Esra Eryilmaz

CSE222 - Spring 2020 Homework 2 03.13.2020

P	'ART	1
_		

somefunction (rows, cols)	Freq	Total
for (i=1; iz=rows; i++)> 2	r+1	2.(1+1)
for $(J=1; J <= cols; J++) \longrightarrow 2$ print (*) print (newline) 1	r. (c+1)	2. (rc+r) 1. rc + 3rc + 5r + 2

The for loop with i as its index will execute r (rows) times Inside this loop, the for loop with J as its index will execute c (cols) times. Thus the total number of times is r.c times.

- It depends on rows and columns.
- Constants can be ignored - Ignore the low order terms like "5r"

2	somefunction (a,b)	Step	Freq	Total
	if $(b==0)$ return 1	1	λ	1
	answer = a> increment = a> foc(i=1:iv b : i+b)>	1 1	1 1 b	Д Ь
	for $(i=1; i < b; i++)>$ for $(j=1; j < a; j++)>$	1	a. (b-1)	ab-a
	answer + = increment>	1	(a-1).(b-1)	ab-a-b+1 -
	increment = answer>	1	(6-1)	λ
	return answer>	1	1 ± 2	ab-2a+b+4

- Constants can be ignored.

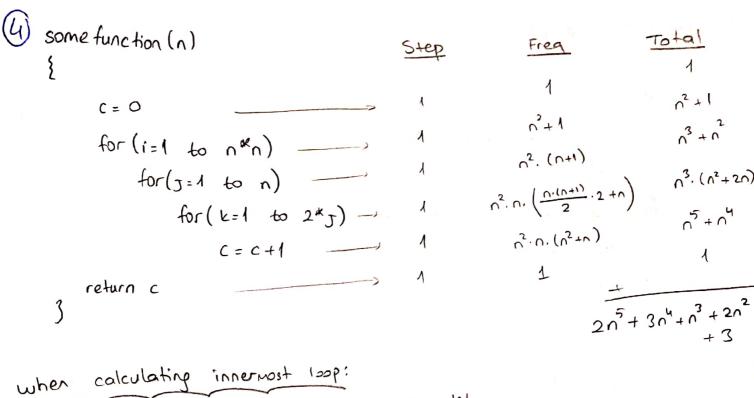
0(1)

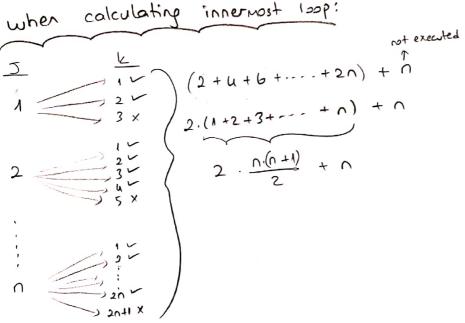
- If the nested loops contain sizes a and b, the cost is O(a.b)

Best-case If b equals zero then function returns 1.	in worst-case runs 206-20 +6+4	times.
So in best-case runs only 1 time.	0 (a·b)	

3) somefunction (arr[], arr_len)	Step	Freq	Total
val = 0	, 1	1	2. (<u>~</u> 1)
for (i=0; i< arr_len/2; i++)	2	$\frac{6}{2}$ + 1 $\frac{6}{2}$	3 0 7
for (i= arr_len/2; i <arr_len; -="" arr[i]<="" i++)="" td="" val="val"><td>) 2</td><td>$\frac{n}{2}+1$ $n/2$</td><td>$2 \cdot \left(\frac{C_2}{2} + 1\right)$ $3 \cdot \frac{C_2}{2}$</td></arr_len;>) 2	$\frac{n}{2}+1$ $n/2$	$2 \cdot \left(\frac{C_2}{2} + 1\right)$ $3 \cdot \frac{C_2}{2}$
if (val >=0) return 1		1	1
else return -1			5n+7
- If the for loop takes of time	and i	increases	or decreases
by a constant, the cost is	<u>, </u>		
- The worst case and the best o	case scen	arios are t	the same.
Because in all cases loops work.	(0(n))		

Function growth rate is linear





- If the first loop runs n^2 times and the inner second loop runs n times and the third loop runs $n^2 + n$ times, then the cost is $O(n^5)$.

- The worst case and the best case scenarios are the same.

Because in all cases all three loops work. (O(n5))

```
other-function (xp, yp)
                    All of them constant
                                                           arr_len=r
                                           0(1)
    some function (arr[], arr_len)
        for (i = 0; ix arr_len-1; i++) ) o(n)
             min_idx=i ) 0(1)
             for (5=i+1; J < arr_len; J++)
            o(1) (if (arr[J] < arr[min_idx])

min_idx = J
            otherfunction (arr [min_idx], arr [i]) > O(1)
 - In somefunction there are loops. The rested loops contain
sizes n and n; the cost is O(n \cdot n) = O(n^2).
-Inside that nested loops, there is a function call which
cost is constant (O(1)). And also there are some constant
operations (Also o(11).
```

- So the total cost is O(n2). (Quadratic)

So time complexity stays same (O(n2))

- In any cases (Best-worst) The loops are execute.

```
otherfunction (a,b)
       if b == 0;
           return 1 }
       answer = a
       increment = a
       for i=1 to b: 10(b)
           increment = onswer
                                                        O((n\%i).2) = O(2n) = dn
         return onswer
     5
   somefunction (arr, arr_len)
       for i=0 to arr_len): ) D(n)
          for j=i to arr_len):) O(n)
              if other-function (arr-len %i, 2) = = arr[i]:
                   ([i] ra) tong
                                                       bis not equal to O.
                                                b is always 2. so function cannot
Best and worst cases
                                                runs constant time. It will execute
It will execute always O(n·n·n)=O(n3)
                                                 always
times.
Function growth rate is cubic.
```

other function (x,i) s=0 for(J=1;Jz=i;J=J*z) $O(log_2i)$ s=s+X(J)return s s=s+X(J) $for(i=0;iz=arr_len-1;i++) O(n)$ $for(i=0;iz=arr_len-1;i++) O(n)$ $for(i=0;iz=arr_len-1;i++) O(n)$ $for(i=0;iz=arr_len-1;i++) O(n)$ $for(i=0;iz=arr_len-1;i++) O(n)$ $for(i=0;iz=arr_len-1;i++) O(n)$

- Some function first loop runs n times. And function calls another function, which name is other function. It will runs logn times.

The cost is O(nlogn).

Function growth rate is Log-linear.

- Best and worst cases same the total cost. (o(nlogn))

(8) somefunction (n) res=0 1 = 1. if (u<10) return n+10) 0(1) for (i=9; i>1; i--) } 0(9) -> constant (0(1)) while (n%i = =0)) 0(n) inside while loop changing field is n. so loop execution depends on Inside that loop there if (n>10) return -1) 0(1) ore some operations which costs are also constat. (o(1)) return res So while loop ost is O(n).

- In somefunction.

Best-case

If n/10, then it returns n+10. So time complexity is constant.

O(1).

Worst - case

If n is not smaller than 10. It will execute for loop.

And time complexity is O(3). O(n) = O(3) is constant)

we can ignore constants.

Time complexity is O(n).

```
PART 2
```

```
find Distance (Point[], Point P(x1,41), arr_len)
              new Dist = 0
                                                                     0(1)
               x pist = 0
               yDist = 0
                Point (x2, y2) = Point [0]
                X Dist = X2 - X1
                 old Dist = Square Root ( XDist * XDist + y Dist * yDist)
                 yoist = 42 - 11
                 for (i=0; iz or -len; i++)
                        Point(x2, y2) = point[i]
                         (new Dist = Square Root (x Dist & X Dist + y Dist & y Dist)
                         X Dist = Y2-X1
                         aldoist = find closest ( new dist , old dist) } o(1)
 I sent constants
so it will execute F
 constant
  find Closest (new Dist, old Dist)
                                   1000
               return old Dist
(I assume that, I have point structure) (which includes x and y points)
 The total ost is O(n).
- find Distance function time complexity: O(1) + O(n) = O(n)
- Best and worst cases one the same with total cost. (O(n))
```

```
2) findLocalMin (A[], arr_len)
        for (i=0; iz arr_len; i++)
             if(A[i] < = A[i+i] and A(i] <= A[i-1]) 0(1)
                  [i] A = nimosol
And All Mins (A[], arr_len)
      local Mins []
       k = 0
       for (i=0; icorr-len; i++)
             if (A[i] <= A[i+1] and A[i] <= A[i-1])
                   localmins[v] = A(i)
Time complexities are the same in two functions.
  There is a for loop and inside that loop there are operations
in constant time. So the loop contains site n, the cost
```

is in any case (worst-Best) O(n).

arr_len = n

- Somefunction finds array contains two numbers whose sum is a given number and returns how may numbers that function have.

- If the rested loops contain sizes n and n, the cost is $O(n^2)$. Inside that loop, there is if statement and that statement cost is constant $O(n^2)$, so total cost is $O(n^2)$.

- Best and worst cases, function runs same so time complexities are the same. $O(n^2)$

(i) is Sum Chain Of Length (arr (3, arr_len))

for (i=1; iz arr_len; i++) } O(n)if (some function (arr, i, arr (i)) ==0)} $O(n^2)$ return -1

3

((some function () is from third question)

Best-case

If the function canot find first number's sum in that array

it will return -1. So it will execute (with calling somefunction)

O(n2) times and it will break the for loop. So time complexity

0(2)

Worst case

If the function find all numbers sum in that array, Then for loop executes O(n) and with calling samefunction, it will also executes $O(n^2)$ times.

Total complexity is = $O(n \cdot n^2) = O(n^3)$. (Cubic)