CSE321 - Introduction to Algorithm Design
Homework 3

1801042103
Ozan GEÇKİN

1) Derive recurrence relations of the following algorithms and solve them to decide the complexity of the algorithms. Which algorithm would you prefer for same problem, elabrate your answer.

a) Algorithm alg1(A[0,...,n-1])
```
if(n==1) return A[0]
else
  tmp = alg1(A[0,...,n-2])
  if(tmp<=A[n-1]) return tmp
  else return A[n-1]
```

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

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

$\vdots$

$T(1) = T(0) + c$

$T(1) = c$

$n \cdot c \Rightarrow T(n) \in \Theta(n)$

b) Algorithm alg2(x[l,...,r])
```
if(l==r) return x[l]
else
  flr = floor((l+r)/2)
  tmp1 = alg2(x[l,...,flr])
  tmp2 = alg2(x[flr+1,...,r])
  if(tmp1<=tmp2) return tmp1
  else return tmp2
```

$T(n) = 2T(\frac{n}{2}) + c$

Use master Theorem;

$a = 2$
$b = 2$
$c = \Theta(1)$

$\Rightarrow n^{\log_b a} = n \quad n \geq 1$

$T(n) = \Theta(n^{\log_a b}) = \Theta(n^{\log_2 2}) = \Theta(n)$

Note = Both algorithms have the same time-complexity. So it doesn't matter which one I use. But algo1 calls itself one time and another algorithm calls two times itself. So algo1 uses less space. So my preference would be algo1.

---

2) You are given a polynomial $p(x)$ like $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ and you are supposed to write a brute-force algorithm for computing the value of the polynomial at a given point x_0. Analyze the complexity.

```
Algorithm polynomial(x0, arr[]):
  res = 0
  loop i to length arr:
    res = res + (x0 ** i) * arr[i]
  return res
```

$T(n) = \sum_{i=0}^{n} 1 + 1$

$T(n) = n + 1$

$T(n) \in \Theta(n)$

Note = There is only one loop, which runs n times. The for loop includes an operation that is constant time.

3) You are asked to design a brute-force algorithm that counts the number of substring that start with a specific letter and end with another letter in a given text.

```
Algorithm counterSubstring(txt, start, end):
    counter = 0
    loop i to txt.length
        if (txt[i] == start)
        loop j to i, txt.length;
            if (txt[j] == end):
                count = count + 1
return count
```

So, There are two for loops in which they include constant time basic operation

Worst case

$$T_{worst} = \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} (n-1) = \frac{n.(n-1)}{2} \approx n^2$$

$$T_{worst} \leq \Theta(n^2)$$

Best case

If the first letter is not repeated, it will not enter the second loop.

$$T_{Best} = \sum_{i=1}^{n} 1 = n.$$

$$T_{Best} = \Theta(n)$$

---

4) A metric space consist of a set and a distance function. We are given a metric space that is made up of a set of n points in k-dimensional space and Euclidean distance function. Design a brute-force algorithm that gives the distance between the closest pair of points, and find the complexity of the algorithm.

Example points = { $x_0, y_0, \dots, x_{k-1} y_0$; $x_0, y_n, \dots, x_{k-1} y_{n-1}$ } — n points

k - dimensional

```
def Distance (P1, P2):
    res = 0
    loop i = 0 to n-1:
        res = P1[i]^2 - P2[i]^2
        res = res + res
return sqrt(res)
```

```
def BruteClosestDistance(P[0,..n-1]):
    min_value = Sys.MaxNumber
    loop i to P.length:
        loop j to i+1, P.length:
            if (Distance(P[i], P[j]) < min_value)
                min_value = Distance(P[i], P[j])
return min_value.
```

Time Complexity Analysis

Distance

$$\sum_{i=1}^{k} 1 = k = T_{Distance} = \Theta(k)$$

BruteClosestDistance

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} k = \sum_{i=1}^{n} k(n-1) = \frac{k.n.(n-1)}{2} = T_{BruteClosestD} = \Theta(n^2)$$

180104103
Ozan GEGKIN

5) a) Brute-Force Algorithm that can find the most profitable cluster.

1801042103
Ozan GEÇKİN

```
def FindCluster(arr):
    StartIndex=0
    lastIndex = 0
    maxProfit = 0
    res = -sys.maxsize
    loop i to arr.length:
        maxProfit= arr[i]
        loop j to i+1, arr.length
        maxProfit += arr[j]
        if (res < maxProfit):
            res = maxProfit
            startIndex = i
            lastIndex = j
    return res, startIndex, lastIndex
```

Time Complexity Analysis

$$T(n) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n} (n-i) = \frac{n.(n-1)}{2} = n^2$$

$T(n) \in \Theta(n^2)$

Note= There are 2 nested for loops and they have basic operations with constant time.

b) Divide and conquer algorithm that finds the maximum profit.

```
def maximumProfit(arr, start, last):
    if (start == last):
        return arr[start]
    m = (start+last) / 2
    return max(maximumProfit(arr, start, m),
               maximumProfit(arr, m+1, last),
               maxCrossingSum(arr, start, m, last))

def maxCrossingSum(arr, start, m, last):
    temp=0
    left_sum=-sys.maximum
    loop i to (m, start-1, -1):
        temp=temp + arr[i]
        if (temp>left_sum):
            left-sum = temp
    temp=0
    right-sum = -sys.maximum
    loop i to (m+1, last+1):
        temp = temp + arr[i]
        if (temp > right_sum):
            right-sum = temp
    return max(left-sum+right_sum, left_sum, right_sum)
```

Time Complexity

(maxCrossingSum)

$$T(n) = 2T\left(\frac{n}{2}\right) + \underbrace{M(n)}$$

$$M(n) = \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} 1 = 2n = \Theta(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n$$

Using Master theorems

$a=2, b=2, c=2n=n$

$$T(n) = \Theta(n^{\log_b a} . \log n)$$

$$= \Theta(n^{\log_2 2} . \log n)$$

$$= \Theta(n \log n)$$