CSE 321 Homework 3

Ahmet Tuğkan AYHAN

1901042692

Question 1

a) Algorithm alg1(L[0.... n-1]) $\rightarrow T(n)$

```
if (n==1)
    return L[0]              } Θ(1)
else
    tmp = alg1(L[0.....n-2]) } T(n-1)
    if (tmp <= L[n-1])
        return tmp
    else                     } Θ(1)
        return L[n-1]
```

Recurrence Relation

$T(n) = T(n-1) + 1 \quad , \quad T(1) = 0$

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

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

$\rightarrow T(n) = \ldots\ldots$

$\rightarrow T(n) = (\underbrace{T(1)}_{0}+1) + (n-2) \Rightarrow T(n) = n-1 \Rightarrow T(n) \in \Theta(n)$

b) Algorithm alg2(X[ℓ..... r]) $\rightarrow T(n)$

```
if (ℓ==r)
    return X[ℓ]                  } Θ(1)
else
    flr = floor((ℓ+r)/2)         → Θ(1)
    tmp1 = alg2(X[ℓ.... flr])    → T(n/2)
    tmp2 = alg2(X[flr+1 .... r]) → T(n/2)
    if (tmp1 <= tmp2)
        return tmp1
    else                         } Θ(1)
        return tmp2
```

Recurrence Relation

$T(n) = 2T(\frac{n}{2}) + 1 \quad , \quad T(1) = 0$

Master's Theorem

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

a : 2 $\quad * \; n^{\log_b a} = n^{\log_2 2} = n$

b : 2

f(n) : 1 $\quad * \; n > f(n)$, thus we use case 1 in theorem

Case 1 : $T(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n)$

Which one to choose?

* I would choose alg1 because solving a problem with recursion is costly. It is costly because the function is called over and over again. Since alg2 divides the problem into more subproblems it costs more, In the end they both make same amount of comparison

2) $p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$

```
Algorithm alg(a[0...n], x)
    result = a[0]      } Θ(1)
    currX  = 1         }
    for i=1 to n do  → Θ(n)
        currX = currX * x                     } Θ(1)
        result = result + (currX * a[i])      }
    return result  → Θ(1)
```
$\Theta(n)$

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

Can we have a better time complexity?

→ No, we can't Because in order to calculate a polynomial function we have to know every x value from x^n to x^0 and we have to multiply them with their coefficients. Thus, this problem can't be solved lower than $\Omega(n)$ complexity

Question 3

```
Algorithm alg (string) → T(n)
    str   = string                      }
    size  = length of the string        } Θ(1)
    count = 0                           }
Θ(n²) { for i=0 to (size-1) do
  Θ(n) { for j=i+1 to size do
            if ((str[i]=="X") and (str[j]=="Z"))   } Θ(1)
                count = count + 1                  }
    return count
```

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

Question 4

```
Algorithm alg (Array[k][n], k) → T(n)
    d = ∞ → Θ(1)
Θ(n²k) { for i=0 to n-1 do
   Θ(n.k) { for j=i+1 to n do
       Θ(k) { sum = 0 → Θ(1)
              for m=0 to k do
                  sum = sum + (array[m][i] - array[m][j])² → Θ(1)
              d = min (d, sqrt(sum)) → Θ(1)
    return d
```

* $T(n) \in \Theta(k.n^2)$

Question 5

a)
```
Algorithm alg (stations, profit_rates) → T(n)
  size = number of stations → Θ(1)
  start = 0 , end = 0 , curr_profit = 0 , max_profit = 0 → Θ(1)
Θ(n²) { for i=0 to size-1 do
            curr_profit = profit_rates [i]
   Θ(n) {   for j=i+1 to size do
                if (curr_profit > max_profit) do     }
                    start = i                        }
                    end = j+1                        } Θ(1)
                    max_profit = curr_profit         }
  return (stations[start : end]) → Θ(1)
```

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

b) *Algorithm* alg (profit_rates , low, high) $\rightarrow T(n)$

```
if ( high <= low )
        return profit_rates[low]                                   } Θ(1)

mid = (low+high) / 2                                               → Θ(1)

left_max_profit = -∞                                               } Θ(1)
sum = 0
for i=mid to low , decreasing by 1 , do
    sum += profit_rates[i]
    if(sum > left_max_profit)                                      } Θ(n)
        left_max_profit = sum

right_max_profit = -∞                                              } Θ(1)
sum = 0
for i = mid+1 to high do
    sum += profit_rates[i]
    if (sum > right_max_profit)                                    } Θ(n)
        right_max_profit = sum

left = alg(profit_rates, low, mid)                                 → T(n/2)
right = alg(profit_rates, mid+1, high)                             → T(n/2)

return max( (left + right) , (left_max_profit + right_max_profit) )  → Θ(1)
```

Recurrence Relation

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

Master's theorem

$a : 2$ $\quad * \ n^{\log_b a} = n^{\log_2 2} = n$

$b : 2$

$f(n) : n$ $\quad * \ n \Leftrightarrow f(n) \rightarrow n == n$, thus we will use case 2

Case 2 ; $T(n) = n^{\log_b a} . \log n \rightarrow T(n) \in \Theta(n.\log n)$