Lab #11

Task #1:

#include <iostream>

using namespace std;

class MinHeap

{

private: //attributes

int\* heapArray;

int capacity;

int heapSize;

public:

MinHeap(int capacity)//constructor

{

this->capacity = capacity;

heapArray = new int[capacity];

heapSize = 0;

}

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

int parent(int i)

{

return (i - 1) / 2;

}

int leftChild(int i)

{

return 2 \* i + 1;

}

int rightChild(int i) {

return 2 \* i + 2;

}

int getMini() // it will return the root element

{

if (heapSize == 0) {

cout << "Heap is empty!" << endl;

return -1;

}

return heapArray[0];

}

// Function to extract the minimum element from the heap (root element)

int extractMin()

{

if (heapSize == 0)

{

cout << "Heap is empty!" << endl;

return -1;

}

int root = heapArray[0];

heapArray[0] = heapArray[heapSize - 1];

heapSize--;

minHeapify(0);

return root;

}

// Helper function to maintain the Min Heap property

void minHeapify(int i) {

int smallest = i;

int left = leftChild(i);

int right = rightChild(i);

if (left < heapSize && heapArray[left] < heapArray[smallest]) {

smallest = left;

}

if (right < heapSize && heapArray[right] < heapArray[smallest]) {

smallest = right;

}

if (smallest != i)

{

swap(heapArray[i], heapArray[smallest]);

minHeapify(smallest);

}

}

// Function to insert a new key into the heap

void insert(int key)

{

if (heapSize == capacity) {

cout << "Heap is full!" << endl;

return;

}

heapArray[heapSize] = key;

int i = heapSize;

heapSize++;

while (i > 0 && heapArray[parent(i)] > heapArray[i]) {

swap(heapArray[i], heapArray[parent(i)]);

i = parent(i);

}

}

// Function to print the elements of the heap

void printHeap() {

for (int i = 0; i < heapSize; i++) {

cout << heapArray[i] << " ";

}

cout << endl;

}

};

int main()

{

int size, value;

cout << "enter size of heap = ";

cin >> size;

MinHeap heap(size);

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

{

cout << "enter " << i + 1 << "th value in heap = ";

cin >> value;

heap.insert(value);

}

cout << "values in heap are = " << endl;

heap.printHeap();

cout << "minimum value is = " << endl;

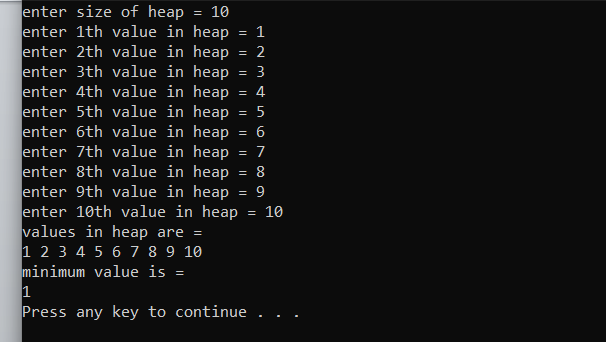
cout << heap.getMini() << endl;

system("pause");

return 0;

}

**Output:**

****

**Task #2:**

#include <iostream>

#include <queue>

using namespace std;

class BuildingHeap

{

private:

priority\_queue<pair<int, int>> maxHeap; // {number of floors, building ID}

int buildingCount;

public:

BuildingHeap() : buildingCount(0) {} //intializing

// Add a building to the construction queue

void addBuilding(int buildingID, int numFloors)

{

maxHeap.push(make\_pair(numFloors, buildingID));

buildingCount++;

}

// Construct the building with the highest priority

void constructBuilding()

{

if (buildingCount == 0)

{

cout << "No buildings in the queue.\n";

return;

}

pair<int, int> topBuilding = maxHeap.top();

maxHeap.pop();

buildingCount--;

cout << "Constructing Building ID: " << topBuilding.second << " with " << topBuilding.first << " floors.\n";

}

// Get the ID of the building with the highest number of floors in the queue

int getHighestPriorityBuilding()

{

if (buildingCount == 0) {

cout << "No buildings in the queue.\n";

return -1; // Assuming -1 indicates no building in the queue

}

return maxHeap.top().second;

}

};

int main() {

BuildingHeap obj;

obj.addBuilding(1, 3);

obj.addBuilding(2, 6);

obj.addBuilding(3, 9);

obj.addBuilding(4, 20);

int highestPrioBuild = obj.getHighestPriorityBuilding();

cout << "Building with the highest priority (ID): " << highestPrioBuild << endl;

obj.constructBuilding();

highestPrioBuild = obj.getHighestPriorityBuilding();

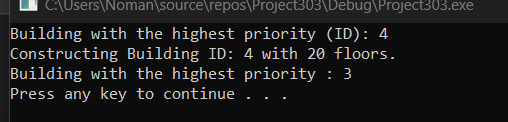
cout << "Building with the highest priority : " << highestPrioBuild << endl;

system("pause");

return 0;

}

Output:



**Task#3:**

#include<iostream>

using namespace std;

void Heap(int Arr[], int n, int data)

{

int smallest = data;

int Left = 2 \* data + 1;

int Right = 2 \* data + 2;

if (Left < n && Arr[Left] < Arr[smallest])

{

smallest = Left;

}

if (Right < n && Arr[Right] < Arr[smallest])

{

smallest = Right;

}

if (smallest != data)

{

swap(Arr[data], Arr[smallest]);

Heap(Arr, n, smallest);

}

}

int minheap(int arr[], int n, int k) // This function will sort the element in the minimum order

{

for (int i = n / 2 - 1; i >= 0; i--)

{

Heap(arr, n, i);

}

for (int i = n - 1; i >= n - k; i--)

{

swap(arr[0], arr[i]);

Heap(arr, i, 0);

}

return arr[n - k];

}

int main()

{

int size = 7;

int arr[] = { 4, 2, 6, 1, 3, 5, 7 };

cout<< "The MIN ElEMENTS IN THE HEAP = ";

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

{

cout << minheap(arr, size, i + 1) << " ";

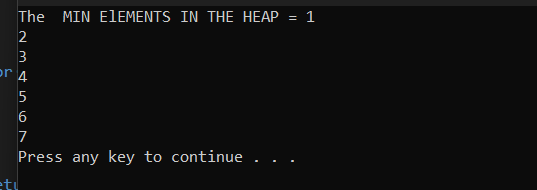
}

system("pause");

return 0;

}

**Output:**

****

**Task#4:**

#include <iostream>

using namespace std;

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

void heapify(int\* arr, int n, int i) // This function will heapify the element

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void buildMaxHeap(int\* arr, int n) //This will return max element first then so on decreasing..

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

}

void printHeap(int\* arr, int n) //Displaying heap element

{

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int size;

cout << "Enter size of heap = ";

cin >> size;

int\* arr = new int[size];

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

{

cout << "Enter " << i + 1 << "th value in heap: " << endl;

cin >> arr[i];

}

int n = size;

buildMaxHeap(arr, n);

cout << "Max Heap: ";

printHeap(arr, n);

system("pause");

return 0;

}

Output:

