Batch:T6

Practical No.1

Title of Assignment: Sorting algorithm

Student Name: Parshwa Herwade

Student PRN: 22510064

1) You are given two sorted array, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order

Pseudocode:

```
function mergeSortedArrays(A, B, m, n):
    i = m - 1
    j = n - 1
    k = m + n - 1

    while i >= 0 and j >= 0:
        if A[i] > B[j]:
            A[k] = A[i]
            i = i - 1
        else:
            A[k] = B[j]
            j = j - 1
        k = k - 1

    while j >= 0:
        A[k] = B[j]
        j = j - 1
        k = k - 1
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
void mergeSortedArrays(vector<int>& A, vector<int>& B) {
    int lastA = A.size() - B.size() - 1;
    int lastB = B.size() - 1;
    int last = A.size() - 1;
```

```
    while (lastB >= 0) {
        if (lastA >= 0 && A[lastA] > B[lastB]) {
            A[last] = A[lastA];
            lastA--;
        } else {
            A[last] = B[lastB];
            lastB--;
        }
        last--;
    }
}

int main() {
    int sizeA, sizeB, input;
    cout << "Enter the number of elements in array A (including
buffer): ";
    cin >> sizeA;
    cout << "Enter the number of elements in array B: ";
    cin >> sizeB;

    vector<int> A(sizeA, 0);
    vector<int> B(sizeB, 0);

    cout << "Enter elements of array A (excluding buffer, sorted):
";
    for (int i = 0; i < sizeA - sizeB; ++i) {
        cin >> A[i];
    }

    cout << "Enter elements of array B (sorted): ";
    for (int i = 0; i < sizeB; ++i) {
        cin >> B[i];
    }

    mergeSortedArrays(A, B);

    cout << "Merged array: ";
    for (int num : A) {
        cout << num << " ";
    }
```

```
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Enter the number of elements in array A (including buffer): 5
Enter the number of elements in array B: 4
Enter elements of array A (excluding buffer, sorted): 1 2 3 4
Enter elements of array B (sorted): 7 8 9 0
Merged array: 1 2 3 4 7
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd
_1 } ; if ($?) { .\1_1 }
Enter the number of elements in array A (including buffer): 4
Enter the number of elements in array B: 3
Enter elements of array A (excluding buffer, sorted): 1 3 5
Enter elements of array B (sorted): 2 4 6
Merged array: 1 3 5 2
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd
_1 } ; if ($?) { .\1_1 }
Enter the number of elements in array A (including buffer): 6
Enter the number of elements in array B: 5
Enter elements of array A (excluding buffer, sorted): 3 4 56 6 7
Enter elements of array B (sorted): 2 3 5 7 1
Merged array: 4 56 6 7 2 3
```

```
Enter the number of elements in array A (including buffer): 5
Enter the number of elements in array B: 2
Enter elements of array A (excluding buffer, sorted): 4 5 6 7
Enter elements of array B (sorted): 8 9
Merged array: 4 5 6 7 8
```

TC: O(m + n)

SC:O(1)

2) Write a method to sort an array of string so that all the anagrams are next to each other.

Pseudocode:

```
function groupAnagrams(strings):
    create a hashmap anagrams
```

```
    for each string in strings:
        sortedStr = sort(string)
        if sortedStr is not in anagrams:
            anagrams[sortedStr] = empty list
        append string to anagrams[sortedStr]

    result = empty list
    for each group in anagrams:
        add group to result

    return result
```

CODE;

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
using namespace std;


void sortAnagrams(vector<string>& strings) {
    unordered_map<string, vector<string>> anagramGroups;

    for (const string& str : strings) {
        string sortedStr = str;
        sort(sortedStr.begin(), sortedStr.end());
        anagramGroups[sortedStr].push_back(str);
    }

    int index = 0;
    for (const auto& pair : anagramGroups) {
        for (const string& str : pair.second) {
            strings[index++] = str;
        }
    }
}

int main() {
```

```
    int numStrings;
    cout << "Enter the number of strings: ";
    cin >> numStrings;

    vector<string> strings(numStrings);
    cout << "Enter the strings: ";
    for (int i = 0; i < numStrings; ++i) {
        cin >> strings[i];
    }

    sortAnagrams(strings);

    cout << "Strings grouped by anagrams: ";
    for (const string& str : strings) {
        cout << str << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Enter the number of strings: 3
Enter the strings: a b c
Strings grouped by anagrams: c b a
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225100
_2 } ; if ($?) { .\1_2 }
Enter the number of strings: 5
Enter the strings: aa nn pp ee tt
Strings grouped by anagrams: tt aa nn pp ee
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225100
_2 } ; if ($?) { .\1_2 }
Enter the number of strings: 3
Enter the strings: cat elbow brag
Strings grouped by anagrams: brag elbow cat
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225100
_2 } ; if ($?) { .\1_2 }
Enter the number of strings: 3
Enter the strings: c a t
Strings grouped by anagrams: t a c
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225100
_2 } ; if ($?) { .\1_2 }
Enter the number of strings: 4
Enter the strings: apple banana berry lemon
Strings grouped by anagrams: berry banana lemon apple
```

TC: O(n * k log k)

SC:O(nk)

But here, TC:O(n log n)

SC:O(n)

3) Given a sorted array of *n* integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.

EXAMPLE

Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}

Output: 8 (the index of 5 in the array)

Pseudocode:

```
function searchInRotatedArray(arr, target):
```

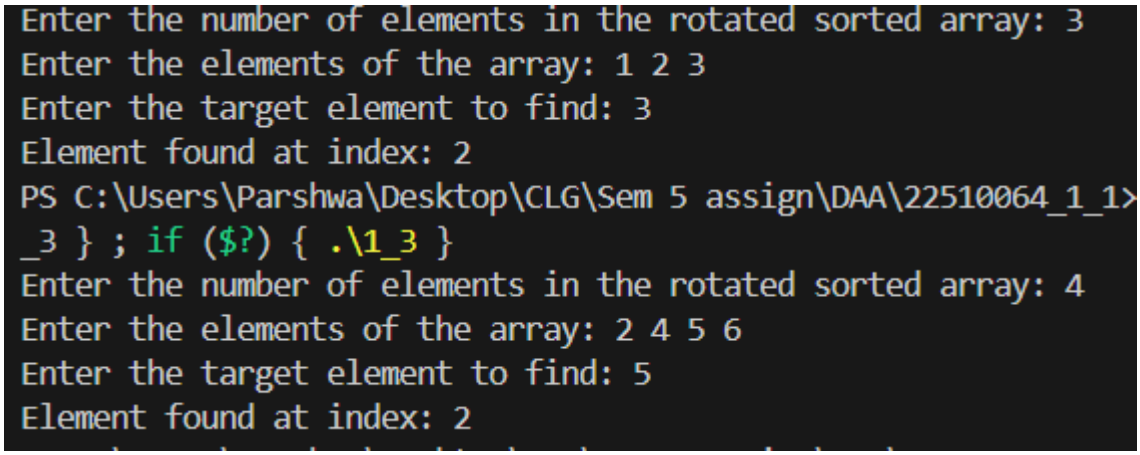
```
    low = 0
    high = length(arr) - 1

    while low <= high:
        mid = (low + high) / 2

        if arr[mid] == target:
            return mid

        if arr[low] <= arr[mid]:
            if arr[low] <= target < arr[mid]:
                high = mid - 1
            else:
                low = mid + 1
        else:
            if arr[mid] < target <= arr[high]:
                low = mid + 1
            else:
                high = mid - 1

    return -1
```

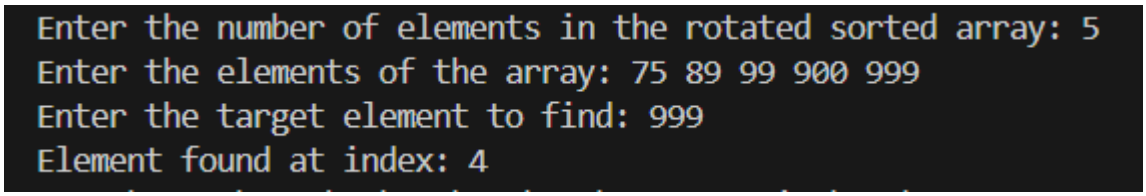
CODE:

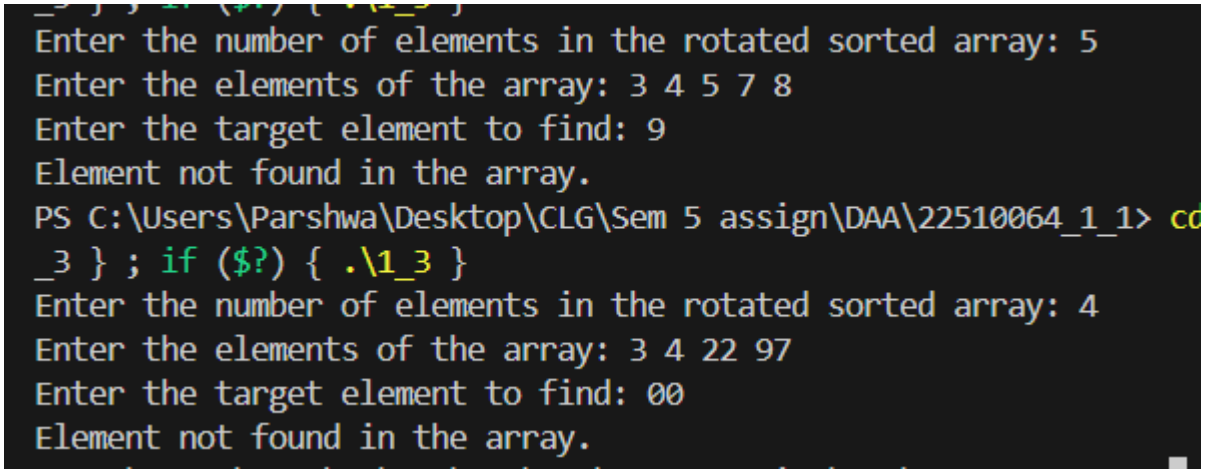
```
#include <iostream>
#include <vector>
using namespace std;


int searchInRotatedArray(const vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] == target) {
            return mid;
        }

        if (nums[left] <= nums[mid]) {
            if (target >= nums[left] && target < nums[mid]) {
```

```
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        } else {
            if (target > nums[mid] && target <= nums[right]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }

    return -1;
}

int main() {
    int n, target;
    cout << "Enter the number of elements in the rotated sorted
array: ";
    cin >> n;

    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }

    cout << "Enter the target element to find: ";
    cin >> target;

    int index = searchInRotatedArray(nums, target);

    if (index != -1) {
        cout << "Element found at index: " << index << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```

OUTPUT:







TC: O (log n)

4) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

SC:O(1)

Chunking: Divide the file into smaller, manageable chunks.

In-memory Sort: Sort each chunk using an efficient algorithm (e.g., Quicksort).

Create Min-Heap: Build a min-heap to hold smallest elements from each chunk.

Merge and Write: Iteratively extract minimum from heap, write to output, replace with next element from corresponding chunk.

Pseudocode:

```
function externalSort(filePath):
    CHUNK_SIZE = determine appropriate size
    chunks = empty list

    while not endOfFile(filePath):
        chunk = readChunk(filePath, CHUNK_SIZE)
        sort(chunk)
        chunkFile = writeToTemporaryFile(chunk)
        append chunkFile to chunks

    outputFile = createNewFile()
    mergeChunks(chunks, outputFile)

    return outputFile

function mergeChunks(chunks, outputFile):
    priorityQueue = new PriorityQueue()
    openFiles = empty list

    for each chunkFile in chunks:
        file = open(chunkFile)
        append file to openFiles
        insert (file.readLine(), file) into priorityQueue

    while not priorityQueue.isEmpty():
        (smallestLine, file) = extractMin(priorityQueue)
        write smallestLine to outputFile

        if not endOfFile(file):
            insert (file.readLine(), file) into priorityQueue

    close all files in openFiles
```

CODE:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <queue>
#include <utility>
using namespace std;

void splitAndSortChunks(const string& inputFile, const string&
tempFilePrefix, int chunkSize) {
    ifstream input(inputFile);
    string line;
    int chunkNumber = 0;

    while (!input.eof()) {
        vector<string> lines;
        lines.reserve(chunkSize);

        for (int i = 0; i < chunkSize && getline(input, line); ++i)
{
            lines.push_back(line);
        }

        sort(lines.begin(), lines.end());

        ofstream tempFile(tempFilePrefix + to_string(chunkNumber++)
+ ".txt");
        for (const string& sortedLine : lines) {
            tempFile << sortedLine << '\n';
        }
        tempFile.close();
    }

    input.close();
}

void mergeChunks(const string& tempFilePrefix, int numChunks, const
string& outputFile) {
```

```
    ofstream output(outputFile);
    vector<ifstream> tempFiles(numChunks);

    auto cmp = [](const pair<string, int>& a, const pair<string,
int>& b) {
        return a.first > b.first;
    };
    priority_queue<pair<string, int>, vector<pair<string, int>>,
decltype(cmp)> minHeap(cmp);

    for (int i = 0; i < numChunks; ++i) {
        tempFiles[i].open(tempFilePrefix + to_string(i) + ".txt");
        string line;
        if (getline(tempFiles[i], line)) {
            minHeap.push(make_pair(line, i));
        }
    }

    while (!minHeap.empty()) {
        pair<string, int> smallestPair = minHeap.top();
        minHeap.pop();

        string smallest = smallestPair.first;
        int fileIndex = smallestPair.second;

        output << smallest << '\n';

        string nextLine;
        if (getline(tempFiles[fileIndex], nextLine)) {
            minHeap.push(make_pair(nextLine, fileIndex));
        }
    }

    output.close();

    for (int i = 0; i < numChunks; ++i) {
        tempFiles[i].close();
    }
}

int main() {
```

```
    string inputFile = "input.txt";
    string outputFile = "output.txt";
    string tempFilePrefix = "temp_chunk_";
    int chunkSize = 3;

    splitAndSortChunks(inputFile, tempFilePrefix, chunkSize);

    int numChunks = 4;
    mergeChunks(tempFilePrefix, numChunks, outputFile);

    cout << "File sorted successfully. Check the output file: " <<
outputFile << endl;
    return 0;
}
```

INPUT.TXT:

```
1   apple
2   orange
3   banana
4   grape
5   kiwi
6   strawberry
7   mango
8   pear
9   peach
10  cherry
11  melon
12  pineapple
13
```

input.txt

```
a
df
gr
juu
bt
ty
```

input.txt

```
jan
mar
oct
dec
nov
```

OUPUT.TXT:

```
apple
banana
cherry
grape
kiwi
mango
melon
orange
peach
pear
pineapple
strawberry
```

output.txt

```
1  a
2  bt
3  df
4  gr
5  juu
6  ty
7
```

output.txt

```
1  dec
2  jan
3  mar
4  nov
5  oct
6
```

OUTPUT:

```
File sorted successfully. Check the output file: output.txt
```

TC: Too large(depends on chunk)

5) Given a sorted array of string which is interspersed with empty string, write a method to find the location of a given string.

EXAMPLE

Input: find “ball” in {“at”, “”, “”, “ball”, “”, “”, “car”, “”, “”, “dad”, “”,””}

Output: 4

Pseudocode:

```
function searchInSparseArray(arr, target):
    low = 0
    high = length(arr) - 1

    while low <= high:
        mid = (low + high) / 2
```

```
        if arr[mid] == "":
            left = mid - 1
            right = mid + 1

            while true:
                if left < low and right > high:
                    return -1
                if right <= high and arr[right] != "":
                    mid = right
                    break
                if left >= low and arr[left] != "":
                    mid = left
                    break
                right = right + 1
                left = left - 1

        if arr[mid] == target:
            return mid
        else if arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1
```

CODE:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;


int findStringWithEmpty(const vector<string>& strings, const string&
target) {
    int left = 0, right = strings.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
```

```
        int midLeft = mid, midRight = mid;
        while (midLeft >= left && strings[midLeft].empty()) --
midLeft;
        while (midRight <= right && strings[midRight].empty())
++midRight;

        if (midLeft < left && midRight > right) {
            return -1;
        }

        mid = (midLeft >= left) ? midLeft : midRight;

        if (strings[mid] == target) {
            return mid;
        } else if (strings[mid] < target) {
            left = midRight + 1;
        } else {
            right = midLeft - 1;
        }
    }

    return -1;
}

int main() {
    int numStrings;
    cout << "Enter the number of strings: ";
    cin >> numStrings;

    vector<string> strings(numStrings);
    cout << "Enter the strings (use empty strings as needed): ";
    for (int i = 0; i < numStrings; ++i) {
        cin >> strings[i];
    }

    string target;
    cout << "Enter the target string to find: ";
    cin >> target;

    int index = findStringWithEmpty(strings, target);
```

```
    if (index != -1) {
        cout << "String found at index: " << index << endl;
    } else {
        cout << "String not found in the array." << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter the number of strings: 4
Enter the strings (use empty strings as needed): iron man '' ''
Enter the target string to find: thor
String not found in the array.
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd "c:
_5 } ; if ($?) { .\1_5 }
Enter the number of strings: 5
Enter the strings (use empty strings as needed): cap america '' hulk
Enter the target string to find: hulk
String found at index: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd "c:
_5 } ; if ($?) { .\1_5 }
Enter the number of strings: 3
Enter the strings (use empty strings as needed): hawk '' eye
Enter the target string to find: ''
String found at index: 1
```

```
Enter the number of strings: 4
Enter the strings (use empty strings as needed): a b c d
Enter the target string to find: c
String found at index: 2
```

TC: O(log n)

SC:O(n)

6) Given an M*N matrix in which each row and each column is sorted in ascending order, write a method to find an element.
Pseudocode:

```
function searchInSortedMatrix(matrix, target):
    row = 0
    col = number of columns in matrix - 1

    while row < number of rows in matrix and col >= 0:
        if matrix[row][col] == target:
            return (row, col)
        else if matrix[row][col] > target:
            col = col - 1
        else:
            row = row + 1

    return (-1, -1)
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;


bool searchMatrix(const vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) {
        return false;
    }

    int row = 0, col = matrix[0].size() - 1;

    while (row < matrix.size() && col >= 0) {
        if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] > target) {
            --col;
        } else {
            ++row;
        }
    }

    return false;
}

int main() {
```

```
    int rows, cols, target;
    cout << "Enter the number of rows in the matrix: ";
    cin >> rows;
    cout << "Enter the number of columns in the matrix: ";
    cin >> cols;

    vector<vector<int>> matrix(rows, vector<int>(cols));
    cout << "Enter the elements of the matrix (sorted row-wise and
column-wise): ";
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cin >> matrix[i][j];
        }
    }

    cout << "Enter the target element to find: ";
    cin >> target;

    bool found = searchMatrix(matrix, target);

    if (found) {
        cout << "Element found in the matrix." << endl;
    } else {
        cout << "Element not found in the matrix." << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter the number of rows in the matrix: 2
Enter the number of columns in the matrix: 2
Enter the elements of the matrix (sorted row-wise and column-wise): 1 2 3 4
Enter the target element to find: 5
Element not found in the matrix.
```

```
Enter the number of rows in the matrix: 3
Enter the number of columns in the matrix: 3
Enter the elements of the matrix (sorted row-wise and column-wise): 1 2 3 4 5 6 7 8 9
Enter the target element to find: 8
Element found in the matrix.
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd "c:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064
_6 } ; if ($?) { .\1_6 }
Enter the number of rows in the matrix: 4
Enter the number of columns in the matrix: 4
Enter the elements of the matrix (sorted row-wise and column-wise):  11 22 34 56 78 89 90 91 92 94 96 98 99 100 150 200
Enter the target element to find: 100
Element found in the matrix.
```

TC: O(m + n)

SC:O(n)

7) A circus is designing a tower routine consisting of people standing atop one another's shoulders. For practical and aesthetic reasons, each person must be both shorter and lighter than the person below him or her. Given the heights and weight of each circus, write a method to compute the largest possible number of people in such tower.

EXAMPLE:

Input(ht,wt): (65, 100) (70, 150) (56, 90) (75,190) (60, 95) (68, 110).

Output: The longest tower is length 6 and includes from top to bottom:

(56, 90) (60, 95) (65, 100) (68, 110) (70, 150) (75, 190)

Pseudocode:

```
function maxCircusTower(people):
    sort people by height, breaking ties by weight

    weights = extract weights from sorted people
    return longestIncreasingSubsequence(weights)

function longestIncreasingSubsequence(weights):
    lis = empty list

    for each weight in weights:
        index = find position to insert weight in lis (using binary
search)
        if index is end of lis:
            append weight to lis
        else:
            replace lis[index] with weight

    return length of lis
```

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;


struct Person {
    int height;
    int weight;

    bool operator<(const Person& other) const {
        return height < other.height || (height == other.height &&
weight < other.weight);
    }
};

int maxTowerLength(vector<Person>& people) {
    sort(people.begin(), people.end());
    int n = people.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (people[i].weight > people[j].weight) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
    }

    return *max_element(dp.begin(), dp.end());
}

int main() {
    int numPeople;
    cout << "Enter the number of people: ";
    cin >> numPeople;

    vector<Person> people(numPeople);
```

```
    cout << "Enter the height and weight of each person: ";
    for (int i = 0; i < numPeople; ++i) {
        cin >> people[i].height >> people[i].weight;
    }

    int maxLength = maxTowerLength(people);

    cout << "The longest tower has length: " << maxLength << endl;
    return 0;
}
```

OUTPUT:

```
Enter the number of people: 3
Enter the height and weight of each person: 180 50 178 60 176 70
The longest tower has length: 1
```

```
Enter the number of people: 3
Enter the height and weight of each person: 170 50 190 77 200 89
The longest tower has length: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd "c:\Us
_7 } ; if ($?) { .\1_7 }
Enter the number of people: 4
Enter the height and weight of each person: 100 50 150 60 160 99 170 95
The longest person has height: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_1_1> cd "c:\Us
_7 } ; if ($?) { .\1_7 }
Enter the number of people: 4
Enter the height and weight of each person: 100 34 170 45 175 78 180 64
The longest person has height: 3
```

TC: O(n^2)

SC:O(n)

8) Imagine you are reading in stream of integers. Periodically, you wish to be able to look up the rank of number *x* (the number of values less than or equal to *x*). Implement the data structures and algorithms to support these operations. That is, Implement the method *track (int x)*, which is called when each number is generated, and the method *getRankOfNumber (int x)*, which return the number of values less than or equal to *x* (not including x itself).

EXAMPLE

Stream (in order of appearance) : 5, 1, 4, 4, 5, 9, 7, 13, 3

getRankOfNumber(1) = 0

getRankOfNumber(3) = 1

getRankOfNumber(4) =3

Pseudocode:

```
class RankStream:
    def __init__():
        self.data = empty list

    function track(x):
        insert x into self.data in sorted order

    function getRankOfNumber(x):
        rank = 0
        for num in self.data:
            if num <= x:
                rank = rank + 1
            else:
                break
        return rank
```

```
CODE: #include <iostream>
#include <vector>
using namespace std;

class RankNode {
public:
    int data;
    RankNode* left;
    RankNode* right;
    int leftSize;

    RankNode(int data) : data(data), left(nullptr), right(nullptr),
leftSize(0) {}
};

class RankTree {
public:
    RankNode* root;
```

```
    RankTree() : root(nullptr) {}

    void track(int x) {
        root = insert(root, x);
    }

    int getRankOfNumber(int x) {
        return getRank(root, x);
    }

private:
    RankNode* insert(RankNode* node, int x) {
        if (node == nullptr) {
            return new RankNode(x);
        }

        if (x < node->data) {
            node->left = insert(node->left, x);
            node->leftSize++;
        } else {
            node->right = insert(node->right, x);
        }

        return node;
    }

    int getRank(RankNode* node, int x) {
        if (node == nullptr) {
            return -1;
        }

        if (x == node->data) {
            return node->leftSize;
        } else if (x < node->data) {
            return getRank(node->left, x);
        } else {
            int rightRank = getRank(node->right, x);
            if (rightRank == -1) {
                return -1;
            }
```

```
            return node->leftSize + 1 + rightRank;
        }
    }
};

int main() {
    RankTree rankTree;
    int n, query, number;
    cout << "Enter the number of elements in the stream: ";
    cin >> n;

    vector<int> stream(n);
    cout << "Enter the elements of the stream: ";
    for (int i = 0; i < n; ++i) {
        cin >> stream[i];
        rankTree.track(stream[i]);
    }

    cout << "Enter the number to find the rank of: ";
    cin >> query;

    int rank = rankTree.getRankOfNumber(query);

    if (rank != -1) {
        cout << "Rank of " << query << " is: " << rank << endl;
    } else {
        cout << query << " not found in the stream." << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter the number of elements in the stream: 3
Enter the elements of the stream: 1 2 3
Enter the number to find the rank of: 2
Rank of 2 is: 1
```

```
Enter the number of elements in the stream: 5
Enter the elements of the stream: 67 87 90 78 90
Enter the number to find the rank of: 90
Rank of 90 is: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225
_8 } ; if ($?) { .\1_8 }
Enter the number of elements in the stream: 4
Enter the elements of the stream: 4 99 100 235
Enter the number to find the rank of: 100
Rank of 100 is: 2
```

```
Enter the number of elements in the stream: 6
Enter the elements of the stream: 1 2 3 8 99 101
Enter the number to find the rank of: 101
Rank of 101 is: 5
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225
```

TC: O(log n)

SC:O(n)