Q-1: Implement a C++ program for Heap Sort, a comparison-based sorting algorithm that utilizes a binary heap data structure to arrange

elements in ascending order.

Sample test case:

|  |
| --- |
| Input: arr={ 60 ,20 ,40 ,70, 30, 10}  Output:  After heapifying array is  70 60 40 20 30 10  Sorted array is  10 20 30 40 60 70 |

Solution:

#include <iostream>

using namespace std;

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

{

int largest = i; // Initialize largest as root Since we are using 0 based indexing

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

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

largest = l;

// If right child is larger than largest so far

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

largest = r;

// If largest is not root

if (largest != i) {

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

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

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

heapify(arr, n, i);

// One by one extract an element from heap

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

// Move current root to end

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

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* 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()

{

int arr[] = { 60 ,20 ,40 ,70, 30, 10};

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

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

heapify(arr,n,i);

}

cout << "After heapifying array is \n";

printArray(arr, n);

heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

return 0;

}

Q-2: Write a C++ program for deleting root element in MAX-Heaps

Sample test case:

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

Solution:

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index of arr[] and n is the size of heap

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

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

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

largest = l;

// If right child is larger than largest so far

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

largest = r;

// If largest is not root

if (largest != i) {

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

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// Function to delete the root from Heap

void deleteRoot(int arr[], int& n)

{

// Get the last element

int lastElement = arr[n - 1];

// Replace root with last element

arr[0] = lastElement;

// Decrease size of heap by 1

n = n - 1;

// heapify the root node

heapify(arr, n, 0);

}

/\* 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[] = { 10, 5, 3, 2, 4 };

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

deleteRoot(arr, n);

printArray(arr, n);

return 0;

}

Q-3: Implement a C++ program to delete an element from a Min-Heap and maintain the Min-Heap property.

Sample test case:

|  |
| --- |
| Input: min-heap{ 1,5,3,8,9,7,6 }  Output:  Min-Heap before deletion: 1 5 3 8 9 7 6  Min-Heap after deleting element at index 2: 1 5 6 8 9 7 |

Solution:

#include <iostream>

using namespace std;

// Function to perform heapify on a Min-Heap

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

int smallest = i; // Initialize smallest as root

int l = 2 \* i + 1; // Left child

int r = 2 \* i + 2; // Right child

// If left child is smaller than root

if (l < n && arr[l] < arr[smallest])

smallest = l;

// If right child is smaller than the smallest so far

if (r < n && arr[r] < arr[smallest])

smallest = r;

// If smallest is not root

if (smallest != i) {

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

// Recursively heapify the affected sub-tree

heapify(arr, n, smallest);

}

}

// Function to delete the element at index 'i' from Min-Heap

void deleteElement(int arr[], int& n, int i) {

if (i >= n)

return;

// Replace the element to be deleted with the last element

arr[i] = arr[n - 1];

n = n - 1;

// Heapify the tree from the updated element position

int parent = (i - 1) / 2;

if (i > 0 && arr[i] < arr[parent]) {

while (i > 0 && arr[i] < arr[parent]) {

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

i = parent;

parent = (i - 1) / 2;

}

} else {

heapify(arr, n, i);

}

}

/\* 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() {

int arr[] = { 1,5,3,8,9,7,6 };

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

cout << "Min-Heap before deletion: ";

printArray(arr, n);

// Deleting the element at index 2 (value 3)

deleteElement(arr, n, 2);

cout << "Min-Heap after deleting element at index 2: ";

printArray(arr, n);

return 0;

}