Q-1: Write a C++ program to insert new element to MAX-Heap.

Sample test case:

|  |
| --- |
| Input: heap={ 10, 5, 3, 2, 4 }  key=15  Output: 15 5 10 2 4 3 |

Solution:

#include <iostream>

using namespace std;

#define MAX 1000 // Max size of Heap

// Function to heapify ith node in a Heap

// of size n following a Bottom-up approach

void heapify(int arr[], int n, int i)

{

// Find parent

int parent = (i - 1) / 2;

if (arr[parent] > 0) {

// For Max-Heap

// If current node is greater than its parent

// Swap both of them and call heapify again

// for the parent

if (arr[i] > arr[parent]) {

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

// Recursively heapify the parent node

heapify(arr, n, parent);

}

}

}

// Function to insert a new node to the Heap

void insertNode(int arr[], int& n, int Key)

{

// Increase the size of Heap by 1

n = n + 1;

// Insert the element at end of Heap

arr[n - 1] = Key;

// Heapify the new node following a

// Bottom-up approach

heapify(arr, n, n - 1);

}

// A utility function to print array of size n

void printArray(int arr[], int n)

{

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

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

cout << "\n";

}

int main()

{

// Array representation of Max-Heap

// 10

// / \

// 5 3

// / \

// 2 4

int arr[MAX] = { 10, 5, 3, 2, 4 };

int n = 5;

int key = 15;

insertNode(arr, n, key);

printArray(arr, n);

// Final Heap will be:

// 15

// / \

// 5 10

// / \ /

// 2 4 3

return 0;

}

Q-2: You need to implement a program that inserts elements into a min heap and displays the heap after each insertion

Sample test case:

|  |
| --- |
| Input: heap\_elements={15, 10, 20, 8, 25, 13}  Output:  Inserted: 15, Heap: 15  Inserted: 10, Heap: 10 15  Inserted: 20, Heap: 10 15 20  Inserted: 8, Heap: 8 10 20 15  Inserted: 25, Heap: 8 10 20 15 25  Inserted: 13, Heap: 8 10 13 15 25 20 |

Solution:

#include <iostream>

#include <vector>

// Function to maintain the min heap property after inserting an element

void heapifyUp(std::vector<int>& heap, int index) {

while (index > 0 && heap[index] < heap[(index - 1) / 2]) {

std::swap(heap[index], heap[(index - 1) / 2]);

index = (index - 1) / 2;

}

}

// Function to insert an element into the min heap

void insertMinHeap(std::vector<int>& heap, int value) {

heap.push\_back(value);

heapifyUp(heap, heap.size() - 1);

}

int main() {

std::vector<int> minHeap;

int elements[] = {50,30,80,60,70,10,20};

for (int element : elements) {

insertMinHeap(minHeap, element);

std::cout << "Inserted: " << element << ", Heap: ";

for (int value : minHeap) {

std::cout << value << " ";

}

std::cout << std::endl;

}

return 0;

}

Q-3: You are given an array of integers. Implement a program to find the kth smallest element in the array using a min-heap.

Sample test case:

|  |
| --- |
| Input: arr = {10, 50, 40, 75, 60, 65, 45} ,k=5  Output: 60 |

Solution:

#include <bits/stdc++.h>

using namespace std;

// Structure for the heap

struct Heap {

vector<int> v;

int n; // Size of the heap

Heap(int i = 0)

: n(i)

{

v = vector<int>(n);

}

};

// Generic function to

// swap two integers

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

// Returns the index of

// the parent node

inline int parent(int i)

{

return (i - 1) / 2;

}

// Returns the index of

// the left child node

inline int left(int i)

{

return 2 \* i + 1;

}

// Returns the index of

// the right child node

inline int right(int i)

{

return 2 \* i + 2;

}

// Maintains the heap property

void heapify(Heap& h, int i)

{

int l = left(i), r = right(i), m = i;

if (l < h.n && h.v[i] > h.v[l])

m = l;

if (r < h.n && h.v[m] > h.v[r])

m = r;

if (m != i) {

swap(h.v[m], h.v[i]);

heapify(h, m);

}

}

// Extracts the minimum element

int extractMin(Heap& h)

{

if (!h.n)

return -1;

int m = h.v[0];

h.v[0] = h.v[h.n-- - 1];

heapify(h, 0);

return m;

}

int findKthSmalles(Heap &h, int k)

{

for (int i = 1; i < k; ++i)

extractMin(h);

return extractMin(h);

}

int main()

{

Heap h(7);

h.v = vector<int>{ 10, 50, 40, 75, 60, 65, 45 };

int k = 5;

cout << findKthSmalles(h, k);

return 0;

}