**Batch:T6**

**Practical No.2**

**Title of Assignment: Searching Algorithm**

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

Q.1You are an IT company's manager. Based on their performance over the last N working days, you must rate your employee. You are given an array of N integers called workload, where workload[i] represents the number of hours an employee worked on an ith day. The employee must be evaluated using the following criteria:

* Rating = the maximum number of consecutive working days when the employee has worked more than 6 hours.

You are given an integer *N*where *N* represents the number of working days. You are given an integer array *workload*where *workload[i]* represents the number of hours an employee worked on an ith day.

**Task**

Determine the employee rating.

**Pseudocode:**

Initialize Variables:

max\_streak = 0

current\_streak = 0

Loop Through Each Day's Workload:

For each hours in workload:

If hours > 6:

Increment current\_streak

Update max\_streak = max(max\_streak, current\_streak)

Else:

Reset current\_streak = 0

Output the Result:

Print max\_streak **Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int findEmployeeRating(const vector<int>& workload) {

    int max\_streak = 0, current\_streak = 0;

    for (int hours : workload) {

        if (hours > 6) {

            current\_streak++;

            max\_streak = max(max\_streak, current\_streak);

        } else {

            current\_streak = 0;

        }

    }

    return max\_streak;

}

vector<int> getWorkloadInput(int N) {

    vector<int> workload(N);

    cout << "Enter the workload for each day: ";

    for (int i = 0; i < N; ++i) {

        cin >> workload[i];

    }

    return workload;

}

int main() {

    int N;

    cout << "Enter the number of working days: ";

    cin >> N;

    vector<int> workload = getWorkloadInput(N);

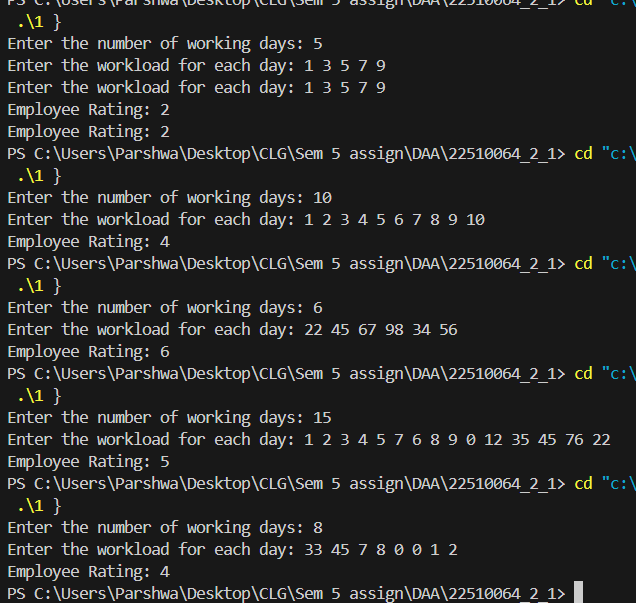
    int rating = findEmployeeRating(workload);

    cout << "Employee Rating: " << rating << endl;

    return 0;

}

**Output:**

**  
Time Complexity:** O(N)

**Space Complexity:** O(1)

Q.2 You have N boxes numbered 1 through N and K candies numbered 1 through K. You put the candies in the boxes in the following order:

* first candy in the first box,
* second candy in the second box,
* .......
* ........
* so up to N-th candy in the Nth box,
* the next candy in (N - 1)-th box,
* the next candy in (N - 2)-th box
* ........
* .......
* and so on up to the first box,
* then the next candy in the second box
* ......    and so on until there is no candy left.

So you put the candies in the boxes in the following order:

Find the index of the box where you put the K-th candy.

**Pseudocode**: Initialize Variables:

index = 1

direction = 1 (1 for forward, -1 for backward)

Loop K-1 Times (from 1 to K-1):

If index == N: Set direction = -1

If index == 1: Set direction = 1

Update index += direction

Output the Result: Return index

**Code:**

#include <iostream>

#include <tuple>

using namespace std;

int findBoxIndex(int N, int K) {

    int currentCandy = 0;

    int step = 1;

    int position = 1;

    while (currentCandy < K) {

        position += step;

        currentCandy++;

        if (position == N || position == 1) {

            step = -step;

        }

    }

    return position;

}

tuple<int, int> getBoxCandyInput() {

    int N, K;

    cout << "Enter the number of boxes (N) and candies (K): ";

    cin >> N >> K;

    return make\_tuple(N, K);

}

int main() {

    int N, K;

    tie(N, K) = getBoxCandyInput();

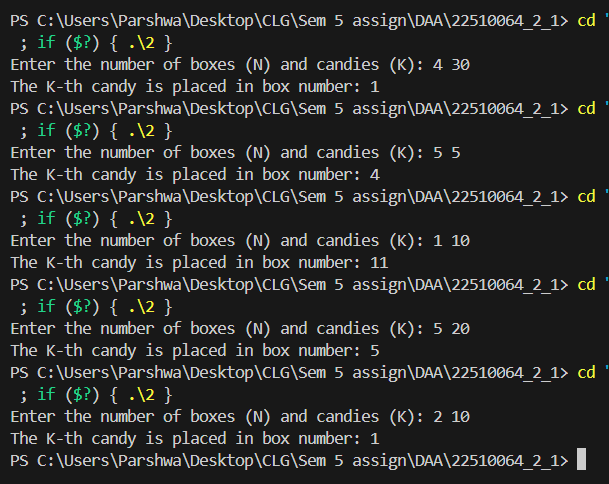
    int boxIndex = findBoxIndex(N, K);

    cout << "The K-th candy is placed in box number: " << boxIndex << endl;

    return 0;

}

**Output:**

**  
Time Complexity:** O(k)

**Space Complexity:** O(1)

Q.3 Implement and Explain Tower of Hanoi algorithm.

**Pseudocode:**

If N == 1:

Move disk from source to destination

Else:

Move N-1 disks from source to auxiliary

Move the N-th disk from source to destination

Move N-1 disks from auxiliary to destination

**Code:**

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char source, char destination, char auxiliary) {

    if (n == 1) {

        cout << "Move disk 1 from " << source << " to " << destination << endl;

        return;

    }

    towerOfHanoi(n - 1, source, auxiliary, destination);

    cout << "Move disk " << n << " from " << source << " to " << destination << endl;

    towerOfHanoi(n - 1, auxiliary, destination, source);

}

int getDiskInput() {

    int N;

    cout << "Enter the number of disks: ";

    cin >> N;

    return N;

}

int main() {

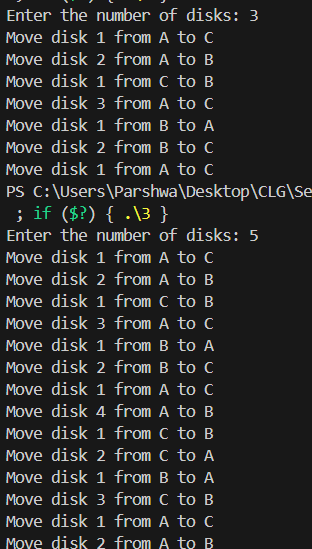
    int N = getDiskInput();

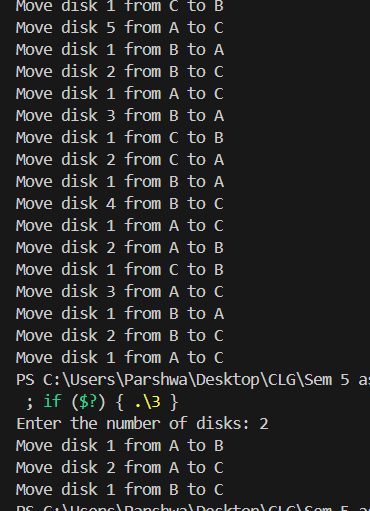
    towerOfHanoi(N, 'A', 'C', 'B');

    return 0;

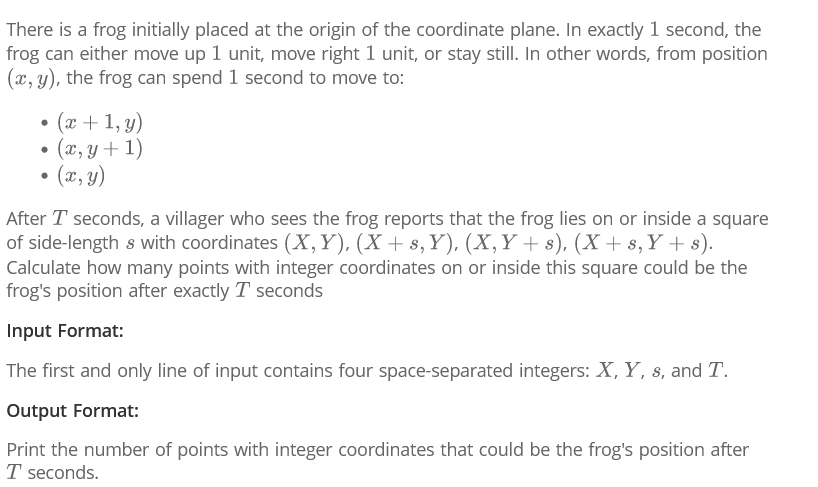
}

**Output:**

****

**  
Time Complexity: O(2^n)**

**Space Complexity: O(N)**

Q.4 

**Pseudocode:**

Calculate Range:

min\_x = max(0, X)

max\_x = min(T, X + s)

min\_y = max(0, Y)

max\_y = min(T, Y + s)

Count Valid Points:

Initialize count = 0

For each x from min\_x to max\_x:

For each y from min\_y to max\_y:

If x + y <= T: Increment count

Output the Result: Return count

**Code:**

#include <iostream>

#include <tuple>

#include <algorithm>

using namespace std;

int countPossiblePositions(int X, int Y, int s, int T) {

    int count = 0;

    int min\_x = max(0, X);

    int max\_x = min(T, X + s);

    int min\_y = max(0, Y);

    int max\_y = min(T, Y + s);

    for (int x = min\_x; x <= max\_x; ++x) {

        for (int y = min\_y; y <= max\_y; ++y) {

            if (x + y <= T) {

                count++;

            }

        }

    }

    return count;

}

tuple<int, int, int, int> getFrogInput() {

    int X, Y, s, T;

    cout << "Enter X, Y, s, and T: ";

    cin >> X >> Y >> s >> T;

    return make\_tuple(X, Y, s, T);

}

int main() {

    int X, Y, s, T;

    tie(X, Y, s, T) = getFrogInput();

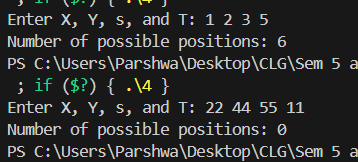
    int result = countPossiblePositions(X, Y, s, T);

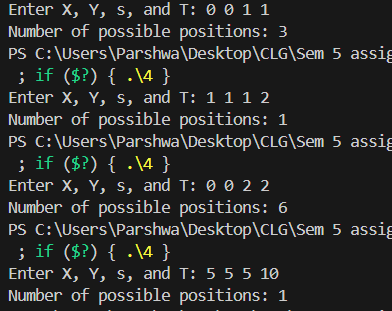
    cout << "Number of possible positions: " << result << endl;

    return 0;

}

**Output:**

****

**   
Time Complexity: O(s^2), s=length of a side**

**Space Complexity: O(1)**

Q.5 Implement linear search Algorithm.

**Pseudocode:**

Loop Through Each Element:

For each element in array:

If element equals target:

Return the current index

If Element Not Found:Return -1

**Code:**

#include <iostream>

#include <vector>

using namespace std;

int linearSearch(const vector<int>& arr, int target) {

    for (size\_t i = 0; i < arr.size(); ++i) {

        if (arr[i] == target) {

            return i;

        }

    }

    return -1;

}

vector<int> getArrayInput(int N) {

    vector<int> arr(N);

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

    for (int i = 0; i < N; ++i) {

        cin >> arr[i];

    }

    return arr;

}

int main() {

    int N, target;

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

    cin >> N;

    vector<int> arr = getArrayInput(N);

    cout << "Enter the target element: ";

    cin >> target;

    int index = linearSearch(arr, target);

    if (index != -1) {

        cout << "Element found at index: " << index << endl;

    } else {

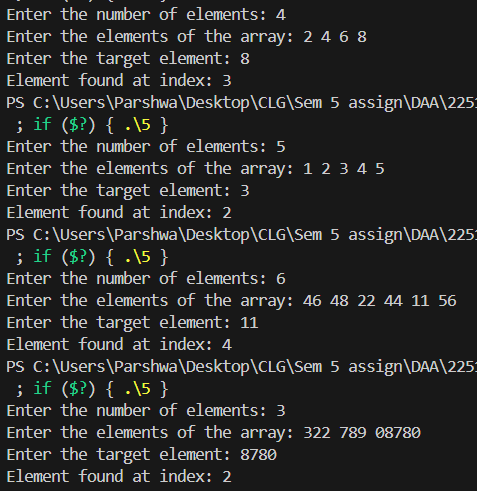
        cout << "Element not found." << endl;

    }

    return 0;

}

**Output:**

**  
Time Complexity: O(N)**

**Space Complexity: O(1)**

Q.6 Implement Binary Search algorithm.  
**Pseudocode:**

Initialize Variables:

left = 0

right = last index of array

While left <= right:

Calculate mid = left + (right - left) / 2

If array[mid] == target: Return mid

If array[mid] < target: Set left = mid + 1

Else: Set right = mid - 1

If Element Not Found:

Return -1

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

int binarySearch(const std::vector<int>& arr, int target) {

    int left = 0;

    int right = arr.size() - 1;

    while (left <= right) {

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

        if (arr[mid] == target) {

            return mid;

        } else if (arr[mid] < target) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1;

}

std::vector<int> getSortedArrayInput(int N) {

    std::vector<int> arr(N);

    std::cout << "Enter the sorted elements: ";

    for (int i = 0; i < N; ++i) {

        std::cin >> arr[i];

    }

    return arr;

}

int main() {

    int N, target;

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

    std::cin >> N;

    std::vector<int> arr = getSortedArrayInput(N);

    std::cout << "Enter the target element: ";

    std::cin >> target;

    int index = binarySearch(arr, target);

    if (index != -1) {

        std::cout << "Element found at index: " << index << std::endl;

    } else {

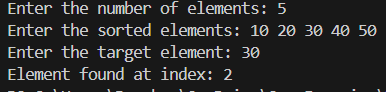
        std::cout << "Element not found." << std::endl;

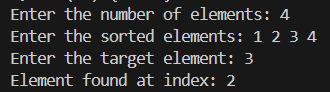
    }

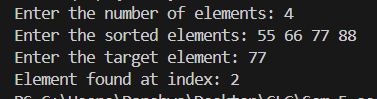
    return 0;

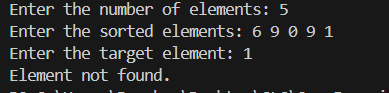
}

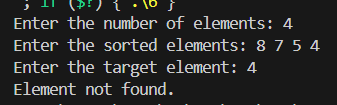
**Output:**

****

****

****

****

**  
Time Complexity: O(log n)**

**Space Complexity: O(1)**