Name : Khalid Abbas

Reg # 2312236

Ma’am : Maria Bibi ,

Section : BS-CS-3D

**Source Code:**

#include <iostream>

#include <algorithm> // for std::max

using namespace std;

#define MAX 100 // Maximum size of the heap

class MinMaxHeap {

private:

int heap\_size;

int harr[MAX];

int parent(int i) { return (i - 1) / 2; }

int left(int i) { return (2 \* i + 1); }

int right(int i) { return (2 \* i + 2); }

void Minheapify(int i);

void Maxheapify(int i);

public:

MinMaxHeap() : heap\_size(0) {}

void insert(int k);

int deleteMin();

int deleteMax();

int getMin();

int getMax();

void buildHeap(int arr[], int n);

void display();

};

// Inserts an element into the Min-Max Heap

void MinMaxHeap::insert(int k) {

if (heap\_size == MAX) {

cout << "Overflow: could not insert key\n";

return;

}

heap\_size++;

int i = heap\_size - 1;

harr[i] = k;

// Alternate between min and max insertion

if (i % 2 == 0) { // Even level (min level)

while (i != 0 && harr[parent(i)] > harr[i]) {

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

i = parent(i);

}

} else { // Odd level (max level)

while (i != 0 && harr[parent(i)] < harr[i]) {

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

i = parent(i);

}

}

}

// Maintains the min-heap property

void MinMaxHeap::Minheapify(int i) {

int smallest = i;

int l = left(i);

int r = right(i);

if (l < heap\_size && harr[l] < harr[smallest])

smallest = l;

if (r < heap\_size && harr[r] < harr[smallest])

smallest = r;

if (smallest != i) {

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

Minheapify(smallest);

}

}

// Maintains the max-heap property

void MinMaxHeap::Maxheapify(int i) {

int largest = i;

int l = left(i);

int r = right(i);

if (l < heap\_size && harr[l] > harr[largest])

largest = l;

if (r < heap\_size && harr[r] > harr[largest])

largest = r;

if (largest != i) {

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

Maxheapify(largest);

}

}

// Returns the minimum element (root of the heap)

int MinMaxHeap::getMin() {

if (heap\_size > 0) {

return harr[0]; // Minimum is at the root

}

return -1; // Indicate that the heap is empty

}

// Returns the maximum element (among the first two children)

int MinMaxHeap::getMax() {

if (heap\_size == 0) {

return -1; // The heap is empty

}

if (heap\_size == 1) {

return harr[0]; // Only one element

}

return std::max(harr[1], harr[2]); // Maximum is among the first

two children

}

// Deletes the minimum element and restores the heap property

int MinMaxHeap::deleteMin() {

if (heap\_size == 0) return -1;

int root = harr[0];

harr[0] = harr[heap\_size - 1];

heap\_size--;

Minheapify(0); // Restore min-heap property

return root;

}

// Deletes the maximum element and restores the heap property

int MinMaxHeap::deleteMax() {

if (heap\_size == 0) return -1;

if (heap\_size == 1) {

heap\_size--;

return harr[0]; // Only one element

}

// Find the max among children of the root

int maxIndex = (harr[1] > harr[2]) ? 1 : 2;

int maxElement = harr[maxIndex];

harr[maxIndex] = harr[heap\_size - 1];

heap\_size--;

Maxheapify(maxIndex); // Restore max-heap property

return maxElement;

}

// Builds a Min-Max Heap from an array

void MinMaxHeap::buildHeap(int arr[], int n) {

heap\_size = n;

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

harr[i] = arr[i];

}

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

if (i % 2 == 0) {

Minheapify(i);

} else {

Maxheapify(i);

}

}

}

// Displays the elements of the heap

void MinMaxHeap::display() {

for (int i = 0; i < heap\_size; i++) {

cout << harr[i] << " ";

}

cout << endl;

}

int main() {

MinMaxHeap heap;

// Elements from the binary tree

int elements[] = {6, 81, 87, 14, 17, 12, 28, 71, 25, 80, 20, 52, 78,

31, 42, 31, 59, 16, 24, 79, 63, 18, 19, 32, 13, 15, 48};

int n = sizeof(elements) / sizeof(elements[0]);

// Insert elements into the heap

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

heap.insert(elements[i]);

}

// Display the heap

cout << "Heap elements: ";

heap.display();

// Display min and max values

cout << "Min: " << heap.getMin() << endl;

cout << "Max: " << heap.getMax() << endl;

// Perform delete operations

cout << heap.deleteMin() << " deleted (min)." << endl;

cout << heap.deleteMax() << " deleted (max)." << endl;

// Display the heap after deletions

cout << "Heap elements after deletions: ";

heap.display();

return 0;

}

//CodeWithCarlito

//The\_End

**Output :**

