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 i^{th} 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 i^{th} 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:

```cpp
#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:

```
 .\1 }
Enter the number of working days: 5
Enter the workload for each day: 1 3 5 7 9
Enter the workload for each day: 1 3 5 7 9
Employee Rating: 2
Employee Rating: 2
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd "c:\
 .\1 }
Enter the number of working days: 10
Enter the workload for each day: 1 2 3 4 5 6 7 8 9 10
Employee Rating: 4
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd "c:\
 .\1 }
Enter the number of working days: 6
Enter the workload for each day: 22 45 67 98 34 56
Employee Rating: 6
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd "c:\
 .\1 }
Enter the number of working days: 15
Enter the workload for each day: 1 2 3 4 5 7 6 8 9 0 12 35 45 76 22
Employee Rating: 5
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd "c:\
 .\1 }
Enter the number of working days: 8
Enter the workload for each day: 33 45 7 8 0 0 1 2
Employee Rating: 4
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1>
```

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:

```
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd '
 ; if ($?) { .\2 }
Enter the number of boxes (N) and candies (K): 4 30
The K-th candy is placed in box number: 1
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd '
 ; if ($?) { .\2 }
Enter the number of boxes (N) and candies (K): 5 5
The K-th candy is placed in box number: 4
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd '
 ; if ($?) { .\2 }
Enter the number of boxes (N) and candies (K): 1 10
The K-th candy is placed in box number: 11
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd '
 ; if ($?) { .\2 }
Enter the number of boxes (N) and candies (K): 5 20
The K-th candy is placed in box number: 5
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1> cd '
 ; if ($?) { .\2 }
Enter the number of boxes (N) and candies (K): 2 10
The K-th candy is placed in box number: 1
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\22510064_2_1>
```

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

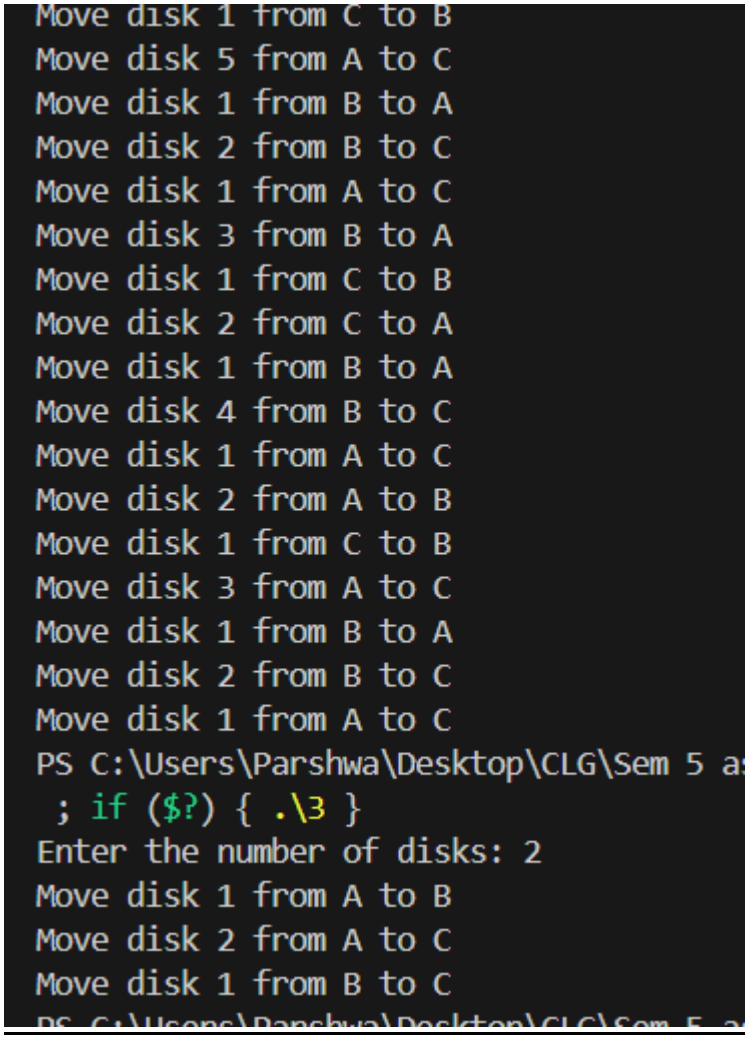
Code:

```
#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:

```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
PS C:\Users\Parshwa\Desktop\CLG\Se
 ; if ($?) { .\3 }
Enter the number of disks: 5
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Move disk 4 from A to B
Move disk 1 from C to B
Move disk 2 from C to A
Move disk 1 from B to A
Move disk 3 from C to B
Move disk 1 from A to C
Move disk 2 from A to B
```



Time Complexity: O(2^n)

Space Complexity: O(N)

Q.4

There is a frog initially placed at the origin of the coordinate plane. In exactly 1 second, the frog can either move up 1 unit, move right 1 unit, or stay still. In other words, from position (x, y), the frog can spend 1 second to move to:

- $(x+1, y)$
- $(x, y+1)$
- (x, y)

After T seconds, a villager who sees the frog reports that the frog lies on or inside a square of side-length s with coordinates (X, Y), $(X+s, Y)$, $(X, Y+s)$, $(X+s, Y+s)$. Calculate how many points with integer coordinates on or inside this square could be the frog's position after exactly T seconds

Input Format:

The first and only line of input contains four space-separated integers: X, Y, s, and T.

Output Format:

Print the number of points with integer coordinates that could be the frog's position after T seconds.

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>
```

```cpp
#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:

```
Enter X, Y, s, and T: 1 2 3 5
Number of possible positions: 6
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 a
 ; if ($?) { .\4 }
Enter X, Y, s, and T: 22 44 55 11
Number of possible positions: 0
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 a
```

```
Enter X, Y, s, and T: 0 0 1 1
Number of possible positions: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assig
 ; if ($?) { .\4 }
Enter X, Y, s, and T: 1 1 1 2
Number of possible positions: 1
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assig
 ; if ($?) { .\4 }
Enter X, Y, s, and T: 0 0 2 2
Number of possible positions: 6
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assig
 ; if ($?) { .\4 }
Enter X, Y, s, and T: 5 5 5 10
Number of possible positions: 1
```

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:

```
Enter the number of elements: 4
Enter the elements of the array: 2 4 6 8
Enter the target element: 8
Element found at index: 3
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225
 ; if ($?) { .\5 }
Enter the number of elements: 5
Enter the elements of the array: 1 2 3 4 5
Enter the target element: 3
Element found at index: 2
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225
 ; if ($?) { .\5 }
Enter the number of elements: 6
Enter the elements of the array: 46 48 22 44 11 56
Enter the target element: 11
Element found at index: 4
PS C:\Users\Parshwa\Desktop\CLG\Sem 5 assign\DAA\225
 ; if ($?) { .\5 }
Enter the number of elements: 3
Enter the elements of the array: 322 789 08780
Enter the target element: 8780
Element found at index: 2
```

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:

```
Enter the number of elements: 5
Enter the sorted elements: 10 20 30 40 50
Enter the target element: 30
Element found at index: 2
```

```
Enter the number of elements: 4
Enter the sorted elements: 1 2 3 4
Enter the target element: 3
Element found at index: 2
```

```
Enter the number of elements: 4
Enter the sorted elements: 55 66 77 88
Enter the target element: 77
Element found at index: 2
```

```
Enter the number of elements: 5
Enter the sorted elements: 6 9 0 9 1
Enter the target element: 1
Element not found.
```

```
; if ($?) { .\b }
Enter the number of elements: 4
Enter the sorted elements: 8 7 5 4
Enter the target element: 4
Element not found.
```

Time Complexity: O(log n)

Space Complexity: O(1)