# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

# Class: BESE 13A

# Lab 11: Implementation of Binary Heap (min heap)

**Date: 8th December, 2023**

**Time: 10 am - 1 pm**

**Lab Engineer: Anum Asif**

# 

# Lab 11: 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++

**Helping Material**

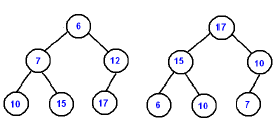
Lecture slides, text book

**Description**

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

 the *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.

 the *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.

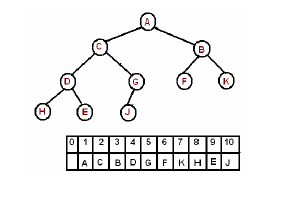


In a heap the highest (or lowest) priority element is always stored at the root, hence the name "heap". A heap is not a sorted structure and can be regarded as partially ordered.

A heap is 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,

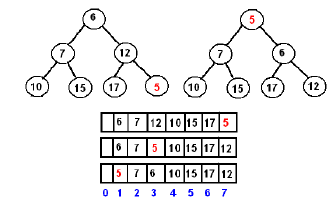
its left child is located at 2\*k index

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 Tasks**

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**

|  |
| --- |
| Solution |
| Lab Task Screen Shots (all functions):    Lab Task Code:  #include <iostream>  #include <vector>  using namespace std;  class BinaryHeap  {  private:      vector<int> heap;      // Helper function to move the pushed and the elements up the heap      void HeapifyUp(int index)      {          while (index > 0)          {              // Calculate the parent index              int parent = (index - 1) / 2;              // Check if the parent is greater, swap if necessary              if (heap[parent] > heap[index])              {                  swap(heap[index], heap[parent]);                  index = parent; // Move up to the parent index              }              else                  break;          }      }      // Helper function to move the element down the heap      void HeapifyDown(int index)      {          int number = index;          int LeftNode = 2 \* index + 1;          int RightNode = 2 \* index + 2;          // Check if the left child exists and is smaller than the current element          if (LeftNode < heap.size() && heap[number] > heap[LeftNode])          {              number = LeftNode;          }          // Check if the right child exists and is smaller than the current element          if (RightNode < heap.size() && heap[number] > heap[RightNode])          {              number = RightNode;          }          // If the index has changed, swap and continue heapifying down          if (number != index)          {              swap(heap[number], heap[index]);              HeapifyDown(number);          }      }  public:      // Function to get the minimum element without removing it      int top()      {          if (!isEmpty())          {              return heap[0];          }          else              cout << "Heap is empty, cannot perform function";          return 0;      }      // Function to check if the heap is empty      bool isEmpty()      {          return heap.empty();      }      // Function to remove and return the minimum element      int pop()      {          if (!isEmpty())          {              int root = heap[0];              heap[0] = heap.back();              heap.pop\_back();              HeapifyDown(0);              return root;          }          else          {              cout << "Heap is empty, cannot perform function";              return 0;          }      }      // Function to get the number of elements in the heap      int size()      {          if (!isEmpty())          {              return heap.size();          }          else              cout << "Heap is empty";          return 0;      }      // Function to get the height of the heap      int height()      {          if (!isEmpty())          {              int levels;              // Loop to find the highest level for which the total nodes are less than the size              for (levels = 0; (1 << levels) - 1 < heap.size(); levels++)                  ;              return levels;          }          else              cout << "The heap is empty";      }      // Function to print the heap      void print()      {          for (auto i : heap)          {              cout << i << ' ';          }      }      // Function to build a heap from an array      void BuildHeap(vector<int> arrary)      {          heap.clear();          for (auto i : arrary)          {              push(i);          }      }      // Function to insert an element into the heap      void push(int i)      {          heap.push\_back(i);          HeapifyUp(heap.size() - 1);      }  };  int main()  {      BinaryHeap minHeap;      // Example usage:      minHeap.push(4);      minHeap.push(2);      minHeap.push(7);      minHeap.push(1);      cout << endl;      cout << "Min Element: " << minHeap.top() << std::endl;      minHeap.print();      minHeap.pop();      cout << endl;      cout << "After Pop:" << std::endl;      minHeap.print();      cout << endl;      cout << "Heap Size: " << minHeap.size() << std::endl;      cout << endl;      cout << "Heap Height: " << minHeap.height() << std::endl;      cout << endl;      vector<int> array = {9, 5, 3, 8, 1};      minHeap.BuildHeap(array);      cout << "After Building Heap from Array:" << std::endl;      minHeap.print();      return 0;  } |

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.