①

Nevzat Seferoglu
171044024

somefunction(row, cols)	Steps/exec	freq	Total
for(i=1 ; i≤rows ; i++)	2	rows+1	2 * (rows+1)
for(j=1 ; j≤cols ; j++)	2	rows * (cols+1)	2 * rows * (cols+1)
print(*)	1	rows * cols	rows * cols
print(newline)	1	rows	rows
			3 * rows . cols + 5rows + 2

rows = n
cols = m

* There is no conditional statement, for this reason, there is no meaning to measure the T_{Best}, T_{worst}, $T_{average}$ as seperate.

• Let we have $T(n,m)$ as a time-complexity of given function. Thanks to table:

$$T(n,m) = 3mn + 5n + 2$$

• Big-O notation is general purpose notation and useful for measuring upper-bound of an algorithm.

$$T(n,m) \le O(f(n,m))$$

$O(f(n,m)) \to T$ grows no faster than f.
* We can use the simplification of two rule of notations.

Rule 1 : Constants have no effect on algorithm.
Rule 2 : Lower-order term has no effect too according to asymtotic notation.

$T(n,m) = \Theta(f(n,m))$ when $n \ge n_0$ $m \ge m_0$

$T(n,m) = \Theta(3mn + 5n + 2)$
$= \Theta(mn)$ → $\Theta(mn)$,
→ Theta notation of algorithm.

$$c_1 . (n.m) \le 3.(n.m) + 5n + 2 \le c_2.(n.m)$$

$c_1 = 2$
$n_0 = 0$
$m_0 = 0$

$c_2 = 12$
$n_0 = 1$
$m_0 = 1$

* $T_{worst} = T_{Best} = T_{average}$

(2)

```
some function ( a,b)
{
T1(a,b)  if( b == 0 )
             return 1
         answer = a
         increment = a
         for( i=1 ; i< b : i++)
         {
            for( j=1 ; j< a : j++)
            {
               answer += increment
            }
            increment = answer.
         }
         return answer
}
```

step/exe	frequency	Total
1	1	1
1	1	1
1	1	1
1	1	1
2	$(b-1)+1$	$2b$
2	$((a-1)+1)(b-1)$	$2ab-2a$
2	$(b-1).(a-1)$	$2ab-2a-2b+2$
1	$b-1$	$b-1$
1	1	1

Total → $4ab-4a+b+b$

* there is an if condition on the statement so T_{best}, T_{worst}, $T_{average}$ are meaningful.

If(b==0) → T1(n)

$T_{worst}(N) = \max\{T(I)\}$

$T_{av}(N) = \sum_{|I|=N} T(I)\,Pr(I)$

$T_{best}(N) = \min_{|I|=N}\{T(I)\}$

$T_{worst}(N) \geq T_{av}(N) \geq T_{best}(N)$

$T(n) = O(T_{worst}(n)) = \Omega(T_{best}(n))$

$T_{worst}(a,b) = T_1(a,b) + \max(1, 4ab-4a+b+4)$

$= 1 + 4ab-4a+b+4$

$= 4ab-4a+5$ → $T_{worst}(a,b) = \Theta(4ab-4a+5) = \Theta(ab)$

$T_{best}(a,b) = T_1(a,b) + \min(1, 4ab-4a+b+4)$

$= 1 + 1 = 2$ $\quad T_{best}(a,b) = \Omega(1) = O(1) = \Theta(1)$ constant-time

$T_{average}(a,b) = pT_1(T).\min(1, 4ab-4a+b+4) + pT_1(F).\max(1, 4ab-4a+b+4) + T_1(a,b)$

* pT_1 is not known so, we can exactly calculate $T_{average}$.

(3)

```
Somefunction (arr[] , arr_len)
{
    val = 0                                          ------>
    for(i=0 ; i< arr_len/2 ; i++)                    ------>
        val = val + arr[i]                           ------>
    for(i=arr_len/2 ; i < arr_len ; i++)             ------>
        val = val - arr[i]                           ------>
(1) if(val >=0)                                      ------>
(2)     return 1                                     ------>
    else
(3)     return -1                                    ------>
}
```

Steps/exec	Frequency	Total
1	1	1
2	$(\frac{arr\_len}{2}+1)+1$	arr_len
2	$\frac{arr\_len}{2}-1$	arr_len - 2
2	$(\frac{arr\_len - \frac{arr\_len}{2}}{1})+1+1$	arr_len
2	$\frac{arr\_len}{2}-1$	arr_len - 2
1	1	1
1	1	1
1	1	1

* There is conditional statement so T_{worst}, T_{best}, $T_{average}$ are significant for function

① → $T_1(arr, arr\_len) = \Omega(1) = O(1) = \Theta(1)$
② → $T_2(arr, arr\_len) = \Omega(1) = O(1) = \Theta(1)$
③ → $T_3(arr, arr\_len) = \Omega(1) = O(1) = \Theta(1)$

} All of them are in constant time complexity.

• Let we have $T(arr, arr\_len)$ as a time-complexity of given function. We didnot have exact calculation and also notation for function, because There are conditional statements

$T_{best} = (arr\_len - 3) + T_1(arr, arr\_len) + min(1,1)$

$= arr\_len - 3 + 1 + 1$

$= arr\_len - 1 = \Theta(arr\_len) = O(arr\_len)$ //

$T_{worst} = (arr\_len - 3) + T_1(arr, arr\_len) + max(1,1)$

$= arr\_len - 3 + 1 + 1$

$= arr\_len - 1 = \Theta(arr\_len) = O(arr\_len)$ //

$T_{average} = (arr\_len - 3) + P_T \cdot (T_1 + T_2) + P_F \cdot (T_1 + T_3)$

$= arr\_len - 3 + P_T \cdot 2 + P_F \cdot 2 = c \cdot f(arr\_len - 3 + \dots)$

$= \Theta(arr\_len) = O(arr\_len)$

$= O(n)$

(*) There has been no changing because inside of conditional statements are constant time for both two situations.

Somefunction(n)	Steps/exec	freq	Total
c=0	1	1	1
for(i=1 to n*n)	2	(n*n)+1	$2n^2+2$
for(j=1 to n)	2	(n*n)*(n+1)	$2n^3+2n^2$
for(k=1 to 2*j)	2	(n*n)*n*(n+2)	$2n^4+4n^3$
c=c+1	2	(n*n)*n*(n+1)	$2n^4+2n^3$
return c	1	1	1
			$4n^4+8n^3+4n^2+4$

* There is no conditional statement so analyzing T_{best}, T_{worst}, $T_{average}$ is not meaningful.

- Let we have $T(n)$ or a time-complexity of given function.

Big-O

$T(n) \leq c_1 \cdot f(4n^4+8n^3+4n^2+4)$ $\quad c_1>0,\ n>n_0$

$T(n) = O(4n^4+8n^3+4n^2+4)$

$T(n) = O(n^4)$

Rule 1 : constant can be removed
Rule 2 : lower-order terms can be removed

Omega

$T(n) \geq c_2 \cdot f(4n^4+8n^3+4n^2+4)$ $\quad c_2>0,\ n>n_0$

$T(n) = \Omega(n^4)$

$T_{best} = T_{worst} = T_{av} = \Theta(n^4)$

Theta

$T(n) = c_3 f(4n^4+8n^3+4n+4)$ $\quad c_3>0,\ n>n_0$

$T(n) = \Theta(n^4)$

(5) (*) There are two different kinds of functions

(A) otherfunction(xp, yp) → T_A

(B) somefunction(arr[], arr_len) → T_B

T_A → Time-complexity of function A.

T_B → Time-complexity of function B.

Analyzing function A

$T_A(xp, yp) = c_1 \cdot f(3)$ → constant $\Omega(1)$, $O(1)$, $\Theta(1)$

function works in constant time.

Analyzing function B → arr_len = n

```
somefunction(arr[], arr_len)
{
    for(i=0 ; i<arr_len-1 ; i++)
    {
        min_idx=i
        for(j=i+1 ; j<arr_len ; j++)        (Part 1)
        {                                   (Part 2)
            if(arr[j] < arr[min_idx])
                min_idx=j
        }
        otherfunction(arr[min_idx], arr[i])
    }
}
```

Step/s real	Frequency	Total
2	$n-1+1$	$2n$
1	$n-1$	$n-1$
2	$\frac{n.(n+1)}{2}$	n^2+n
1	$\frac{(n-1).n}{2}$	$\frac{n^2}{2}-\frac{n}{2}$
1	$\frac{(n-1).n}{2}$	$\frac{n^2}{2}-\frac{n}{2}$
3	$n-1$	$3n-3$

Worst case also → $T(arr, n) = O(2n^2+6n-4)$

$= O(n^2)$

Part 1

if i=0

j=1 to n-1+1 → n

i=1 j=2 to n-1+1 → n-1

⋮

i=n-1 j=n to n-1+1 → 1

$\sum_{k=1}^{n} k = \frac{n.(n+1)}{2}$

Part 2

if i=0 j=1 to n-1 → n-1

i=1 j=2 to n-1

⋮

i=n-2 j=n-1 to n-1 → 1

i=n-1 j=n to n-1 → 0

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

(6) There are two different function on this question

TA → otherfunction (a,b)

TB → Somefunction (arr, arr-len)

Analyze for function A

```
other function (a,b)
{
    if b==0:
        return 1
    answer = a
    increment = a
    for i=1 to b:
    {
        for j=1 to a:
            answer += increment
        increment = answer.
    }
    return answer.
}
```

$T_1(a,b)$

step/exec	Frequency	Total
1	1	1
1	1	1
1	1	1
1	1	1
2	b+1	2b+2
2	(a+1).b	2ab+2b
2	ab	2ab
1	b	b
1	1	1

$T_{best} = T_1(a,b) + min(1, 4ab+5b+3)$

$= 1 + 1 = 2 = \Omega(2) = \Omega(1) =$ constant - time

$T_{worst} = T_1(a,b) + max(1, 4ab+5b+3)$

$= 1 + 4ab+5b+3 = 4ab+5b+4$

$= O(ab)$

$T_{average} = T_1(a,b) + P_T.(1) + P_F.(4ab+5b+3)$ — P_T and P_F constant

$= 1 + P_T + P_F.(4ab+5b+3)$

$= O(ab)$

```
Somefunction (arr, arr-len)  {arr-len = n}
{
for i=0 to arr-len):
(2) for j=i. to arr-len):
(3)     if otherfunction(arr[i], i) == arr[j]:
            print (arr[i], arr[j])
}
```

step/exec	Frequency	Total
2	n+2	2n+4
2	$\frac{(n+2).(n+3)}{2} - 1$	n^2+5n+4
1	$\frac{(n+1).(n+2).(arr.len)}{2}$	$\frac{n^2+3n+2}{2}.n$
1	$\frac{(n+1).(n+2)}{2}$	$\frac{n^2}{2} + \frac{3n}{2} + 1$

(2) i=0 j=0 to n (n+1) times, n+2

i=1 j=1 to n n+1 = n+1

i=n to n 1+1 = 2

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

(*) there is conditional statement so we can determine TBest, Tworst, Taverage of this function.

$T_{Best} = \frac{3n^2+13n}{2} - 4$ } For if statement is always false.

$= O(n^2) = \Theta(n^2) = \Omega(n^2)$ //

$T_{worst} = 2n^2 + 6n - 4$

$= O(n^2) = \Theta(n^2) = \Omega(n^2)$ // } For if statement is always +ve

$T_{average} = P_T\left(\frac{n^2}{2} - \frac{n}{2}\right) + P_F(0) + \left(\frac{3n^2+13n}{2} - 4\right)$

$= P_T(O(n^2)) + 0 + O(n^2)$

$= P_T$ is real numbers, so $n \longrightarrow$ Result $= O(n^2)$ // average

$= \Theta(n^2)$

③ calculation

$i=1 \rightarrow n$ 이하 1
$i=2 \rightarrow n$ 이하 2
$i=3 \rightarrow n$ 이하 3
⋮
$i=n \rightarrow n$ 이하 n

} Below summing equation explains why.

✱ we don't need to measure T_{Best}, T_{Worst}, $T_{Average}$ because all of them will be exact asymtatic notation, Big-O notation measured below.

$$T_{Worst}^{(n)} = T_{Best} = T(n) = \sum_{i=1}^{\frac{n\cdot(n+1)}{2}} 2\cdot(n\text{이하}i)$$

=

⟶ lower-order term removed
constant removed

(7)

* There are two different function in this question.

TA → other function → represent time_complexity.

TB → some function → represent time-complexity.

Analyze of function A:

```
otherfunction(X, i)        i=n
{
    s=0
(1) for(j=1 ; j<=i ; j=j*2)
        s=s + x[j]
    return s
}
```

Step/exec	Frequency	Total
1	1	1
2	$\log_2 n + 1$	$2\log_2 n + 2$
2	$\log_2 n$	$2\log_2 n$
1	1	1

(1) $j=0 \quad 2^0$

$j=2 \quad 2^1$; Assume $2^k > n$

$j=4 \quad 2^2$

$k > \log_2 n$

$j=k \quad 2^k$

$O(\log_2 n)$ // $= T_A$

```
someFunction(arr[], arr_len)        arrlen=n
{
    for(i=0 ; i<=arr_len-1 ; i++)
        A[i] = otherfunction(arr, i) / i+1
    return A
}
```

Step/exec	Frequency	Total
2	$n+1$	$2n+2$
3	$\log n!$	$3\log n!$
1	1	1

$T_B = 3\log n! + 2n + 3$

$\sum_{k=1}^{n} \log_2 k = \log_2 1 + \log_2 2 + \log_2 3 - - - \log_2 n$

$= \log n!$ //

* There is no need to analyze Twrite, Tbelt, Tinverse for this question Because there is no conditional or branch statement

$c_1 \log n! \le 3\log n! + 2n + 2 \le c_2 \log n!$

$c_1 = 2$, $n_0 = 2$; $c_2 = 4$, $n_0 = 2$ } works

(*) Constant and lower-order terms are ignored.

$T_B \le \Theta(\log n!)$

(8)

(*) There is only one function to analyze which is:

SomeFunction(n)

(*) There are more than one conditional statement so, every conditional statement have to be analyzed one by one.

```
{ if(n<10)   ——————→ TA → time
  return n+10 ——→ TB → time ——→ O(1) = Ω(1) = Θ(1)
```

```
for (i=9 ; i>=2 ; i--)  ——→ (n o/p 1) phase save us from infinite loop right here.
    while (n%i==0)  ——→ here.
        n = n/i
        res = res + j*i
        j *= 10
```

(*) outer for loop has constant time-complexity but it cannot make infinite and due to inner.

T_{Best} if $n<10$ $\Rightarrow T_A + T_B$ $\Rightarrow \Omega(1) + \Omega(1) \Rightarrow \Omega(2) \Rightarrow \Omega(1)$

T_A → constant, T_B → constant

constant time complexity

*We cannot examine the best case and average case of the algorithm There is oscillate situation. For different input create different and irrelivent output.

PART 2

(1)

```
Point {
    x, y
}
```

} Point structure that keeps the coordinates of the point.

(A)

```
getDistance (Point p1 , Point p2) {
    return distance between p1 and p2.
        /* sqrt( pow (x value of p1 - x value of p2) + pow (y value of p1
                - y value of p2) */
```

(B)

```
func (Point givenPoint , Point arr[] ) {
    calculatedDistance = 0
    Point minDistancePoint(0,0)   /* default */
```

(T1)

```
    if( arr is empty)
        forward message to customer
```

(T2)

```
    else
        minDistancePoint = arr[0];
        calculatedDistance = getDistance( givenPoint , arr[0] )
        for ( i to length-1 )
            if( getDistance( givenPoint , arr[i] ) < calculatedDistance ){
                calculatedDistance = getDistance (givenPoint , arr[i])
                minDistancePoint = arr[i]
            }
    }
    print minDistancePoint
}
```

Analyze (A):

It is constant time,

* there is just return

$\Omega(1)$

T1 is not considered, because it is an extreme situation. Array should not be empty.

Analyze (B):

If arr is empty, func cannot work, so I did not consider this situation as best time complexity.

Time T2 will determine the time-complexity of working-algorithm.

$T_{best} = T_{worst} = T_{average}$ because there is no return that can influence the algorithm itself.

* inside of the if statement is constant time complexity so, our algorithm does not depends on whether if works or not.

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

(2) Part 2

Question (A)

```
integer findUniqueLocalMin ( integer arr[] (arr ref) , int array length) {

    integer localmin ;

    for( i=1 ≤ arraylength -1 ) {
        if ( arr[i] ≤ arr[i+1]  88  arr[i] ≤ arr[i-1] ) {
            localmin = arr[i]
            increment i by arraylength ( to make (for loop) stop)
        }
    }

    return (localmin)
}
```

Question (B):

```
void printAllLocalMin ( integer arr(array ref) , int array length ) {

    for ( i=1 ≤ array length -1 ) {
        if( arr[i] ≤ arr[i+1]  88  arr[i] ≤ arr[i-1] ) {
            print localmin arr[i]
            /* there is no incrementation because more than one
               can be local min which is same its previous and next. */
        }
    }
}
```

Analyze (A):

* If local-min located at the beginning of the array, loop work just for one time, it will be best case.

$T_{best} = \Omega(1)$ constant time

* If local-min located at the end of the array, it will be worst case because loop iterate all over the array till the border indicator.

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

* Average is depends on location of local min in array.

$T_{General} = O(n)$

Analyze (B):

* For determine all localmin in array, loop have to iterate all over the array because any local-min located in last index of array.

* Therefore there is no different T_{best}, T_{worst}, $T_{average}$.

* Inside of the if statement cannot change the total complexity of the general algorithm.

$T_{worst} = T_{best} = T_{Average} = \Theta(n)$

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

(3) Part 2

```
void matchWithSumOfArray( inputNumber , array( arr[]) , length of array,
boolean indexFlag)
{
    if (indexFlag : true )
    {
        for( i=0   to   length-1)
        {
            for( j=i   to   length-1)
            {
                if( arr[i] + arr[j] = inputNumber ) {
                    print → inputNumber = arr[j] + arr[i]
                            i = length
                            j = length
                }
            }
        }
        else
        {
            for( i=0   to   length-1)
            {
                for(j=i+1   to   length-1)
                {
                    if( arr[i] + arr[j] = inputNumber ) {
                        print → inputNumber = arr[j] + arr[i]
                                i = length
                                j = length
                    }
                }
            }
        }
}
```

Analyzing Function

* There are two conditional statement. But it cannot affect total complexity of algorithm. If statements, which are inside both else and if, does has same time-complexity

$\sum_{k=1}^{n} k \rightarrow \frac{k(k+1)}{2}$ } both if and else statement has some complexity

$g(n) = \frac{n.(n+1)}{2}$

$T(n) \leq c_1 . g(n)$

$T(n) \leq c_1 . n^2$ $\quad c_1 = 2$ for (n)

$O(n^2)$

$T_{Best} = \Theta(1)$ (If it is found first)

$T_{worst} = O(n)$ (If my answer at last)

Part 2

Question (4):

```
void sumChainOfSequence(integer arr[], integer length, boolean indexFlag)
{
    for(i=0 to length-1)
        matchWithSumOfElements(arr[i], length, true); } Question 3 function
}
```

Analyzing

$$\sum_{i=1}^{length} i^2 = 1+4+9 \cdots + (length)^2$$

$T(N) = O(N^3)$ $T_{Best} = \Theta(1)$ *If it is match to the first element. It will be best situation.

* If it is match at the last.

$T_{worst} = O(n^3)$