CSE 321

Homework 4

Berke Belgin

171044065

1)

00000000000000000… -> n

For an n sized 0 sequence, in worst scenario it will take **n – 3** comparisons to verify if the sequence contains a sequence like “0010” since the length of the searched text is 4.

…000000

The last possible sequence is the last four characters in terms of character length, so we do not check the last three. Since the character that breaks the pattern is the third one (0010), there will be three characters comparisons for each n sized sequence character which is equal to **3.(n - 3)** = **3n - 9**.

The worst-case input pattern of length 3 is **001** since the character that breaks the pattern is the last one.

2)

Since the starting point and traveling in forward or reverse order of the same point sequence does not affect total distance travelled, we will divide permutation of 5 by 5 and 2 respectively as,

$$\frac{\mathrm{P}(5,5)}{5.2} = \frac{5!}{5.2} = 4.3 = 12$$

A -> B -> E -> D -> C -> A = 16

A -> D -> C -> B -> E -> A = 16

A -> B -> C -> D -> E -> A = 22

A -> B -> C -> E -> D -> A = 25

A -> B -> D -> C -> E -> A = 21

A -> E -> D -> E -> C -> A = 27

A -> B -> E -> C -> D -> A = 19

A -> C -> B -> D -> E -> A = 27

A -> C -> B -> E -> D -> A = 22

A -> C -> D -> B -> E -> A = 18

A -> C -> E -> B -> D -> A = 21

A -> D -> B -> C -> E -> A = 24

3)

```
PROCEDURE log2(n)
        IF n <= 1 THEN
                RETURN 0
        ELSE THEN
                RETURN 1 + log2(n / 2)
        ENDIF
END
```

$T(n) = T(n / 2) + 1, n > 1, A(1) = 0$

$T(n) = \log_2 n \in Q(\log n)$

4)

```
PROCEDURE findBottle(arr, avrWeight)
        IF arr1.size == 1 THEN
                RETURN arr[0]
        ENDIF

        totalWeightL = totalWeight(arr, 0, arr.size / 2)
        totalWeightR = totalWeight(arr, arr.size / 2 + 1, arr.size)

        IF totalWeightL != avrWeight  * (arr.size / 2) THEN
                RETURN getXth(arr[0 : arr.size / 2])
        ELSE THEN
                RETURN getXth(arr[arr.size / 2 + 1 : arr.size])
        ENDIF
END

PROCEDURE totalWeight(arr, start, end)
        totalWeight = 0
        FOR i IN range(start, end)
                totalWeight = totalWeight + arr[i]
        ENDFOR
        RETURN totalWeight
END
```

In this part I implemented an algorithm that divides the bottles into two halves, calculates their total weight and checks which half does not meet “bottle count * average bottle weight” requirement and then calls the recursive function again for that incorrect half. The best, worst and average case complexities of the algorithm is Q(logn) if we assume, we can get the total weight of a half in constant time.

5)

```
PROCEDURE getXth(arr1, arr2, x)
        IF arr1.size == 0 THEN
                RETURN arr2[x]
        ENDIF
        IF arr2.size == 0 THEN
                RETURN arr1[x]
        ENDIF

        mid1 = (arr1.size - 1) / 2
        mid2 = (arr2.size - 1) / 2

        IF mid1 + mid2 < x THEN
                IF arr1[mid1] > arr2[mid2] THEN
                        RETURN getXth(arr1, arr2[mid2 + 1 : arr2.size], x - mid2 - 1)
                ELSE THEN
                        RETURN getXth(arr1[mid1 + 1 : arr1.size], arr2, x - mid1 - 1)
                ENDIF
        ELSE THEN
                IF arr1[mid1] > arr2[mid2] THEN
                        RETURN getXth(arr1[0 : mid1], arr2, x)
                ELSE THEN
                        RETURN getXth(arr1, arr2[0 : mid2], x)
                ENDIF
        ENDIF
END
```

Worst-case -> O(logn + logm)

n -> arr1.size,

m -> arr2.size