**Algorithms:**

**1.**

**Build Min Heap:**

**Create a min heap data structure.**

**Insert Elements:**

**Insert all elements from the input array into the min heap.**

**Extract Min Element:**

**Extract the minimum element from the min heap (this will be the largest element in the original array).**

**Replace Array Element:**

**Replace the last element in the array with the extracted minimum element.**

**Heapify:**

**Perform heapify operation to maintain the min heap property.**

**Repeat:**

**Repeat steps 3-5 for the remaining elements in the min heap.**

**Result:**

**The array is now sorted in decreasing order.**

**2.**

**Initialize MinPriorityQueue Class:**

**Create a class MinPriorityQueue with a private member vector heap.**

**Heapify Function:**

**Define a private function heapify(i) to maintain the min-heap property.**

**Constructor:**

**Define a constructor for the MinPriorityQueue class.**

**HeapMinimum Function (HEAP-MINIMUM):**

**Define a public function heapMinimum() to return the element with the smallest key.**

**HeapExtractMin Function (HEAP-EXTRACT-MIN):**

**Define a public function heapExtractMin() to remove and return the element with the smallest key.**

**HeapDecreaseKey Function (HEAP-DECREASE-KEY):**

**Define a public function heapDecreaseKey(i, newValue) to decrease the value of an element at index i to a new value newValue.**

**MinHeapInsert Function (MIN-HEAP-INSERT):**

**Define a public function minHeapInsert(key) to insert a new element into the min heap.**

**Main Function:**

**In the main() function:**

**a. Create an instance of MinPriorityQueue called minQueue.**

**b. Read the number of elements to insert (numElements) from the user.**

**c. Insert the specified elements into the min heap using minHeapInsert.**

**d. Print the minimum element in the heap using heapMinimum.**

**e. Extract and print the minimum element from the heap using heapExtractMin.**

**f. Decrease the value of an element at index 2 to 0 using heapDecreaseKey.**

**g. Print the new minimum element in the heap using heapMinimum.**

**3.**

**Define findLargest Function:**

**Create a function findLargest that takes a vector of integers arr as input and returns the largest number in the array.**

**Check for Empty Array:**

**If the array arr is empty, return a default value (e.g., 0) indicating that the array is empty.**

**Initialize Largest Variable:**

**Initialize a variable largest with the first element of the array (arr[0]).**

**Iterate Through Array:**

**For each element arr[i] from the second element to the last element in the array (arr.size()):**

**a. If arr[i] is greater than the current largest, update largest to be arr[i].**

**Return Largest:**

**Return the final value of largest.**

**Main Function:**

**In the main() function:**

**a. Read the number of elements (n) from the user.**

**b. Create a vector nums of size n to store the elements.**

**c. Read the elements from the user and store them in the vector nums.**

**d. Call the findLargest function with the vector nums to find the largest element.**

**e. Print the result.**

**4.**

**Define Node Structure:**

**Create a structure Node representing a node of the Binary Search Tree (BST) with fields data, left, and right.**

**Define getNode Function:**

**Create a function getNode(data) to allocate a new node with the given data and return a pointer to it.**

**Define Inorder Traversal Function:**

**Create a function inorderTraversal(root, arr) to perform an inorder traversal of the BST and store the elements in a vector arr.**

**Define BSTToMinHeap Function:**

**Create a function BSTToMinHeap(root, arr, i) to convert the BST to a Min Heap using preorder traversal and the elements in the vector arr.**

**Define convertToMinHeapUtil Function:**

**Create a function convertToMinHeapUtil(root) to initiate the conversion process by first obtaining the inorder traversal of the BST and then converting the BST to a Min Heap.**

**Define Preorder Traversal Function:**

**Create a function preorderTraversal(root) to perform a preorder traversal of the tree and print the elements.**

**Main Function:**

**In the main() function:**

**a. Create a BST with sample data.**

**b. Call convertToMinHeapUtil(root) to convert the BST to a Min Heap.**

**c. Print the Preorder Traversal of the converted Min Heap.**

**Codes:**

**1.**

**Code:**

#include <bits/stdc++.h>

using namespace std;

// Function to heap sort in decreasing order using min heap

void heapSort(int arr[], int n){

// Creating min heap using a priority queue

priority\_queue<int, vector<int>, greater<int> > minHeap;

// Inserting input array to min heap

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

minHeap.push(arr[i]);

}

// Iterating backwards in the input array, where each element is replaced by the smallest element extracted from min heap

int i = n - 1;

while (!minHeap.empty()){

arr[i--] = minHeap.top();

minHeap.pop();

}

}

int main(){

int arr[6] = {5, 2, 9, 1, 5, 6};

int n = 6;

heapSort(arr, n);

cout << "Sorted array : ";

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

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

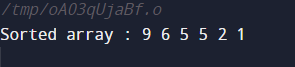
}

cout << endl;

return 0;

}

**Output:**



2.

#include <iostream>

#include <vector>

#include <limits>

using namespace std;

class MinPriorityQueue {

private:

vector<int> heap;

// the min-heap property

void heapify(int i) {

int left = 2 \* i + 1;

int right = 2 \* i + 2;

int smallest = i;

if (left < heap.size() && heap[left] < heap[smallest]) {

smallest = left;

}

if (right < heap.size() && heap[right] < heap[smallest]) {

smallest = right;

}

if (smallest != i) {

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

heapify(smallest);

}

}

public:

// Constructor

MinPriorityQueue() {}

// Function to get the element with the smallest key (HEAP-MINIMUM)

int heapMinimum() {

if (!heap.empty()) {

return heap[0];

} else {

cerr << "Heap is empty." << endl;

return numeric\_limits<int>::max(); // Return a large value to indicate an empty heap

}

}

// Function to remove and return the element with the smallest key (HEAP-EXTRACT-MIN)

int heapExtractMin() {

if (heap.empty()) {

cerr << "Heap underflow." << endl;

return numeric\_limits<int>::max(); }

int minElement = heap[0];

heap[0] = heap.back();

heap.pop\_back();

heapify(0);

return minElement;

}

// Function to decrease the value of the element to a new value (HEAP-DECREASE-KEY)

void heapDecreaseKey(int i, int newValue) {

if (i < heap.size() && newValue < heap[i]) {

heap[i] = newValue;

// Fix the min-heap property

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

swap(heap[i], heap[(i - 1) / 2]);

i = (i - 1) / 2;

}

} else {

cerr << "Invalid index or new value is not smaller." << endl;

}

}

// Function to insert the element (MIN-HEAP-INSERT)

void minHeapInsert(int key) {

heap.push\_back(numeric\_limits<int>::max()); // Insert a large value initially

heapDecreaseKey(heap.size() - 1, key);

}

};

int main() {

MinPriorityQueue minQueue;

int numElements;

cout << "Enter the number of elements to insert: ";

cin >> numElements;

cout << "Enter the elements to insert: ";

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

int element;

cin >> element;

minQueue.minHeapInsert(element);

}

cout << "Heap Minimum: " << minQueue.heapMinimum() << endl;

int extractedMin = minQueue.heapExtractMin();

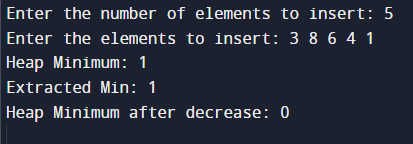
cout << "Extracted Min: " << extractedMin << endl;

minQueue.heapDecreaseKey(2, 0);

cout << "Heap Minimum after decrease: " << minQueue.heapMinimum() << endl;

return 0;

}



3.

**Code:**

#include <iostream>

#include <vector>

// Function to find the largest number in an unsorted array

int findLargest(const std::vector<int>& arr) {

if (arr.empty()) {

return 0; // Return default value for empty array

}

int largest = arr[0];

for (size\_t i = 1; i < arr.size(); i++) {

if (arr[i] > largest) {

largest = arr[i];

}

}

return largest;

}

int main() {

std::cout << "Enter the number of elements: ";

int n;

std::cin >> n;

std::vector<int> nums(n);

std::cout << "Enter the elements of the array: ";

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

std::cin >> nums[i];

}

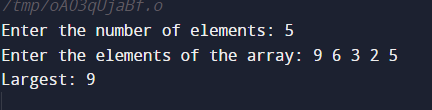
int largest = findLargest(nums);

std::cout << "Largest: " << largest << std::endl;

return 0;

}

**Output:**



4.

#include <bits/stdc++.h>

using namespace std;

// Structure of a node of BST

struct Node {

int data;

Node \*left, \*right;

};

struct Node\* getNode(int data)

{

struct Node\* newNode = new Node;

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

// function prototype for preorder traversal

void preorderTraversal(Node\*);

// function for the inorder traversal of the tree

void inorderTraversal(Node\* root, vector<int>& arr)

{

if (root == NULL)

return;

inorderTraversal(root->left, arr);

arr.push\_back(root->data);

inorderTraversal(root->right, arr);

}

// function to convert the given BST to MIN HEAP performs preorder traversal of the tree

void BSTToMinHeap(Node\* root, vector<int> arr, int\* i)

{

if (root == NULL)

return;

// first copy data at index 'i' of 'arr' to

// the node

root->data = arr[++\*i];

// then recur on left subtree

BSTToMinHeap(root->left, arr, i);

// now recur on right subtree

BSTToMinHeap(root->right, arr, i);

}

// utility function to convert the given BST to MIN HEAP

void convertToMinHeapUtil(Node\* root)

{

vector<int> arr;

int i = -1;

inorderTraversal(root, arr);

BSTToMinHeap(root, arr, &i);

}

// function for the preorder traversal of the tree

void preorderTraversal(Node\* root)

{

if (!root)

return;

cout << root->data << " ";

preorderTraversal(root->left);

preorderTraversal(root->right);

}

int main()

{

struct Node\* root = getNode(4);

root->left = getNode(2);

root->right = getNode(6);

root->left->left = getNode(1);

root->left->right = getNode(3);

root->right->left = getNode(5);

root->right->right = getNode(7);

convertToMinHeapUtil(root);

cout << "Preorder Traversal:" << endl;

preorderTraversal(root);

return 0;

}

**Output:**

