

**Department of Computing**

**CS250: Data Structures and Algorithms**

**Class: BSCS-10C**

**Lab 09:**

**Date: 19th November 2021**

**Time: 9 am – 11:50 am**

# Instructor: Prof. Dr. Faisal Shafait

**Lab 09 : Binary Heap / Heap Sort**

**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**

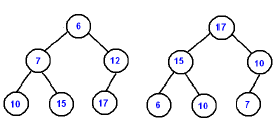
Visual Studio C++

**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.

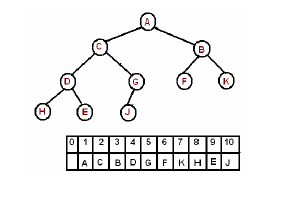


In a heap the highest (or lowest) priority element is always stored at the root, hence the name "heap". A heap is not a sorted structure and can be regarded as partially ordered.

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,

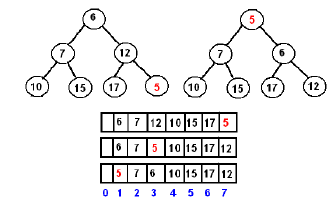
its left child is located at 2\*k index

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

**Code:**

|  |
| --- |
| #include<iostream>  #include<vector>  #include<math.h>  using namespace std;  void swap(int& p, int& q) {  int temp;  temp = p;  p = q;  q = temp;  }  class Heap {  public:  int\* arr;  int capacity;  int size;  Heap(int cap) {  //Constructor to initalize all the value  capacity = cap;  size = 0;  arr = new int[cap];  }  int parent(int i) {  return (i - 1) / 2;  }  void inputArray() {  for (int i = 0; i < size; i++) {  cin >> arr[i];  }  }  void heapSearch(int value) {  //Searching somevalue in the heap  for (int i = 0; i < size; i++) {  if (arr[i] == value) {  cout << "The value" << arr[i] << " is in the heap " << endl;  }  }  cout << "The value " << value << " is not in the heap!" << endl;  }  void printHeap() {  //Printing the heap funciton  for (int i = 0; i < size; i++) {  cout << arr[i] << endl;  }  }  int heightHeep() {  //Because we know the tree grows log wise that's why we use log2  if (size == 0) {  return 0;  }    return ceil(log2(size + 1)) - 1;  }  void minHeap(int index) {    int largest = index;  int leftChild = 2 \* index + 1;  int rightChild = 2 \* index + 2;  if (leftChild< size && arr[leftChild]>arr[largest]) {  largest = leftChild;  }  if (rightChild< size && arr[rightChild]>arr[largest]) {  largest = rightChild;  }  if (largest != index) {  //means if there is any change then swap them else not  swap(arr[index], arr[largest]);  minHeap(largest);  //keep doing this  }  }  void insertKey(int k) {  if (size == capacity) {  cout << "\nOverflow: Could not insert Key\n";  return;  }  // First insert the new key at the end  size++;  int i = size - 1;  arr[i] = k;  // Fix the min heap property if it is violated  while (i != 0 && arr[parent(i)] > arr[i]) {  swap(arr[i], arr[parent(i)]);  i = parent(i);  }  }  void maxHeap(int index) {  int largest = index;  int leftChild = 2 \* index + 1;  int rightChild = 2 \* index + 2;  if (leftChild< size && arr[leftChild]>arr[largest]) {  largest = leftChild;  }  if (rightChild< size && arr[rightChild]>arr[largest]) {  largest = rightChild;  }  if (largest != index) {  //means if there is any change then swap them else not  swap(arr[index], arr[largest]);  maxHeap(largest);  //keep doing this  }    }  int Top() {  if (size <= 0) {  return 0;  }  else {  return arr[1];  }  }    int minPop() {  if (size <= 0) {  return 0;  }  if (size == 1) {    return arr[1];  size--;  }  int root = arr[1];    arr[1] = arr[size - 1];  size--;  minHeap(1);  return root;  }  int maxPop() {  if (size <= 0) {  return 0;  }  if (size == 1) {  return arr[1];  size--;  }  int root = arr[1];  arr[1] = arr[size - 1];  size--;  maxHeap(1);  return root;  }  bool isEmpty() {  if (size == 0) {  return true;  }  else {  return false;  }  }  void heapSort() {  for (int i = 0; i < size; i++) {  cout << minPop() << endl;  }  }  };  void Menu() {    cout << "1-----Height of the tree-------" << endl;  cout << "2-----Your unsorted array-------" << endl;  cout << "3-----Check if it is Empty!-------" << endl;  cout << "4-----Popping out the element-------" << endl;  cout << "5-----Top element-------" << endl;  cout << "6-----Sorted Array by heap sort-------" << endl;  cout << "7-----insert an element in the heap-------" << endl;  cout << "8-----Exit-------" << endl;  }  int main() {  int size,key,choice;  cout << "-----Enter the size of the array-----------" << endl;  cin >> size;  Heap obj(size + 1);    do{  cout << endl;  Menu();    cout << "Enter your choice :" << endl;  cin >> choice;  system("cls");  switch (choice)  {  case 0:  break;  case 1:  cout << "The height of the tree is :" << obj.heightHeep() << endl;  break;  case 2:  cout << "Your unsorted Array is :" << endl;  obj.printHeap();    break;  case 3:  if (obj.isEmpty()) {  cout << "The array is Empty" << endl;    }  else {  cout << "Not Empty!!" << endl;    }  break;  case 4:  cout << "The Popped element is :" << endl;  cout << obj.minPop();  break;  case 5:  cout << "The top element is :" << endl;  cout<<obj.Top();    break;  case 6:  cout << "Your sorted Array is :" << endl;  obj.heapSort();  obj.printHeap();    break;  case 7:  cout << "Insert the element:" << endl;  cin >> key;  obj.insertKey(key);    break;  case 8:  return 0;    break;  default:  cout << "Please Enter some valid value:" << endl;  break;  }  } while (choice!=0);  } |

**Output**

Text

Description automatically generatedText

Description automatically generated

**Deliverables**

Students are required to upload the lab on LMS before deadline.

**Note:** Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks. You will submit your workingcodes in **word document** (do **NOT** take screenshot of code, just copy your code and paste it). The name of word document should follow this format. i.e. **YOUR\_NAME\_Lab#**