Berkcan EKİCİ
171044015

Answer 1 :

a) $T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$T(0) = 1$

$T(n) = T(n-2) + 1 + 1$

$T(n) = T(n-3) + 1 + 1 + 1$

$T(n) = T(n-4) + 1 + 1 + 1 + 1$

$\vdots$

$T(n) = T(0) + \underbrace{1 + 1 + \dots + 1}_{n}$

$1 + n = O(n)$

b) $T(n) = 2T(n/2) + 1$

$T(n/2) = 2T(n/4) + 1$

$T(n/4) = 2T(n/8) + 1$

$T(1) = 1$

$T(n) = 2(2T(n/4) + 1) + 1$

$T(n) = 2(2(2T(n/8) + 1) + 1) + 1$

$T(n) = 2^k . T(n/2^k) + 1 + 2 + \dots + k$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2^n$

$\frac{2^{\log_2^n} - 1}{2} = \frac{n-1}{2}$

$T(n) = 2^{\log_2^n} . T(1) + \frac{n-1}{2} = n + \frac{n-1}{2} = \frac{3n-1}{2}$

$= \frac{3}{2} . n \Rightarrow O(n)$

Berkcan EKİCİ
171044015

Answer 2 : A polynomial is an expression that contains more than two terms. A term comprises of a coefficient and an exponent.

Example : $6x^4 + 8x^3 + 3x^2 + 5x + 4$

Brute-force method: 6*x*x*x*x + 8*x*x*x + 3*x*x + 5*x + 4

Pseudocode :

```
Algorithm Polynomial(x, coefficients[])
    n←len(coefficients)
    result ← 0
    for i←0 to (n-1) do
        result += coefficients[n-1]
    endfor
    result += coefficients[n-1]
    return result
```

Analysis of Brute Force Method : A brute force approach to evaluate a polynomial is to evaluate all terms one by one. First calculate x^n, multiply the value with the related coefficient a_n, repeat the same steps for other terms and return the sum.

In the first term, it will take n multiplications, in the second term it will take n-1 multiplications, in the third term it takes n-2 multiplications...

In the last two terms : $a_2*x*x + a_1*x$ it takes 2 multiplications and 1 multiplication accordingly.

Number of multiplications needed in the worst case is :

$$T(n) = n + (n-1) + (n-2) + \dots + 2 + 1$$
$$= n.(n+1)/2 = O(n^2)$$

Berkcan Ekici
171044015

Example: $6x^4+8x^3+3x^2+5x+4$

Horner's Method: $(((6*x+8)*x+3)*x+5)*x+4$

In the first term, it takes one multiplication, in the second term one multiplication, in the third term it takes one multiplication... Similarly in all other terms it will take one multiplication.

Pseudocode:

```
Algorithm Polynomial(x, coefficients[])
    result ← coefficients[0]
    for i←1 to length(coefficients) do
        result ← result*x + coefficients[i]
    return result
```

Analysis of Horner's Method: Number of multiplications needed in the worst case is:

$$T(n) = \sum_{i=1}^{n} 1 = n$$

$$T(n) = n = O(n)$$

* Just computing single term by the brute force algorithm would require n multiplications, whereas Horner's rule requires only one multiplication in every term.

Berkcan EKİCİ
171044015

Answer 3 :

Algorithm : Initialize the count of the desired substrings to 0. Scan the text left to right doing the following for every character except the last one: If a 'X' is encountered, count the number of all the 'Z's following it and add this number to the count of desired substrings. After the scan ends, return the last value of the count.

Pseudocode :

```
Algorithm NumberOfSubstrings(text[ ], firstLetter, lastLetter)
    count ← 0
    for i←0 to length(text) do
        if (text[i]==firstLetter)
            for j←i+1 to length(text) do
                if (text[j]==lastLetter)
                    count = count +1
                endif
            endfor
        endif
    endfor
    return count
```

Analysis : For the worst case of the text composed of n 'X's, the total number of character comparisons is :

$$n+(n-1)+\dots+2+1=n.(n+1)/2 = \Theta(n^2)$$

Berkcan EKİCİ
171044015

Answer 4: The brute force way is, like one that counts inversions in an array, to calculate the distances of every pair of points in the universe.

Pseudocode:

```
Algorithm ClosestPoints (P)
//Input: A list P of n (n>=2) points P1 = (x1,y1), ..., Pn(xn,yn)
// Output: Indices index1 and index2 of the closest pair of points.

   minVal <- ∞
   for i<-1 to (n-1) do
      for j<-i+1 to n do
         min <- sqrt((xi-xj)^2+(yi-yj)^2)
         if min<minVal
            minVal <- min
            index1 <- i
            index2 <- j
         endif
      end for
   endfor
   return index1, index2
```

Analysis: For n number of points, we would need to measure:

$$\frac{n.(n-1)}{2} = \Theta(n^2)$$

Berkcan EKİCİ
171044015

Answer 5:

a)

Pseudocode:

Algorithm MostProfitableCluster (m)

// Input: A list m of branches.

// Output: Indices index1 and index2 of the range showing the most profitable cluster.

```
startIndex <- 0
endIndex <- 0
totalSum <- 0

for i <- 0 to length(m) do
    sum <- 0
    for (k <- i to length(m) do
        sum += m[k]
        if sum > totalSum
            totalSum <- sum
            startIndex <- i
            endIndex <- k
        endif
    endfor
endfor

return startIndex, endIndex
```

Analysis: For n number of branches, we would need to measure:

$$n + (n-1) + \dots + 2 + 1 = \frac{n.(n+1)}{2} = \Theta(n^2)$$

Berkcan EKİCİ
171044015

b)

Pseudocode:

```
Algorithm FindMaximumProfit(m,left,right)
// Input: A list m of branches, left side, right side
// Output: Result of maximum profit

    if(left = right)
        return m[left]
    mid ← (left+right)/2

    leftMax ← -∞
    sum ← 0

    for i ← mid to left do i=i-1
        sum += m[i]
        if (sum > leftMax)
            leftMax ← sum
        endif
    endfor

    rightMax ← -∞
    sum ← 0

    for i ← mid+1 to right do i=i+1
        sum += m[i]
        if (sum > rightMax)
            rightMax ← sum
        endif
    endfor

    maxLeftRight = max (findMaximumProfit(m,left,mid),
                        findMaximumProfit(m,mid+1,right))

    return max(maxLeftRight, leftMax + rightMax)
```

Analysis: The time complexity of the above divide and conquer solution is $O(n\log n)$ as for the given array of size n, we make two recursive calls on input size $n/2$ and finding the maximum subarray crosses midpoint takes $O(n)$ time in the worst case.

$T(n) = 2T(n/2) + O(n) = O(n\log n)$