Assignment-2

Prabhnoor
Singh
102115059
3Nc3

Q1 Implement the Heapsort algorithm to arrange numbers in descending order.

Ans-
```
#include <iostream>
#include <vector>

void heapify(std::vector<int>& arr, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        std::swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(std::vector<int>& arr) {
    int n = arr.size();


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


    for (int i = n - 1; i > 0; i--) {
        std::swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    std::vector<int> numbers = {12, 11, 13, 5, 6, 7};

    std::cout << "Original array: ";
    for (int num : numbers) {
        std::cout << num << " ";
    }

    heapSort(numbers);

    std::cout << "\nArray sorted in descending order: ";
    for (int num : numbers) {
```

```
        std::cout << num << " ";
    }

    return 0;
}
```

Q2 Implement a min-priority queue with a min-heap. The program should have functions for the following operations:
HEAP-MINIMUM to get the element with the smallest key,
HEAP-EXTRACT-MIN to remove and return the element with the smallest key,
HEAP-DECREASE-KEY decreases the value of the element to a new value, and
MIN-HEAP-INSERT to insert the element.

Ans
```
#include <iostream>
#include <vector>
#include <limits>

class MinPriorityQueue {
private:
    std::vector<int> heap;

    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 heapifyUp(int i) {
        while (i > 0 && heap[parent(i)] > heap[i]) {
            std::swap(heap[i], heap[parent(i)]);
            i = parent(i);
        }
    }

    void heapifyDown(int i) {
        int l = left(i);
        int r = right(i);
        int smallest = i;

        if (l < heap.size() && heap[l] < heap[i]) {
            smallest = l;
        }

        if (r < heap.size() && heap[r] < heap[smallest]) {
            smallest = r;
        }

        if (smallest != i) {
```

```
            std::swap(heap[i], heap[smallest]);
            heapifyDown(smallest);
        }
    }

public:
    int heapMinimum() {
        if (!heap.empty()) {
            return heap[0];
        } else {
            std::cerr << "Heap is empty.\n";
            return std::numeric_limits<int>::max(); // return maximum value for simplicity
        }
    }

    int heapExtractMin() {
        if (heap.empty()) {
            std::cerr << "Heap underflow.\n";
            return std::numeric_limits<int>::max(); // return maximum value for simplicity
        }

        int min = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);
        return min;
    }

    void heapDecreaseKey(int i, int newKey) {
        if (i >= heap.size() || heap[i] <= newKey) {
            std::cerr << "Invalid index or new key is not smaller.\n";
            return;
        }

        heap[i] = newKey;
        heapifyUp(i);
    }

    void minHeapInsert(int key) {
        heap.push_back(std::numeric_limits<int>::max()); // initialize with positive infinity
        heapDecreaseKey(heap.size() - 1, key);
    }
};

int main() {
    MinPriorityQueue minQueue;

    minQueue.minHeapInsert(4);
    minQueue.minHeapInsert(3);
    minQueue.minHeapInsert(5);
    minQueue.minHeapInsert(2);
    minQueue.minHeapInsert(1);

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

```
    std::cout << "Extracted Min: " << minQueue.heapExtractMin() << std::endl;

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

    minQueue.heapDecreaseKey(2, 1);

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

    return 0;
}
```

Q3 Write a program to find the largest element in an unsorted array.

```
Ans #include <iostream>
#include <vector>

int findLargestElement(const std::vector<int>& arr) {
    if (arr.empty()) {
        std::cerr << "Array is empty.\n";
        return -1;
    }

    int largest = arr[0];

    for (int i = 1; i < arr.size(); ++i) {
        if (arr[i] > largest) {
            largest = arr[i];
        }
    }

    return largest;
}

int main() {
    std::vector<int> numbers = {12, 5, 8, 23, 7, 15, 9, 4};

    int result = findLargestElement(numbers);

    if (result != -1) {
        std::cout << "The largest element in the array is: " << result << std::endl;
    }

    return 0;
}
```

Q4 Write a program to convert a binary search tree into a min-heap

```
Ans #include <iostream>


struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
```

```
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Function to perform in-order traversal of the BST and convert it to a min-heap
void convertBSTtoMinHeap(TreeNode* root, int& index, int* arr) {
    if (root == nullptr) {
        return;
    }


    convertBSTtoMinHeap(root->left, index, arr);


    root->data = arr[index++];


    convertBSTtoMinHeap(root->right, index, arr);
}


void storeBSTInArray(TreeNode* root, int* arr, int& index) {
    if (root == nullptr) {
        return;
    }


    storeBSTInArray(root->left, arr, index);


    arr[index++] = root->data;


    storeBSTInArray(root->right, arr, index);
}

// Function to convert a BST to a min-heap
void convertBSTToMinHeap(TreeNode* root) {
    // Count the number of nodes in the BST
    int nodeCount = 0;
    storeBSTInArray(root, nullptr, nodeCount);


    int* arr = new int[nodeCount];
    int index = 0;
    storeBSTInArray(root, arr, index);

    std::sort(arr, arr + nodeCount);


    index = 0;
    convertBSTtoMinHeap(root, index, arr);


    delete[] arr;
}
```

```
void printInOrder(TreeNode* root) {
    if (root == nullptr) {
        return;
    }


    printInOrder(root->left);


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


    printInOrder(root->right);
}

int main() {

    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(6);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(5);
    root->right->right = new TreeNode(7);

    std::cout << "In-order traversal of the original BST: ";
    printInOrder(root);
    std::cout << std::endl;

    // Convert the BST to a min-heap
    convertBSTToMinHeap(root);

    std::cout << "In-order traversal of the BST after conversion to min-heap: ";
    printInOrder(root);
    std::cout << std::endl;


    delete root->left->left;
    delete root->left->right;
    delete  root->left;
    delete root->right->left;
    delete root->right->right;
    delete  root->right;
    delete root;

    return 0;
}
```