CSE 222 Homework 2

Name: Türker Tercan
Student ID: 171044032

Part 1

1.

	steps	freq	total
somefunction(rows, cols)			
{			
for (i=1 ; i<= rows ; i++) →	2	rows + 1	2rows + 2
{			
for (j=1 ; j<= cols ; j++) →	2	rows(cols+1)	2rows.cols + 2cols
print(*) →	1	rows. cols	rows. cols
print (newline) →	1	rows	rows
}			+
}			3rows.cols + 3rows + 2cols + 2

$T(n) = 3n^2 + 5n + 2$

$T(n) = O(n^2)$

Best Case: if rows and cols are 1 it runs only 1 time

Worst Case: runs cols . rows times

2.

```
somefunction(a, b)
{
  if ( b ==0)   } → if b equals to
    return 1    } T1(n)   1 it runs 1 time
  answer = a
  increment = a
  for(i=1 ; i<b; i++) → it runs b times
  {
   for(j=1 ; j<a ; j++) → it runs (b-1). a times
   {
    answer += increment → it runs (b-1)(a-1) times
   }
   increment = answer
  }
  return answer
}
```

(brace grouping the lines from "answer = a" to "return answer": $T_2(n)$)

$T_1(n) = \Omega(1)$

$T_2(n) = O(n^2)$

Best Case

$T_2(n)$ if b is 1 it runs only 1 time

Worst Case

it runs $(b-1)(a-1)$ times

3.

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

Worst Case : It runs arr_len times

Best Case : If arr_len equals to 2 it runs 1 times

4.

```
Somefunction(n)
{
    c=0
    for(i=1 to n*n)
        for(j=1 to n)
            for(k=1 to 2*j)
                c = c+1
    return c
}
```

step	freq	total
1	1	1
2	n^2+1	$2n^2+2$
2	$n^2(n+1)$	$2n^3+2n^2$
2	$n^3\left(2\sum_{k=1}^{n} k+1\right)$	$2n^5+2n^4+2n^3$
2	n^5+n^3	$2n^5+2n^3$
1	1	+ 1

$T(n) = 4n^5+2n^4+6n^3+2n^3+4n^2+2$

$T(n) = O(n^5)$

5.

```
other-function(xp, yp)
{
    temp = xp
    xp = yp
    yp = temp
}
```

Some constant k

$T(n) = \frac{arr\_len^2 - arr\_len}{2}$

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

Best Case : array sorted in ascending order

Worst Case : array sorted in descending order

```
some-function(arr[], arr_len)
{
    for(i=0; i<arr_len-1; i++)   → runs arr_len times
    {
        min_idx = i
        for(j=i+1; j<arr_len; j++)   → runs Σ_{i=0}^{arr_len-1} arr_len-i-1
            if(arr[j] < arr[min_idx]
                min_idx = j
        other-function(arr[min_idx], arr[i])
    }
}
```

$= \frac{(arr\_len-1)\,arr\_len}{2}$

$= \frac{arr\_len^2 - arr\_len}{2}$

6.

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

Best Case : if b is equals to zero it runs 1 time

Worst Case : it runs (a-1)(b-1) times

$T_1(n) = O(n^2)$

steps	freq	total
1	1	1
1	1	1
2	b-1	2b-2
2	(b-1)(a-1)	2ab-2a-2b+1
1	b-1	b-1
1	1	1

$T_1(n) = 2ab + b - 2a + 1$

```
some-function ( arr[], arr_len)
{
    for i=0 to arr_len): → runs arr_len +1 times
      for j=1 to arr_len): → runs Σ_{i=0}^{arr_len} arr_len - i - 1   } T2(n)
        if ( other function (n % i,2) == arr[i]) : → T2(n) . T1(n) times
          print(arr[i])
}
```

(in otherfunction first loop runs only one time)

$T_2(n) = \frac{(arr\_len - 1)\, arr\_len}{2} = O(n^2)$

$T(n) = T_2(n) \cdot T_1(n) = O(n^2) \cdot O(n)$

$= O(n^3)$

7.

```
other-function(X, i)
{
    s=0
    for ( j = 0 ; j <= i ; j = j *2)
        s = s + X[j]
    return s
}
```

j	$2^k >= i$
1	$2^k > n$
2	$k = \log_2 n$
2^2	$O(\log_2 n) = T_2(n)$
2^3	
i	
2^k	

```
some-function( arr[], arr_len)
{
    for( i=0; i<= arr_len -1; i++)   → runs arr_len times
      A[i] = otherfunction (arr, i) / (i+1)
    return A
}
```

$T(n) = T_1(n) \cdot T_2(n)$

$T(n) = O(\log_2 n) \cdot O(n)$

$= O(n \log_2 n)$

8.

```
Somefunction(n)
{
  res=0
  j=1
  if(n<10)
    return n+10

  for(i=9; i>1; i--)
    while(n%i==0)
      n=n/i
      res = res + j*i
      j *= 10

  if(n>10)
    return -1
  return res
}
```

Best Case : If n is lesser than 10 it runs only 1 time

$T(n) = O(1)$

Worst Case : Worst case can not be calculated because while loop runs how many times the n can be divided into different digits.

Part-2

1.

```
fun( xpos[], ypos[], target_x, target_y, size)
{
  x = xpos[0] - target_x
  y = ypos[0] - target_y

  min = sqrt( x*x + y*y)
  index=0
  for( i=1 ; i<size ; i++)
  {
    x = xpos[i] - target_x
    y = ypos[i] - target_y

    distance = sqrt( x*x + y*y)
    if ( distance < min)
    {
      min = distance
      index = i
    }
  }
  return index
}
```

Step	freq	total
2	1	2
2	1	2
k	1	k
1	1	1
2	size+1	2size+2
2	size	2size
2	size	2size
k	size	k.size
1	size	size
1	size	size
1	size	size
1	1	1

$T(n) = size(9+k) + k + b$

Algorithm Complexity : Takes first elements in array like min distance and compares every element in two array and finds closest one

$T(n) = n(9+k) + k + b$

$T(n) = g(n)$

2.

```
fun_a( arr[], size)
{
    for( i=1; i < size - 1; i++)
    {
        if(arr[i] <= arr[i+1])
            if(arr[i] <= arr[i-1])
                return i
    }
}
```

→ Runs size-1 times
$T(n) = O(n)$

Algorithm Complexity : Search all elements in array and if an element smaller than previous and next one return it

In part b, just store all indexes and returns indexes

```
fun_b( arr[], size)
{
  count = 0
  for( i=0 ; i<size-1; i++)
  {
    if( arr[i] <= arr[i+1])
      if( arr[i] <= arr[i-1])
        temp[count++] = i
  }
  return temp
}
```

3.

```
fun( arr[], arr_size, target)
{
  for( i=0; i< arr_size; i++)
  {
    a= arr[i]
    for(j=i ; j < arr_size; j++)
    {
      b= arr[j]
      if( a+b == target)
        return 1
    }
  }
  return 0
}
```

→ runs arr_size +1 time

→ runs arr_size time

→ runs $\sum_{i=0}^{arr\_size} n-1-2 = \frac{(arr\_size-2)(arr\_size-1)}{2}$

$T(n) = \frac{arr\_size^2 - 3\,arr\_size + 2}{2}$

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

Best Case : The sum of first two elements' in array equals to target

Worst Case : The sum of the two elements in array is not equal to target

4.

```
is_sequence(arr[], n)
{
  for(i=1; i<n ; i++)  ---- Θ(n)
  {
    if( arr[i-1] >= arr[i])
      return 0
    if( !fun( arr, n, arr[i])  ---- Θ(n²)
      return 0
  }
  return 1
}
```

$T(n) = \Theta(n) \cdot \Theta(n^2)$

$= \Theta(n^3)$

Algorithm Complexity:

Checks all elements if it is sorted in ascending order and if all elements in array is the sum of two elements in array