**Department of Computing**

**School of Electrical Engineering and Computer Science**

**CS250 – Data Structures and Algorithms**



**Lab 9: Implementation of Binary Heap (Min-Heap)**

**Submission Details**

|  |  |
| --- | --- |
| Name | CMS ID |
| Muhammad Umer | 345834 |
| Group | GP – 1 |
| Lab Engineer | Anum Asif |
| Faculty Member | Bostan Khan |
| Class | BEE12 |
| Date | 08/04/2024 |
| Time | 10:00 am – 12:50 pm |

**Table of Contents**

[2 Implementation of Binary Heap (Min-Heap) 3](#_Toc163483786)

[2.1 Introduction 3](#_Toc163483787)

[2.2 Objectives 3](#_Toc163483788)

[2.3 Tools/Software Requirement 3](#_Toc163483789)

[2.4 Deliverables 3](#_Toc163483790)

[2.5 Description 3](#_Toc163483791)

[3 Lab Task 5](#_Toc163483792)

# Implementation of Binary Heap (Min-Heap)

## Introduction

This lab is based on the Binary Heap (min-heap).

## Objectives

The objective of this lab is to implement binary heap (min-heap).

## Tools/Software Requirement

* Visual Studio C++

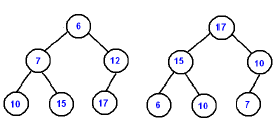
## Deliverables

Compile a single word document by filling in the solution parts and submit this file on LMS. The name of word document should follow this format. i.e., YourFullName(reg)\_Lab#. You must show the implementation of the tasks in a complete manner to get your work graded.

## Description

A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:

1. ***min-heap property*:** the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
2. ***max-heap property*:** the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

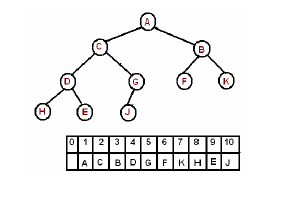


In a heap the highest (or lowest) priority element is always stored at the root, hence the name "heap". A heap is not a sorted structure and can be regarded as partially ordered.

A heap is a useful data structure when you need to remove the object with the highest (or lowest) priority. A common use of a heap is to implement a priority queue.

**Array Implementation**

A complete binary tree can be uniquely represented by storing its level order traversal in an array.

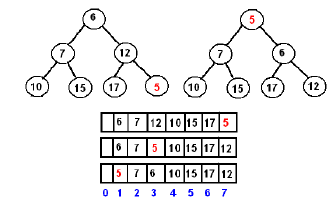


The root is the second item in the array. We skip the index zero cell of the array for the convenience of implementation. Consider k-th element of the array,

* its left child is located at 2\*k index
* its right child is located at 2\*k+1. index
* its parent is located at k/2 index

**Insert**

The new element is initially appended to the end of the heap (as the last element of the array). The heap property is repaired by comparing the added element with its parent and moving the added element up a level (swapping positions with the parent).



**Delete Min**

The minimum element can be found at the root, which is the first element of the array. We remove the root and replace it with the last element of the heap and then restore the heap property

# Lab Task

Implement a binary heap using an array & implement the following functions.

**Top** - returns min element without removing it from the heap

**Pop** - Make sure that the tree remains a complete binary tree.

**Push** – Insert a number into the heap and make sure the heap maintains its   
 key property

**isEmpty** – return TRUE if the heap is empty

**size** - returns the number of elements in the heap

**height** - returns the height of the tree

**buildHeap(array)** - converts any array of numbers into a heap.

**print** - prints the tree

**Important Note:** Practice your knowledge of OOP with C++ when creating a solution. Remember to comment your code properly. Inappropriate or no comment may result in deduction of marks.

**Solution**

#include <math.h>

#include <iostream>

using *namespace* std;

*void* swap(*int* *&a*, *int* *&b*) {

*int* temp = *a*;

*a* = *b*;

*b* = temp;

}

*class* Heap {

*public:*

*int* values[30];

*int* size;

*public:*

    Heap() {

        size = 0;

    }

*bool* isEmpty() {

        return size == 0;

    }

*bool* isFull() {

        return size == 30;

    }

*int* top() {

        if (isEmpty()) {

            cout << "Heap is empty!\n";

            return -1;

        } else {

            return values[1];

        }

    }

*void* push(*int* *value*) {

        if (isFull()) {

            cout << "Heap is full!\n";

            return;

        }

        size++;

        values[size] = *value*;

    }

*void* heapify(*int* *index*) {

*int* smallest = *index*;

*int* left = 2 \* *index*;

*int* right = 2 \* *index* + 1;

        if (left <= size && values[left] < values[smallest])

            smallest = left;

        if (right <= size && values[right] < values[smallest])

            smallest = right;

*// if parent is not smallest*

        if (smallest != *index*) {

            swap(values[*index*], values[smallest]);

            heapify(smallest);

        }

    }

*void* buildHeap() {

        for (*int* i = size / 2; i >= 1; i--) {

            heapify(i);

        }

    }

*void* buildHeapFromArray(*int* *\*arr*, *int* *size*) {

        for (*int* i = 0; i < *size*; i++)

            push(*arr*[i]);

        buildHeap();

    }

*void* deleteMin() {

        if (isEmpty()) {

            cout << "Heap is empty!\n";

            return;

        }

        values[1] = values[size];

        size--;

        buildHeap();

    }

*int* pop(*int* *index*) {

        if (*index* < 1 || *index* > size) {

            cout << "Invalid index!\n";

            return -1;

        } else {

*int* value = values[*index*];

            values[*index*] = values[size];

            size--;

            buildHeap();

            return value;

        }

    }

*void* print() {

        if (isEmpty()) {

            cout << "Heap is empty!\n";

            return;

        }

        for (*int* i = 1; i <= size; i++) {

            cout << values[i] << " ";

            if (i == 1 || i == 3 || i == 7 || i == 15)

                cout << endl;

        }

    }

*int* getSize() {

        return size;

    }

*int* getHeight() {

        return log2(size);

    }

};

*void* displayMenu() {

    cout << "\*\*== Heap Operations ==\*\*\n";

    cout << "1) -> Insert a value\n";

    cout << "2) -> Delete the minimum value\n";

    cout << "3) -> Get the minimum value\n";

    cout << "4) -> Get the size of the heap\n";

    cout << "5) -> Get the height of the heap\n";

    cout << "6) -> Pop a value\n";

    cout << "7) -> Print the heap\n";

    cout << "8) -> Exit\n";

}

*int* main() {

    Heap myHeap = Heap();

*const* *int* arraySize = 15;

*int* array[arraySize];

    for (*int* i = 0; i < arraySize; i++)

        array[i] = (rand() % 100) + 1;

    myHeap.buildHeapFromArray(array, arraySize);

*// create a Menu for all functions*

*int* choice;

    displayMenu();

    while (true) {

        cout << "\nEnter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1: {

*int* value;

                cout << "Enter the value to insert: ";

                cin >> value;

                myHeap.push(value);

                break;

            }

            case 2: {

                myHeap.deleteMin();

                break;

            }

            case 3: {

                cout << "Minimum value: " << myHeap.top() << endl;

                break;

            }

            case 4: {

                cout << "Size of the heap: " << myHeap.getSize() << endl;

                break;

            }

            case 5: {

                cout << "Height of the heap: " << myHeap.getHeight() << endl;

                break;

            }

            case 6: {

*int* index;

                cout << "Enter the index to pop: ";

                cin >> index;

                cout << "Popped value: " << myHeap.pop(index) << endl;

                break;

            }

            case 7: {

                myHeap.print();

                break;

            }

            case 8: {

                cout << "Exiting...\n";

                return 0;

            }

            default: {

                cout << "Invalid choice!\n";

                break;

            }

        }

    }

    getchar();

    return 0;

}

**Output**

root@Zonularity:/home/zonularity/dsa# cd "/home/zonularity/dsa/lab\_9/" && g++ task.cpp -o task && "/home/zonularity/dsa/lab\_9/"task

\*\*== Heap Operations ==\*\*

1) -> Insert a value

2) -> Delete the minimum value

3) -> Get the minimum value

4) -> Get the size of the heap

5) -> Get the height of the heap

6) -> Pop a value

7) -> Print the heap

8) -> Exit

Enter your choice: 7

16

22 28

50 63 36 60

93 87 94 84 78 91 87 64

Enter your choice: 3

Minimum value: 16

Enter your choice: 2

Enter your choice: 7

22

50 28

64 63 36 60

93 87 94 84 78 91 87

Enter your choice: 4

Size of the heap: 14

Enter your choice: 5

Height of the heap: 3

Enter your choice: 6

Enter the index to pop: 4

Popped value: 64

Enter your choice: 7

22

50 28

87 63 36 60

93 87 94 84 78 91

Enter your choice: 8

Exiting...