CSE 222 - Spring 2020

Homework 2 Solutions

171044098
Akif Kartal

Part 1

1)

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

Statement	steps/exec	freq	Total
for(i = 1; i <= rows; i++)	2	rows+1	2rows+2
for(j = 1; j <= cols; j++)	2	rows(cols+1)	2rows.cols+2rows
print(*)	1	rows.cols	rows.cols
print(newline)	1	rows	rows
			3rows.cols + 5rows + 2

Analysis

Let say rows = n, cols = m so,

* $f(n) = 3nm + 5n + 2$

* Since we don't have any condition we don't have best, worst and average case.

* General Running Time

$$T(n) = \Theta(f(n)) = \Theta(nm)$$

2)

```
Somefunction(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
}
```

Statement	Steps/exec	freq	Total
if(b==0)	1	1	1
return 1	1	1	1
answer = a	1	1	1
increment = a	1	1	1
for(i=1; i<b; i++)	2	$(b-1)+1$	$2b$
for(j=1; j<a; j++)	2	$b-1(a-1+1)$	$2ba-2a$
answer += increment	1	$(b-1)(a-1)$	$ba-b-a+1$
increment = answer	1	$b-1$	$b-1$
return answer	1	1	1
			$3ba+2b-3a+5$

Analysis

Let say $n=a$, $m=b$ so;

- $T_{best}(n,m) = \Theta(1)$ (since we have a condition we have best case)
- $T_{worst}(n,m) \Rightarrow f(n,m) = 3mn+2m-3n+5 \Rightarrow \Theta(f(n)) = \Theta(nm)$
- $T_{avg}(n,m) \Rightarrow f(n,m) = 3mn+2m-3n+5 \Rightarrow O(f(n)) = O(nm)$

General Running Time

$T(n,m) = O(nm)$

$T(n,m) = \Omega(1)$

3)

Let say m = arr_len

```
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]
    if (val >= 0)
        return 1
    else
        return -1
}
```

steps/exec	freq	Total
1	1	1
2	$\frac{m}{2}+1$	$m+2$
2	$\frac{m}{2}$	m
2	$((m-1)-\frac{m}{2})+1$	m
2	$\frac{m-2}{2}$	$m-2$
1	1	1
1	1	1
1	1	1
1	1	1
		$4m+5$

Analysis

Let say $n = m$ so $f(n) = 4n+5$

- Since we don't have branch because of if condition we don't have best, worst and average case.
- General Running Time
- $T(n) = \Theta(f(n)) = \Theta(n)$

Akif Kartal

4)

```
Some function (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
}
```

steps/exec	freq	Total
1	1	1
2	n^2+1	$2n^2+2$
2	$n^2(n+1)$	$2n^3+2$
2	$n^2(n.(n+1))$	$2n^4+2n^3$
1	$n^2.n.n$	n^4
1	1	1
		$3n^4+4n^3+2n^2+6$

Analysis

- Since we don't have any if statement we don't have
- best, worst, average case.
-

General Running Time

if we count number of operation for both inner for loop since last loop depends on second loop we will see number of operation like this; 2, 4, 6, 8, 10....2n this gives us $\boxed{n.(n+1)}$ formula so from both inner for loop we get $\boxed{n^2+n}$ then if we multiply this with $\boxed{n^2}$ because of first loop we will get n^4+n^3 number of operation. So;

$f(n) = n^4+n^3$

$$\boxed{T(n) = \Theta(f(n)) = \Theta(n^4)}$$

5)

```
otherfunction(xp, yp)
{
    temp = xp
    xp = yp
    yp = temp
}

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

Let say $n = arr\_len$

if we trace the code.

i	j	Number of repeat of second loop
0	1	$n-1$
1	2	$n-2$
2	3	$n-3$
⋮	⋮	⋮
		2
$n-3$	$n-2$	
$n-2$	$n-1$	1

We see from trace table number of repeating for the loop is,

$1+2+3+4+ \dots n-2+n-1$

Our formula is

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

Note = This is not table method. Just to show.

Analysis

We saw that number of operation for this code is

$$f(n) = \frac{n^2-n}{2}.c$$ (c is a costant number) so,

Since we don't have any if statement we don't have any best, worst or average case.

General running time

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

Note = I didn't consider basic statements such as if, assignment, adding etc. and line which is calling other function because otherfunction contains basic statements. We don't need to concern about it.

Akif Kartal

6) otherfunction(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
}
Somefunction(arr, arr_len)
{
    for i=0 to arr_len):
        for j=i to arr_len):
            if otherfunction(n%(i+1),2)== arr[i]:
                Print(arr[i])
}
```

Let say n = arr_len and a

if we trace the code

i	j	number of operation in otherfunction	total number of operation
0	$n+1$	$2n$	$(n+1)(2n)$
1	n	$2n$	$n(2n)$
2	$n-1$	$2n$	$(n-1)(2n)$
3	$n-2$	$2n$	$(n-2)(2n)$
⋮	⋮	⋮	⋮
$n-1$	2	$2n$	$2(2n)$
n	1	$2n$	$1(2n)$

We see from trace table number of repating is

$$2n(\underbrace{1+2+3+\dots n}+n+1)$$

$$2n\left(\frac{n.(n+1)}{2}+n+1\right)$$

$$2n\left(\frac{n^2+3n+2}{2}\right) = n(n^2+3n+2)$$

$$= n^3+3n^2+2n$$

Note = b is always 2.

Analysis

We saw that number of operation for this code;

$f(n) = n^3+3n^2+2n$

- Since we don't have any branch in the code (we have but it doesn't work always) we don't have best, worst, average cases.

General Running Time

$T(n) = \Theta(f(n)) = \Theta(n^3)$

7)

```
otherfunction(X, i)
{
    s = 0
    for(j = 1; j <= i; j = j*2)
        s = s + X[j]
    return s
}
somefunction(arr[], arr_len)
{
    for(i = 0; i <= arr_len-1; i++)
        A[i] = otherfunction(arr, i)/(i+1)
    return A
}
```

Let say n = arr_len
if we trace the code (we ignore constant operations)

i	Number of operation in otherfunction	Total Number of operations
0	0	0
1	1	1
2	2	2
3	2	2
4	3	3
5	3	3
⋮	⋮	⋮
8	4	4
⋮	⋮	⋮
16	5	5
⋮	⋮	⋮
2^k	n	n

Note: In somefunction loop we get; n = arr_len

$0 + \log_2 1 + \log_2 2 + \dots + \log_2 (n-1)$

$= \log_2(1.2.\dots(\text{arr\_len}-1)) = \log_2((n-1)!)$

Observation

We see that our method has $f(n) = \log_2((n-1)!)$ (n = arr_len)

Analysis

Assume n = arr_len, $f(n) = \log_2((n-1)!)$

General running time

$$T(n) = \Theta(f(n)) = \Theta(\log_2(n!))$$

Akif Kartal

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
}
```

Observation= When we look at the code we see that for n values which is less than or equal to 9 we will constant running time which is best case. But for n values which is bigger than or equal to 10 our operation numbers depends on input value so we get n operation number. if we calculate operation numbers with basic operations. We get

$f(n) = n.c$ operation. (c is constant)

Analysis $f(n) = n.c$

$T_{best}(n) = \Theta(1)$ $(n<10)$

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

$T_{avg}(n) = O(f(n)) = O(n)$

General running time

$T(n) = O(f(n)) = O(n)$

Part-2

1) Assume you have an array that each element has x and y information of points in 2D space and you can reach them just O(1) time without any loop.

```
somefunction(arr[], arr_len, x, y)
{
    min = getDistance(arr[0], x, y)
    index = 0
    for(i=0; i< arr_len; i++)
    {
        if(getDistance(arr[i], x, y) < min)
            min = getDistance(arr[i], x, y)
            index = i
        }
    }
    return index
}

getDistance(element, x, y){
    return sqrt(pow(element.x-x), 2) + pow((element.y-y), 2)
}
```

Note = we assume sqrt and pow method has O(1) complexity.

Analysis

Let say n = arr_len, since our code contains only one loop and constant (ordinary) operations such as assingment etc. Our analysis is

General Running Time

$T(n) = O(n)$

(n = arr_len)

Note = We don't consider constant (ordinary) operations when we use asymptotic notations, therefore we only interested in loops. if, assignment, return etc. we don't care them because they have constant running time O(1).

Akif Kartal

2) a)

```
Somefunction( arr[], arr_len)
{
    for(i=0; i<arr_len; i++)
    {
        if(i!=0 && i!=arr_len-1){
            if(arr[i]<=arr[i+1]){
                if(arr[i]<=arr[i-1]){
                    return arr[i]
                }
            }
        }
    }
}
```

Analysis

Let say n= arr_len so, our code contains only one loop and some constant statements therefore, our complexity is;

$T_{best}(n) = \Theta(1)$ (if we find directly)

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

$T_{avg}(n) = O(n)$

$T(n) = O(n)$

Note =

I explained why I used just n in first question. in back page, in note part.

b)

```
somefunction(arr[], arr_len, arr2[])
{   k=0;
    for(i=0; i<arr_len; i++)
    {
        if(i!=0 && i!=arr_len-1){
            if(arr[i]<=arr[i+1]){
                if(arr[i]<=arr[i-1]){
                    arr2[k]=arr[i]
                    k++
                }
            }
        }
    }
}
```

Analysis

Let say n= arr_len so, our code contains only one loop and some constant statements hence, our complexity is;

General Running Time

$f(n) = n$

$T(n) = O(f(n)) = O(n)$

Note = Reason is in back page (Note).

3)

```
myAlgorithm(arr[], arr_len, b)
{
    for(i=0; i<arr_len-1; i++){
        for(j=i+1; j<arr_len; j++){
            if(arr[i]+arr[j]==b){
                return 1
            }
        }
    }
    return 0
}
```

Let say $n = arr\_len$

if we count number of operations

i	j	Number of operation
0	1	$n-1$
1	2	$n-2$
2	3	$n-3$
⋮		⋮
$n-3$	$n-2$	2
$n-2$	$n-1$	1
$n-1$	n	0

Analysis

According to number of operation we have $0+1+2+3+\dots+n-2+n-1$ operation. In this case our general formula is:

(c is a constant number from ordinary statements)

$$\frac{n.(n+1)}{2} - n = \frac{n^2-n}{2}.c$$

$$f(n) = \frac{n^2-n}{2}.c$$ so,

$T_{best}(n) = \Theta(1)$ (if we find directly)

$T_{worst}(n) = \Theta(f(n)) = \Theta(n^2)$ (we ignore lower order terms and constants)

$T_{avg}(n) = O(f(n)) = O(n^2)$

General running time

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

Akif Kartal

4)

```
somefunction(arr[], arr_len){
    flag = 0;
    result = 0
    for(k = 1; k < arr_len; k++){
        result = myAlgorithm(arr, k, arr[k])
        if(result == 0){
            flag = 0;
            for(int i = 0; i < k; i++){
                if(arr[i] + arr[i] == arr[k]){
                    flag = 1
                }
            }
        }
        if(flag == 0)
            return 0
    }
}
}
}
```

Note = Since in myAlgorithm we don't check sum of element itself we need to second loop to do this.

Analysis Let say $n = arr\_len$

From previous algorith we have $\frac{n^2 - n}{2} \cdot c$ number of operation but in here since J starts from I+1 so our number of operation is

$$\frac{n^2 - n}{2} \cdot n + n \cdot n = \frac{n^3 - n^2 + n^2}{2} = \frac{n^3 + n^2}{2}$$

we have from somefunction method, therefor our Analysis is

$$f(n) = \left(\frac{n^3 + n^2}{2}\right)$$

$\frac{n^2 - n}{2}$ from myAlgorithm

$n \rightarrow$ first loop

$n \rightarrow$ inner loop

Continue

- $T_{best}(n) = \Theta(1)$ (if we don't find in first element)
- $T_{worst}(n) = \Theta(f(n)) = \Theta(n^3)$

$T_{avg}(n) = O(f(n)) = O(n^3)$

General Running Time

$T(n) = O(f(n)) = O(n^3)$

Akif Kartal