HW02

161044123

MOHAMMAD ASHRAF YAWAR

1)

```
somefunction(rows, cols){
  for(i = 1; i <= rows; i++){          O( rows)
    for( j = 1; j <= cols; j++)        O( rows)  * O(cols)
      print(*)                         O( rows) * O(cols)
    print(newline)                     O( rows)
  }
}
```

Ω --→ beset case or lower bound
O --→ worst case or upper bound
θ --→ average bound

list of the asymptotic functions and their ordres:

$1 < \log^{n} < \sqrt{n} < n < n \log^{n} < n^{2} < n^{3} < \ldots < 2^{n} < 3^{n} < \ldots < n^{n}$

so for this nested loops Ω(1) is the best case and O(n) = rows * cols = (rows * cols) and θ(n) = rows * cols therefor n = rows and m = cols
T(n) = θ(n * m)

TABLE METHOD

	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
		(rows*cols)+2rows+2+2rows*cols+2+rows = rows * cols	

2)

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
	steps/exec	**freq**	**total**

	1	1	1
	1	1	1
	1	1	1
	1	1	1
	2	b+1	2(b+1)
	2	b*(a+1)	2ba
	2	b*a	2b*a
	1	b	b
	1	1	1
	2b+2+2ba+2ba+b+1+1+1+1 == >> T(n) = θ(a * b)		

Explanation:

best case if omega of 1 and the worst case is O(b*a) and we can conclude that T(n) = θ(a * b) if we condisder a = n and b = m T(n) = (n*m)

3)

1
val = 0
 	
for (i = 0; i < arr_len / 2; i++)
  val = val + arr[i]
 	
for (i = n / 2; i < arr_len; i++)
  val = val – arr[i]
	steps/exec	**freq**	**total**

	1	1	1
	2	arr_len/2	2(arr_len/2)
	2	2*arr_len	4*arr_len
	2	arr_len+1	2(arr_len+1)
	2	arr_len	2*arr_len

if (val >= 0)
 return 1
 	
else
 return –1
	1	1	1

	1	1	1
	1	1	1
	1	1	1
	1+arr_len+4*arr_len+2*arr_len+2+2*arr_len+1+1+1+1 = >> arr_len so T(n) = O(arr_len = n) = θ(n)		

4)

	steps/exec	**freq**	**total**

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

	2	n*n	$2n^2$
	2	n^2*n	$2n^3$
	2	n^3*n	$2n^4$
	2	n^4	$2n^4$
	1	1	1
	$1+2n^2+2n^3+2n^4+2n^4+1$ = $T(n) = O(n^4) = \theta(n^4)$		

5)

other function:

	steps/exec	**freq**	**total**
temp = xp xp = yp yp = temp	1	1	1
	1	1	1
	1	1	1
	$T(n) = \theta(1)$		

Somefunction:
arr_len = n

	steps/ exec	**freq**	**total**

for (i = 0; i < arr_len - 1;i++)
  min_idx = i
  for (j = i+1; j < arr_len; j++)
	2	(n-1)+1 = n	2n

	1	(n-1)	(n-1)
	2	n(n+1)/2	2(n(n+1)/2)

	1	1	1
if (arr[j] < arr[min_idx]) min_idx = j	1	n(n-1)/2	n(n-1)/2
	1	n * 1	n
otherfunction(arr[min_idx],arr[i])	2n + (n-1) + 2*n(n-1)/2 + 1 + n(n-1)/2 + n = T(n) = $\theta(n^2)$		

Explanation:

→ for first loop statement we have I = 0 to arr_len - 1 and (arr_len-1 +1) as execution so in the last time it check and do not enter the loop so for first loop we have O(n)
→ for the second loop we have j starting from i+1 and to arr_len and If we trace this we get:

I	j
0	1 to n-1+1 → n
1	2 to n-2+2 → n-1
.	.
.	.
.	.
n-1	n-1+1 → n to n-1+1 = n

so we have for j :
1 + 2 +3 ...+n
n(n+1)/2

6)

```
otherfunction(a, b){

        If b == 0:            → 1
                return 1      → 1
        answer = a            → 1
        increment = a         → 1

        for i = 1 to b:{      → b

                for j = 1 to a:            → a
                        answer += increment     → a*b

                increment = answer         → b
        }

        return answer         → 1
}
```

O(a*b)

somefunction(arr, arr_len){

for i = 0 to arr_len): → (arr_len+1)
for j = i to arr_len): → (n (n-1)/2)
if otherfunction(arr[i], 2) == arr[j]: → $O(a*b)*(n^2)$
print(arr[i], arr[j])
elif otherfunction(arr[j], 2) == arr[i]: → $O(a*b)*(n^2)$
print(arr[j], arr[i])
}

if we consider arr_len as arr_len = n and a= m and b = o we have:

$\Omega(n) = \Omega(1)$ and $O(n) = (n^2*m*o)$

$T(n) = \theta(n^2*m*o)$

7)

otherfunction(X, i){
s = 0 → 1
for(j = 1; j <= i; j=j*2) → $O(\log^n)$
s = s + X[j] → $O(\log^n)$
return s → 1
}

→ $O(\log^n)$

if we say arr_len = n

somefunction(arr[], arr_len){

for(i = 0; i <= arr_len-1; i++) → n
A[i] = otherfunction(arr, i) / (i + 1) → $\log^n$
return A → 1
}

→ $O(n* \log^n)$

so in total we conclude that :

$T(n) = O(n\log^n)+1$ == $T(n) = \theta(n \log^n)$

8)

```
somefunction(n){

        res = 0                                  1
        j = 1                                    1

        if(n < 10)
                return n + 10                            1

        for(i = 9; i > 1; i--)                           O(9) = O(1)
                while (n % i == 0)                         O(4) = O(1)        O(1)
                        n = n / I                        1
                        res = res + j * I                      1
                        j *= 10                          1

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

$T(n) = \theta(1)$

PART02

1)

Assume you have an array of points in 2d space. Find the closest point in the array to a given point.

Example:

	X0	**X1**	**X2**	**X3**	**X4**
Y0	*				*
Y1	*		*		
Y2				*	
Y3		*			*
Y4				*	

To find the closest point:

---→ we can make use of brute force algorithm which will cost $O(n^2)$

---→ we can make use of divide and conquer algorithm which will cost $O(n\ log^n)$

so the best Asymptotic choose which be the second one divide an conquer algorithm:

using java programming language:

pseudo code of findClosestNeighborhood() :

– get points 2d array [][]points = {filled with set of points}
– get coordinate 1d array consisting of two x and y coordinate values   [] coordinate = {1,2}
– set the 0^{th} element of point array to a temp 1d array   []closestPoint = points[0]
– find the distance between 0^{th} point on the 2d array and coordinate points using :

d(P, Q) = p(x2 − x1)2 + (y2 − y1)2 {Distance formula}

– assign the found distance to a variable   ClosestDistance = distance(coordinate[0],coordinate[0],closestPoint[0],closestPoint[1])
– traverse all the points in 2d array until the end of it's length:

for(i to points.lenght)
– call distance funntion and assign it to closestDistance
distance = distance(coordinate[0],coordinate[0],points[0][i],point[1][i])
– if distance < closestDistance and distance !=0 then
closestDistance = distance
closestPoint = points[i]

-return closestPoint

distance function pseudo code:

– get variables a1,b1,a2,b2 as order
– calculate power of (a2 – a1,2) and assign it to x
– calculate power of (b2 – b1,2) and assign it to y
– return square root of(x+y)
formula used to this function is :
d(P, Q) = p(x2 − x1)2 + (y2 − y1)2 {Distance formula}

– >>complexity of the function power is O(1) and for square root it is O(n) so total complexity of the function distance is O(n)

– >> complexity of the function findClosestNeighborhood() is O(n) so total complexity of the hole program would be O(n) *O(n) == $O(n^2)$

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

2)

The ith element of an array A is a local minimum if, A[i] <= A[i+1] and A[i] <= A[I-1].

a: Find a local minimum in a given array A
b. Find all local minimums in a given array A.

so to find the local minimum of the given array we can make use of linear search which will cost O(n) but the more efficient way would be using binary search which takes $O(\log^n)$ time complexity.

Pseudo code for finding all local minimums of a given array using binary search:

– get the array with elements in it
– set a flag = true
– call binary search function and assign it's value to x:
– print x

-inside binary search function named **findMinimal()** to find the first occurrence of local minimal and return it to the calling function:

– get array
– get it's starting index as **s**
– get it's ending index(in this case it is the length of given array) **e**
– get the length of the array as **n**
– find index of middle element from the list and assign it to variable mid
– Compare middle element with its neighbours
– if neighbours exist then
  – return mid
– else if middle element is not minima and its left neighbour is smaller than it, then left half must have a local minima
  – return **findMinimal(array,s,mid – 1 ,n)**
-If middle element is not minima and its right neighbour is smaller than it, then right half must have a local minima.
- return **findMinimal(array, mid + 1, e, n)**

– start while loop (if flag is true then):
– set the starting point **s** as **x+1** index for the next call of **findMinimal()** search function in loop
– call binary search function and assign it's value to x
– x is the index of the first occurrence of minimal number in the given array
– print new x
– if x is smaller than the length of array then:
  – set flag = false // this meant that we have reached to the end of the list and no more element to be traversed in the given array.

3)
Find if a given array of integers contains two numbers whose sum is a given number b.

Solution:

we can make use of sorting algorithm for this which is going to take time complexity of $O(n^2)$
but we can also make use of hashing algorithm which is more efficient than sorting .
Hashing algorithm take O(n) time to solve this problem so we use hashing algorithm here:

pseudo code for hashing algorithm of this problem:

– get the array
– get the number b
– initialize an empty hash table x
– start the loop until the array.lenght
    –set temp as b – array[i]
    – if s contains the element temp then:
        – print array[i] + temp as the found pair of elements in the given array
    – push the array[i] into hash table x

THE TIME complexity of this algorithm is $T(n) = \boldsymbol{\theta}(n)$

4)
I claimed to use linear search algorithm here:

pseudo code for this algorithm:
– start loop until the end of array
    – if arr[i-1]*2 is equal to arr[i] then:
        – increment loop variable
    – else
        – start while loop:
            – start for loop j = counter until j < i:
                – if arr[counter] + arr[j+1] == arr[i] then:
                    – stop while loop and go to first loop increment i

CODE SAMPLE :

```
public class ShortestAdditionChain{

	public static void main(String arg[]){

		int array[] = {1, 2, 3, 4, 7, 8, 16, 32, 39, 71};
		int current = array[0],counter;
		boolean flag;

		for (int i = 1;i < array.length ;i++){

			if(current*2 == array[i]){
				System.out.println(current +","+current);
				current = array[i];
			}
			else{

				flag = true;counter = 0;
				while(flag == true && counter < i){
					for(int j = counter;j < i ;j++){
						if(array[counter] + array[j+1] == array[i]){
							System.out.println(array[counter]

							+","+array[j+1]);
							flag = false;
						}
					}
					counter++;
				}
				current = array[i];
			}
			System.out.println(current);
		}
	}
}
```