemudian akan dipanggil pada function void displaySort.

i. Void displaySort

Function ini menampilkan data yang ada di dalam *array* dalam bentuk tabel dan dalam formasi yang terurut berdasarkan atribut “Est. time”, dari waktu yang paling sedikit hingga paling banyak.

j. Void displayTree

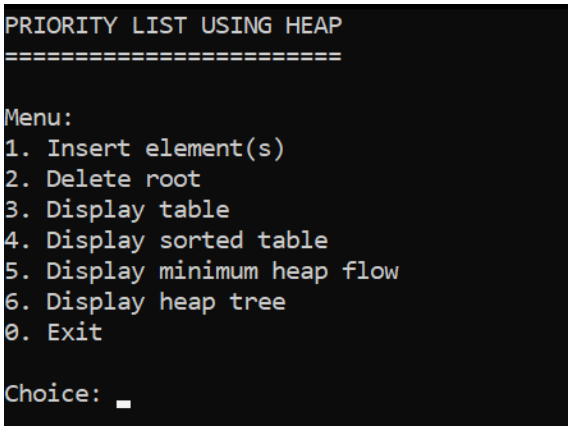
Function ini menampilkan data yang ada di dalam *array* dalam bentuk formasi *tree.* Function ini merupakan function *recursive* karena akan memanggil ulang fungsinya sendiri. Pada function ini, *tree* tidak ditampilkan mengarah ke bawah, melainkan ke arah kanan.

k. Void displayTime

Void ini menampilkan data dalam bentuk *flow* estimasi waktu, dimulai dari *root* dan berlanjut ke anak kiri kemudian kanan hingga semua node sudah di proses.

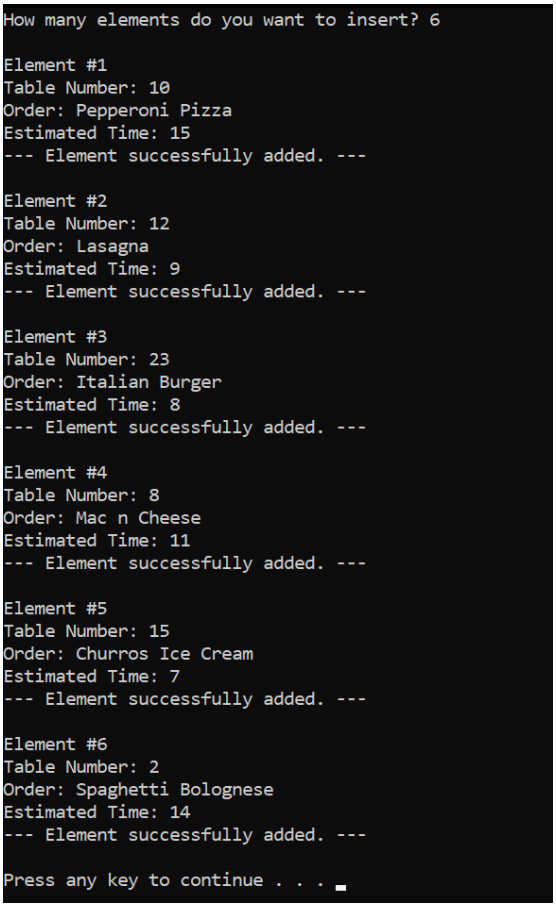
3.2.2 Layout Design

Pada awal menjalankan aplikasi, *user* akan diberikan beberapa pilihan menu, dan akan meminta *user* untuk memasukkan menu yang ingin dijalankan. Beberapa menu yang ada adalah (1) Insert element(s), (2) Delete root, (3) Display table, (4) Display sorted table, (5) Display minimum heap flow, (6) Display heap tree, dan (0) Exit.



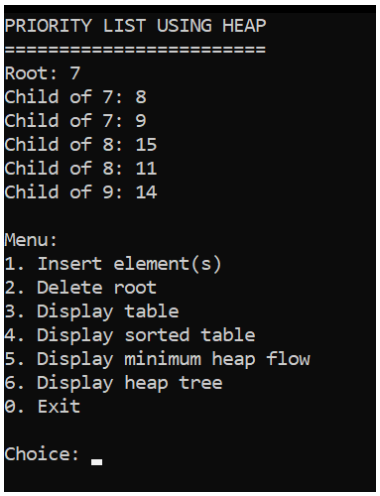
Gambar 3.2.1 Tampilan awal aplikasi

Pada menu 1, yaitu menu untuk menambahkan elemen pada *heap,* aplikasi akan meminta *user* untuk memasukkan jumlah data yang ingin dimasukkan pada *heap.* Setelah itu, *user* akan diminta untuk memasukkan data nomor meja, menu pesanan, dan estimasi waktu pembuatan pada aplikasi sebanyak jumlah data yang telah dimasukkan sebelumnya.



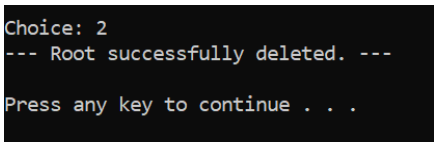
Gambar 3.2.2 Input data pada menu 1

Setelah itu, *user* akan kembali pada tampilan aplikasi awal dengan data yang telah dimasukkan sebelumnya. Data yang ditampilkan pada aplikasi adalah data estimasi waktu. Kemudian, *user* dapat melanjutkan menjalankan aplikasi.

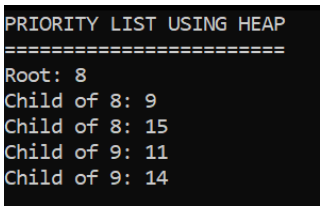


Gambar 3.2.3 Tampilan aplikasi awal setelah input

Pada menu 2, yaitu menu untuk menghapus *root, user* dapat menghapus data pertama (prioritas paling atas) apabila pesanan telah selesai dibuat. Pada menu ini, program akan menghapus *root* dari *heap* kemudian merapikan kembali data dalam *heap*.

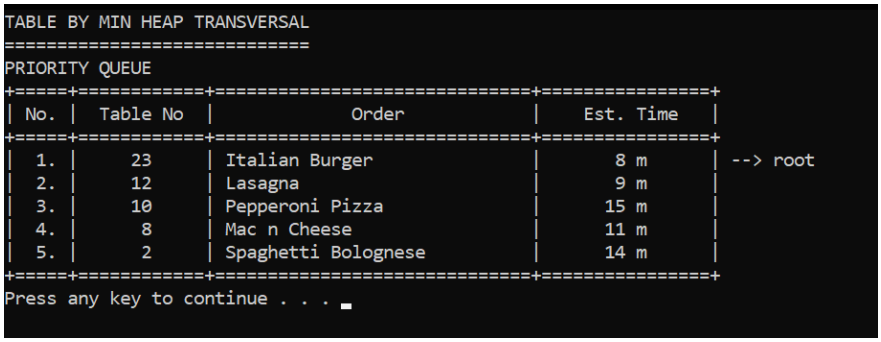


Gambar 3.2.4 Menghapus root pada menu 2



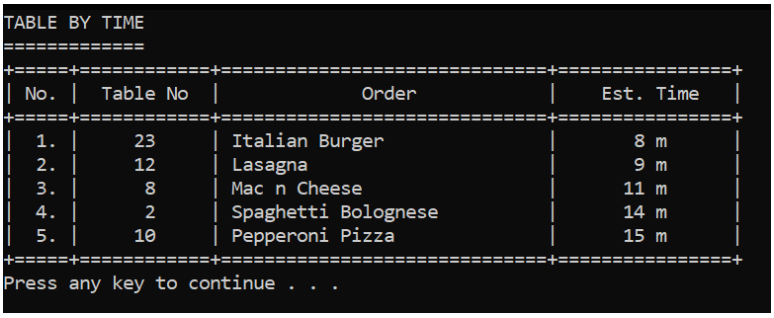
Gambar 3.2.5 Tampilan aplikasi awal setelah delete

Pada menu 3, yaitu menu display table, aplikasi akan menampilkan data pada *heap* dalam bentuk tabel sesuai dengan urutan pada *heap.* Pada tabel akan ditampilan nomor meja, menu pesanan, estimasi waktu pembuatan, serta data yang merupakan *root.*



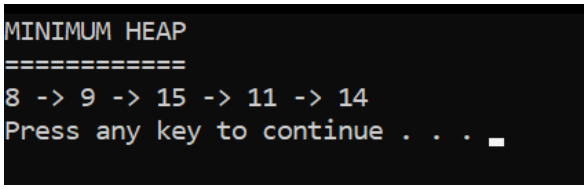
Gambar 3.2.6 Tampilan tabel pada menu 3

Pada menu 4, yaitu menu untuk *sorted table,* aplikasi akan menampilkan data pada *heap* dalam bentuk tabel. Pada menu ini, data dalam tabel akan diurutkan dari estimasi waktu tercepat hingga waktu terlambat menggunakan *bubble sort.*



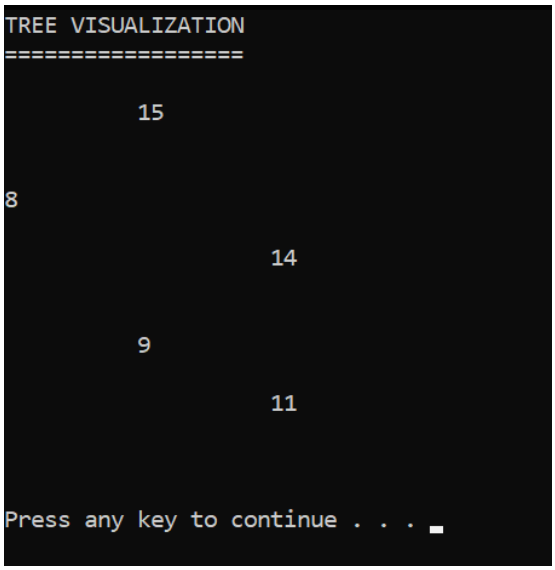
Gambar 3.2.7 Tampilan *sorted table*

Pada menu 5, yaitu menu untuk menampilkan *flow* dari *min-heap*, program akan menampilkan estimasi waktu dimulai dari *root* dan dilanjut ke anak kiri dan anak kanan. Menu ini menampilkan *flow* menggunakan *root-left child-right child* atau bisa juga dibilang metode menampilkan data berdasarkan *level order*.



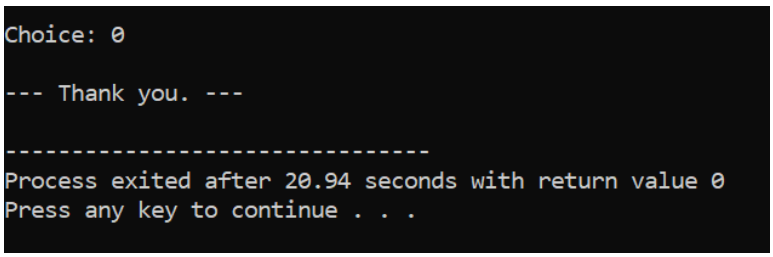
Gambar 3.2.7 Tampilan *flow minimum heap*

Pada menu 6, yaitu menu untuk menampilkan visualisasi dari *heap tree,* aplikasi akan menampilkan data-data dari *heap* dalam bentuk *tree* yang mengarah ke kanan, dimana node bawah adalah node kiri dan node atas adalah node kanan.



Gambar 3.2.9 Tampilan visualisasi *tree*

Pada menu 0 atau menu exit, aplikasi akan selesai berjalan dan seluruh proses pada program akan berhenti.



Gambar 3.2.10 Tampilan akhir aplikasi pada menu 0

3.2.3 Fitur Program

a. Insert data

Menu ini berguna untuk memasukkan node baru dalam sebuah *binary tree*. Fungsi *insertion* ini akan meminta user memasukkan data elemen yang ingin dimasukkan ke dalam *heap*, kemudian membandingkan nilai tersebut dengan node-node yang telah ada, apakah node baru lebih kecil atau lebih besar dibandingkan dengan *parent node.* Dalam *min-heap,* apabila node baru memiliki *value* yang lebih kecil dibandingkan dengan *parent,* maka posisi antara *node* baru dan *parent* akan ditukar. Fungsi ini akan terus dipanggil *(recursive)* hingga seluruh node telah terurut.

b. Delete root

Menu ini berguna untuk menghapus *node* dengan *value* terkecil atau *root.* Setelah fungsi ini dipanggil kemudian data yang dihapus akan digantikan dengan *node* dengan *value* terkecil selanjutnya.

c. Display table

Menu ini berguna untuk menampilkan data yang sudah diurutkan. Dalam kasus *min-heap,* tampilan data dimulai dari data *root* dan berlanjut ke data anak kiri dan anak kanan hingga semua *node* telah ditampilkan. Data yang ditampilkan adalah nomor urut, nomor meja, order, dan estimasi waktu. Apabila data adalah *root,* akan muncul *marker root* di samping data.

d. Display sorted table

Menu ini berguna untuk menampilkan data yang sudah diurutkan berdasarkan estimasi waktu terkecil hingga terbesar. Data yang ditampilkan adalah nomor urut, nomor meja, order, dan estimasi waktu.

e. Display minimum heap flow

Menu ini berguna untuk menampilkan flow data estimasi waktu dengan simbol "->". Data dimulai dari root dan berlanjut ke anak kiri dan anak kanan hingga semua node telah ditampilkan.

f. Display heap tree

Menu ini berguna untuk menampilkan min heap tree secara horizontal. Data yang ditampilkan adalah estimasi waktu. Data dimulai dari *root* yang akan terletak di kiri, dan berlanjut ke anak kiri yang yang berada di bawah, kemudian anak kanan yang berada di atas, dan seterusnya hingga semua data telah ditampilkan.

3.2.4 Detail Program

C Library yang digunakan adalah stdio.h, string.h, dan stdlib.h.

Struct heap berisi char order[30], int time, dan int tableNum.

Parent index = i/2

Left child index = 1*2

Right child index = 1*2 + 1

Function:

Void swap adalah fungsi yang digunakan untuk penukaran variable integer.

Void swapchar adalah fungsi yang digunakan untuk penukaran variabel char.

Void upheap adalah fungsi untuk sorting value anak yang lebih besar dari parentnya.

Void insert adalah fungsi untuk memasukkan data baru.

Void delRoot adalah fungsi untuk menghapus data terkecil atau *root.*

Void displayTable adalah fungsi untuk menampilkan data pada tabel dengan urutan level order.

Void display adalah fungsi untuk menampilkan data dengan penggunaan level order

Void displayTree adalah fungsi untuk menampilkan data dalam bentuk tree

Void bubbleSort adalah fungsi untuk mengurutkan data dimulai dengan estimasi waktu tercepat hingga terlama.

Void displaySort adalah fungsi yang digunakan untuk menampilkan data yang telah diurutkan dalam bentuk tabel.

Void displayTime adalah fungsi yang digunakan untuk menampilkan *flow min-heap.*

BAB 4

HASIL

4.1 Screenshot Program

```
//Array Implementation of MinHeap Data Structure
//Kelompok 4

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

int size = 0;

struct heap{
    char order[30];
    int time;
    int tableNum;
};

heap data[100];
```

Program menggunakan beberapa *library*, di antaranya adalah stdio.h, string.h, dan stdlib.h. Pada program ini, digunakan sebuah *struct* dengan isi char order[30] untuk menu pesanan, int time untuk estimasi waktu, dan int tableNum untuk nomor meja. *Struct* tersebut dimasukkan ke dalam *array* data[100]. Pada awal program juga dideklarasikan sebuah *integer* "size" yang bernilai 0. *Integer* ini digunakan untuk mengukur ada berapa banyak data di dalam *array*.

```
//to swap the key variable(s)
void swap(int *a, int *b){

//to swap the string variable(s)
void swapChar(char *a, char *b){

//to sort data
//sort if the newnode value is smaller than the parent
void upheap(int i){

//to insert new data
void insert(heap data[], int num, char order[30], int time){

//to delete the root node from heap
void delRoot(){

//to display the root and its child(ren)
void display(){

//to display data in table form
void displayTable(){

//to bubble sort element by time
void bubbleSort(){

//to display bubblesort data in table form
void displaySort(){

//to display heap in tree visualization
//sideways tree
void displayTree(int space, int index){

//to display minimum heap flow
void displayTime(){
```

Program ini menggunakan 11 function *void*. Function-function tersebut akan kemudian digunakan untuk menjalankan menu yang ada pada function *main,* atau pada function *void* lainnya. Beberapa function ini adalah *void* swap dan swapChar yang berfungsi untuk menukar 2 data, *void* upheap untuk mengurutkan data *heap, void* insert untuk memasukkan data baru, *void* delRoot untuk menghapus data *root,* dan *void* display untuk menampilkan data pada *array heap.*

```
//to swap the key variable(s)
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Function swap berfungsi untuk menukar posisi dua data (pada gambar int a dan int b). Pada function ini, digunakan sebuah variabel tambahan berupa int temp untuk membantu pertukaran data.

```
//to swap the string variable(s)
void swapChar(char *a, char *b){
    char *temp = (char *)malloc((strlen(a) + 1) * sizeof(char));
    strcpy(temp, a);
    strcpy(a, b);
    strcpy(b, temp);
    free(temp);
}
```

Function swapChar memiliki fungsi yang sama dengan function swap, yaitu untuk menukar dua buah data. Pada function ini, data yang ditukar menggunakan tipe data *string* atau *char.* Sama seperti function swap, function swapChar juga menggunakan variabel tambahan yaitu char temp.

```
//to sort data
//sort if the newnode value is smaller than the parent
void upheap(int i){
    if(i == 1) return;

    int parent = i/2;
    if(data[parent].time > data[i].time){
        swap(&data[parent].time, &data[i].time);
        swapChar(data[parent].order, data[i].order);
        swap(&data[parent].tableNum, &data[i].tableNum);
        upheap(parent);
    }
}
```

Function upheap berfungsi untuk memposisikan atau mengurutkan data pada *heap.* Proses upheap bekerja dengan cara membandingkan dua nilai data, yaitu nilai *parent* dan nilai *child.* Pada kasus *min-heap,* nilai yang memiliki prioritas tertinggi adalah nilai terkecil. Jadi, apabila nilai *child* lebih kecil dari nilai *parent*, maka kedua data tersebut akan ditukar dengan cara memanggil function swap dan swapChar. Function ini merupakan function *recursive* karena function ini memanggil dirinya sendiri. Pada kasus ini, pengurutan data dilakukan berdasarkan estimasi waktu.

```
//to insert new data
void insert(heap data[], int num, char order[30], int time){
    size++;
    strcpy(data[size].order, order);
    data[size].tableNum = num;
    data[size].time = time;

    upheap(size);
}
```

Function insert berfungsi untuk memasukkan atau mencatat data baru dari hasil *input* oleh *user* yang dilakukan pada function *main.* Karena function ini menambahkan data baru, maka size perlu untuk

ditambah. Setelah itu, program akan mencatat data baru ke dalam *array* data, kemudian memanggil function upheap untuk memposisikan data baru ke dalam *heap.*

```
//to delete the root node from heap
void delRoot(){
    if(size == 0) printf("--- Heap is empty. ---\n");
    else{
        for(int i=1; i<=size; i++){
            data[i].time = data[i+1].time;
            strcpy(data[i].order, data[i+1].order);
            data[i].tableNum = data[i+1].tableNum;
        }

        size--;
        upheap(size);

        printf("--- Root successfully deleted. ---\n\n");
    }
}
```

Function delRoot berfungsi untuk menghapus data pada indeks pertama *array*. Cara function ini untuk menghapus *root* adalah dengan mengisi masing-masing data pada suatu indeks dengan nilai indeks setelahnya. Setelah itu, function ini juga akan mengurangi nilai int size serta memanggil function upheap untuk memposisikan kembali masing-masing data pada *heap.*

```
//to display the root and its child(ren)
void display(){
    for(int i=1; i<=size; i++){
        if(i == 1) printf("Root: %d\n", data[i].time);
        else{
            printf("Child of %d: %d\n", data[(i)/2].time, data[i].time);
        }
    }
    puts("");
}
```

Function display berfungsi untuk menampilkan data di dalam *array* data. Pada function ini, data akan ditampilkan dengan cara menampilkan *root* dari *heap tree,* diikuti dengan *child(ren)*nya. Untuk menampilkan data, digunakan *for loop* dan *if-else condition.*

```
//to display data in table form
void displayTable(){
    if(size == 0){
        printf("--- Data doesn't exist. ---\n\n");
        printf("");
    }
    else{
    printf("PRIORITY QUEUE\n");
        printf("+=====+===========+============================+================+\n");
        printf("| No. |  Table No  |            Order           |    Est. Time   |\n");
        printf("+=====+===========+============================+================+\n");
        for(int i=1; i<=size; i++){
            if(i == 1) printf("|  %d. | %6d     | %-28s | %7d m      | --> root\n", i, data[i].tableNum, data[i].order, data[i].time);
            else printf("|  %d. | %6d     | %-28s | %7d m      |\n", i, data[i].tableNum, data[i].order, data[i].time);
        }
        printf("+=====+===========+============================+================+\n");
    }
}
```

Function displayTable berfungsi untuk menampilkan data di dalam *array* ke dalam bentuk tabel. Data yang disajikan pada tabel terdiri atas nomor, nomor meja, menu pesanan, dan estimasi waktu pembuatan. Data yang ditampilkan diurutkan berdasarkan data yang disimpan di dalam *array* atau *heap.* Pada tabel ini juga akan ditunjukkan posisi *root*, yang mana adalah data pertama.

```
//to bubble sort element by time
void bubbleSort(){
    int i, j;
    for (i = 1; i <= size; i++)

        // Last i elements are already in place
        for (j = 1; j <= size-i; j++)
            if (data[j].time > data[j + 1].time){
                swap(&data[j].time, &data[j + 1].time);
                swapChar(data[j].order, data[j+1].order);
                swap(&data[j].tableNum, &data[j+1].tableNum);
            }
}
```

Function bubbleSort berfungsi untuk mengurutkan data pada *array*. Berbeda dengan function upheap yang hanya membandingkan nilai *child* dengan nilai *parent,* function ini membandingkan seluruh data, sehingga hasil dari function ini adalah data yang benar-benar terurut, mulai dari nilai yang terkecil hingga terbesar. Function ini menggunakan metode *bubble sort*, yaitu dengan cara membandingkan dua data yang bersebelahan.

```
//to display bubblesort data in table form
void displaySort(){
    if(size == 0)
        printf("--- Data doesn't exist. ---\n\n");
    else{
    bubbleSort();
        printf("+=====+===========+============================+================+\n");
        printf("| No. |  Table No  |            Order           |    Est. Time   |\n");
        printf("+=====+===========+============================+================+\n");
        for(int i=1; i<=size; i++){
            printf("|  %d. | %6d     | %-28s | %7d m      |\n", i, data[i].tableNum, data[i].order, data[i].time);
        }
        printf("+=====+===========+============================+================+\n");
    }
}
```

Function displaySort menampilkan data yang ada di dalam *array* dalam bentuk tabel dengan data yang telah diurutkan. Cara kerja function ini adalah dengan cara memanggil function bubbleSort terlebih dahulu, kemudian menampilkan tabel dengan menggunakan *for loop.*

```
//to display heap in tree visualization
//sideways tree
void displayTree(int space, int index){
    if(size == 0){
        printf("--- Data doesn't exist. ---\n\n");
        printf("");
    }
    else{
        int COUNT = 10;
        if(data[index].time== 0 || index>100){
            return;
    }
    space += COUNT;

    displayTree(space,index*2 + 1);
    printf("\n");

    for(int i=COUNT; i<space; i++){
        printf(" ");
    }

    if(data[index].time != 999999)
        printf("%d\n", data[index].time);
    displayTree(space, index*2);

    puts("");
    }
}
```

Function displayTree menampilkan data di dalam *array* dalam bentuk *tree* yang mengarah ke kanan. Function ini merupakan function *recursive* karena memanggil function itu kembali. Cara kerja function ini adalah dengan menggunakan spasi dan *for loop* hingga semua node telah ditampilkan. Node yang ditampilkan adalah estimasi waktu setiap meja.

```
//to display minimum heap flow
void displayTime(){
    if(size == 0) printf("--- Data doesn't exist. ---\n");
    else{
        for(int i=1; i<=size; ++i){
        if(i == 1) printf("%d ", data[i].time);
        else printf("-> %d ", data[i].time);
        }
    }
}
```

Function displayTime menampilkan node dalam bentuk flow dengan simbol "->'. Node yang ditampilkan adalah estimasi waktu setiap meja. Node ditampilkan mulai dari node *root* dan berlanjut ke anak kiri dan anak kanannya, begitu seterusnya hingga semua node sudah ditampilkan dengan menggunakan *for loop*.

```
int main(){
    int menu, num;

    char newOrder[30];
    int newNum, est;

    do{
        system("cls");
        printf("PRIORITY LIST USING HEAP\n");
        printf("=======================\n");
        display();
        printf("Menu:\n");
        printf("1. Insert element(s)\n");
        printf("2. Delete root\n");
        printf("3. Display table\n");
        printf("4. Display sorted table\n")-
        printf("5. Display minimum heap flow\n");
        printf("6. Display heap tree\n");
        printf("0. Exit\n\n");
        printf("Choice: "); scanf("%d", &menu);
```

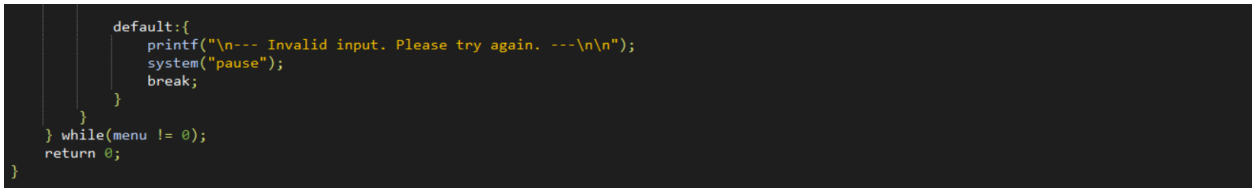
Seluruh function void sebelumnya akan kemudian dipanggil dan dijalankan pada function main. Pada function ini, digunakan *do-while loop* untuk menampilkan dan menjalankan masing-masing menu. Pada awal *loop,* digunakan function system("cls") untuk "membersihkan" seluruh layar pada aplikasi. Kemudian, program akan menampilkan menu serta meminta *input* dari *user* untuk melanjutkan jalannya program.

```
switch(menu){
    case 1:{
        //insert
        int n;
        system("cls");
        printf("INSERT\n");
        printf("======\n");
        printf("How many elements do you want to insert? "); scanf("%d", &n);
        puts("");
        for(int i=1; i<=n; i++){
            printf("Element #%d\n", i);
            printf("Table Number: "); scanf("%d", &newNum);
            printf("Order: "); getchar(); scanf("%[^\n]s", newOrder);
            printf("Estimated Time: "); scanf("%d", &est);
            insert(data, newNum, newOrder, est);
            printf("--- Element successfully added. ---\n\n");
        }
        system("pause");
        break;
    }
```

Untuk masuk ke dalam masing-masing menu, digunakan *switch case statement*. Untuk masuk ke menu 1, maka program akan menjalankan case 1, yaitu untuk *insert* data. Pada case ini, *user* akan diminta untuk memasukkan berapa banyak data yang ingin ditambahkan ke dalam *array*, kemudian memasukkan data-data seperti nomor meja, menu pesanan, dan estimasi waktu sebanyak jumlah yang sebelumnya telah dimasukkan. Setelah itu, data akan dimasukkan ke dalam *array* dengan cara memanggil function insert.

```
    case 2:{
        //delete root
        delRoot();
        system("pause");
        break;
    }

    case 3:{
        //display table
        system("cls");
        printf("TABLE BY MIN HEAP TRANSVERSAL\n");
        printf("============================\n");
        displayTable();
        system("pause");
        break;
    }

    case 4:{
        //display sorted table
        system("cls");
        printf("TABLE BY TIME\n");
        printf("=============\n");
        displaySort();
        system("pause");
        break;
    }
```

```
    case 5:{
        //display the minimum heap flow (est time)
        system("cls");
        printf("MINIMUM HEAP\n");
        printf("============\n");
        displayTime();
        printf("\n");
        system("pause");
        break;
    }

    case 6:{
        //display tree
        system("cls");
        printf("TREE VISUALIZATION\n");
            printf("==================\n");
            displayTree(0, 1);
            system("pause");
            break;
    }

    case 0:{
        printf("\n--- Thank you. ---\n");
        break;
    }
```



Pada case-case selanjutnya, program akan menjalankan menu dengan cara memanggil function-function sebelumnya pada masing-masing case. *Switch case statement* ini juga memiliki sebuah *case* default yang akan dijalankan apabila *user* salah menginput menu atau memasukkan angka yang tidak ada pada *switch case.*

Apabila case 0 atau menu exit dijalankan, aplikasi akan berhenti dan me*return* value 0. Aplikasi akan berhenti berjalan karena *loop* sudah tidak memenuhi syarat, dimana kondisi yang ada adalah nilai menu harus tidak sama dengan 0.

4.2 Code Program

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

int size = 0;

struct heap{
	char order[30];
	int time;
	int tableNum;
};

heap data[100];

//to swap the key variable(s)
void swap(int *a, int *b){
	int temp = *a;
	*a = *b;
	*b = temp;
}

//to swap the string variable(s)
void swapChar(char *a, char *b){
	char *temp = (char *)malloc((strlen(a) + 1) * sizeof(char));
	strcpy(temp, a);
	strcpy(a, b);
	strcpy(b, temp);
	free(temp);
}

//to sort data
//sort if the newnode value is smaller than the parent
void upheap(int i){
	if(i == 1) return;

	int parent = i/2;
	if(data[parent].time > data[i].time){
		swap(&data[parent].time, &data[i].time);
		swapChar(data[parent].order, data[i].order);
```

```
			swap(&data[parent].tableNum, &data[i].tableNum);
			upheap(parent);
		}
}

//to insert new data
void insert(heap data[], int num, char order[30], int time){
		size++;
		strcpy(data[size].order, order);
		data[size].tableNum = num;
		data[size].time = time;

		upheap(size);
}

//to delete the root node from heap
void delRoot(){
		if(size == 0) printf("--- Heap is empty. ---\n");
		else{
			for(int i=1; i<=size; i++){
				data[i].time = data[i+1].time;
				strcpy(data[i].order, data[i+1].order);
				data[i].tableNum = data[i+1].tableNum;
			}

			size--;
			upheap(size);

			printf("--- Root successfully deleted. ---\n\n");
		}
}

//to display the root and its child(ren)
void display(){
		for(int i=1; i<=size; i++){
			if(i == 1) printf("Root: %d\n", data[i].time);
			else{
				printf("Child of %d: %d\n", data[(i)/2].time, data[i].time);
			}
		}
		puts("");
}

//to display data in table form
void displayTable(){
		if(size == 0){
			printf("--- Data doesn't exist. ---\n\n");
			printf("");
		}
		else{
	    printf("PRIORITY QUEUE\n");

printf("+=====+============+==============================+================+\n");
			printf("| No. |  Table No  |             Order            |    Est. Time
|\n");

printf("+=====+============+==============================+================+\n");
			for(int i=1; i<=size; i++){
				if(i == 1) printf("|  %d. | %6d     | %-28s | %7d m      | -->
root\n", i, data[i].tableNum, data[i].order, data[i].time);
				else printf("|  %d. | %6d     | %-28s | %7d m      |\n", i,
data[i].tableNum, data[i].order, data[i].time);
			}
```

```
printf("+=====+============+================================+================+\n");
       }
}

//to bubble sort element by time
void bubbleSort(){
    int i, j;
    for (i = 1; i <= size; i++)

        // Last i elements are already in place
        for (j = 1; j <= size-i; j++)
            if (data[j].time > data[j + 1].time){
                swap(&data[j].time, &data[j + 1].time);
                swapChar(data[j].order, data[j+1].order);
                           swap(&data[j].tableNum, &data[j+1].tableNum);
                    }
}

//to display bubblesort data in table form
void displaySort(){
       if(size == 0)
              printf("--- Data doesn't exist. ---\n\n");
       else{
       bubbleSort();

printf("+=====+============+================================+================+\n");
              printf("| No. |  Table No  |             Order             |    Est. Time
|\n");

printf("+=====+============+================================+================+\n");
              for(int i=1; i<=size; i++){
                     printf("|  %d. | %6d     | %-28s | %7d m      |\n", i,
data[i].tableNum, data[i].order, data[i].time);
              }

printf("+=====+============+================================+================+\n");
       }
}

//to display heap in tree visualization
//sideways tree
void displayTree(int space, int index){
       if(size == 0){
              printf("--- Data doesn't exist. ---\n\n");
              printf("");
       }
       else{
              int COUNT = 10;
              if(data[index].time== 0 || index>100){
                     return;
       }
       space += COUNT;

       displayTree(space,index*2 + 1);
       printf("\n");

       for(int i=COUNT; i<space; i++){
              printf(" ");
       }

       if(data[index].time != 999999)
              printf("%d\n", data[index].time);
```

```
	displayTree(space, index*2);

	puts("");
	}
}

//to display minimum heap flow
void displayTime(){
	if(size == 0) printf("--- Data doesn't exist. ---\n");
	else{
		for(int i=1; i<=size; ++i){
		if(i == 1) printf("%d ", data[i].time);
		else printf("-> %d ", data[i].time);
		}
	}
}

int main(){
	int menu, num;

	char newOrder[30];
	int newNum, est;

	do{
		system("cls");
		printf("PRIORITY LIST USING HEAP\n");
		printf("=======================\n");
		display();
		printf("Menu:\n");
		printf("1. Insert element(s)\n");
		printf("2. Delete root\n");
		printf("3. Display table\n");
		printf("4. Display sorted table\n")-
	printf("5. Display minimum heap flow\n");
		printf("6. Display heap tree\n");
		printf("0. Exit\n\n");
		printf("Choice: "); scanf("%d", &menu);

		switch(menu){
			case 1:{
				//insert
				int n;
				system("cls");
				printf("INSERT\n");
				printf("======\n");
				printf("How many elements do you want to insert? ");
scanf("%d", &n);
				puts("");
				for(int i=1; i<=n; i++){
					printf("Element #%d\n", i);
					printf("Table Number: "); scanf("%d", &newNum);
					printf("Order: "); getchar(); scanf("%[^\n]s",
newOrder);
					printf("Estimated Time: "); scanf("%d", &est);
					insert(data, newNum, newOrder, est);
					printf("--- Element successfully added. ---\n\n");
				}
				system("pause");
				break;
			}

			case 2:{
				//delete root
```

```
                        delRoot();
                        system("pause");
                        break;
                    }

                    case 3:{
                        //display table
                    system("cls");
                    printf("TABLE BY MIN HEAP TRANSVERSAL\n");
                        printf("============================\n");
                        displayTable();
                        system("pause");
                        break;
                    }

                case 4:{
                        //display sorted table
                        system("cls");
                        printf("TABLE BY TIME\n");
                           printf("=============\n");
                        displaySort();
                        system("pause");
                        break;
                }

                case 5:{
                        //display the minimum heap flow (est time)
                        system("cls");
                        printf("MINIMUM HEAP\n");
                        printf("============\n");
                        displayTime();
                        printf("\n");
                        system("pause");
                        break;
                }

                    case 6:{
                        //display tree
                    system("cls");
                    printf("TREE VISUALIZATION\n");
                        printf("==================\n");
                        displayTree(0, 1);
                        system("pause");
                        break;
            }

                    case 0:{
                        printf("\n--- Thank you. ---\n");
                        break;
                    }

                    default:{
                        printf("\n--- Invalid input. Please try again. ---\n\n");
                        system("pause");
                        break;
                    }
            }
        } while(menu != 0);
        return 0;
}
```

BAB 5

DAFTAR PUSTAKA

Abdi, H. (2022). *Array adalah Tipe Data Terstruktur untuk Menyimpan Data Bertipe Sama, Kenali Jenisnya*. liputan6.com. Retrieved 17 June 2022, from https://hot.liputan6.com/read/4870896/array-adalah-tipe-data-terstruktur-untuk-menyimpan-data-bertipe-sama-kenali-jenisnya.

Binary Heap - GeeksforGeeks. GeeksforGeeks. (2022). Retrieved 18 June 2022, from https://www.geeksforgeeks.org/binary-heap/.

C++ Structures (struct). W3schools.com. (2022). Retrieved 17 June 2022, from https://www.w3schools.com/cpp/cpp_structs.asp#:~:text=Structures%20(also%20called%20structs)%20are,%2C%20bool%2C%20et.

Heap Data Structure. Programiz.com. (2022). Retrieved 18 June 2022, from https://www.programiz.com/dsa/heap-data-structure

Priority Queue Data Structure. Programiz.com. (2022). Retrieved 18 June 2022, from https://www.programiz.com/dsa/priority-queue.

Queue | Set 1 (Introduction and Array Implementation) - GeeksforGeeks. GeeksforGeeks. (2022). Retrieved 19 June 2022, from https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/.

switch...case in C Programming. Programiz.com. (2022). Retrieved 18 June 2022, from https://www.programiz.com/c-programming/c-switch-case-statement.

LEMBAR PENILAIAN

Priority Queue for Food Serving Based on the Estimated Time Using Minimum Heap Tree

MATA KULIAH COMP3632004 – DATA STRUCTURES

KELAS BC20

Semester Genap, 2021/2022

DAFTAR MAHASISWA	**NILAI**				**BOBOT**				**KREDIT**				**TOTAL KREDIT**
	1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	
2502012546 - Keyla Azzahra					20 %	30 %	30 %	20 %					
2502020390 - Theodorus Austin Valenzio					20 %	30 %	30 %	20 %					
2501966143 - Vanessa Danuwijaya					20 %	30 %	30 %	20 %					
TOTAL													

KETERANGAN:

- **Skala Penilaian: 0 s/d 100**
- **Komponen**
    1. Laporan
    2. Produk
    3. Pengetahuan dan Solusi
    4. Presentasi

Malang,

(Elizabeth Paskahlia Gunawan, S. Kom., M.Cs.)

D5707

PEMBAGIAN TUGAS

1. Keyla Azzahra
    - *Coding*

      i. Membuat dan mengembangkan base code (void insert, swap, delRoot, displayTime)

    - Laporan

      i. Menulis latar belakang

      ii. Menulis tinjauan pustaka

      iii. Menulis *pseudocode*

      iv. Menulis hasil

2. Theodorus Austin Valenzio
    - *Coding*

      i. Menambahkan fitur sorting menggunakan metode bubble sort

      ii. Menambahkan fitur tree visualization

    - Laporan

      i. Menulis detail program

      ii. Menulis program overview

      iii. Menulis hasil

3. Vanessa Danuwijaya
    - *Coding*

      i. Membuat dan mengembangkan base code (void swapChar, display, displayTable, insert, upheap)

    - Laporan

      i. Menulis tinjauan pustaka

      ii. Membuat *flowchart*

      iii. Menulis program overview

      iv. Menulis layout design

      v. Menulis hasil