GIT Department of Computer Engineering

CSE 222-505

Spring 2020 - Homework 2

Emirhan UZUN
171044019

PART 1

1.
```
somefunction (rows,cols){
    for(i=1; i<=rows;i++)        -> 1
    {
        for (j=1; j<=cols; ++j)  -> 2
            print (*)            -> 3

        print(newline)           -> 4
    }
}
```

Step No	Steps	Frequency	Total
1	2	rows+1	2rows + 2
2	2	(cols+1). rows	2rows.cols + 2rows
3	1	rows. cols	rows cols
4	1	rows	rows

+ 3rows.cols + 5rows + 2

$T_{worst}(rows, cols) = O(rows \times cols)$

$T_{best}(row, cols) = \Omega(rows \times cols) = \Theta(rows \times cols)$

Best and worst times are same. Because both of them, It depends on rows and columns. Also I wrote Θ for best time, since if best and worst are same, it also should be the average

2. somefunction(a,b)

```
{
    if(b==0)            → 1
        return 1        → 2
    answer =a           → 3
    increment=a         → 4
    for(i=1 ; i<b ; i++)    → 5
    {
        for(j=1 ; j<a ; j++)   → 6
        {
            answer += increment   → 7
        }
        increment = answer    → 8
    }
    return answer         → 9
}
```

Step No	Steps	Frequency	Total
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	2	b	2b
6	2	$(b-1).a$	$2ab-2a$
7	1	$(b-1).(a-1)$	$ab-a-b+1$
8	1	$(b-1)$	$b-1$
9	1	1	1

+

$3ab-3a+2b+5$

$T_{worst}(a,b) = O(a.b)$

$T_{best}(a,b) = \Omega(1)$

Best is $\Omega(1)$ because if returns in 2nd line, time will be constant

3. somefunction (arr[], arr_len)

```
{
    val=0                              → 1
    for(i=0 ; i< arr_len/2 ; i++)      → 2
        val = val + arr[i]             → 3
    for (i= arr_len/2 ; i< arr_len ;i++) → 4
        val = val - arr[i]             → 5
    if (val >= 0)                      → 6
        return 1
    else                               → 7
        return -1
}
```

Step No	Steps	Frequency	Total
1	1	1	1
2	2	arr_len/2 +1	arr_len +2
3	2	arr_len/2	arr_len
4	2	arr_len/2 +1	arr_len +2
5	2	arr_len/2	arr_len
6	1	1	1
7	1	1	1

+ ______

4arr_len + 7

T_{worst} (arr_len) = O (arr_len)

T_{best} (arr_len) = Ω (arr_len) = Θ (arr_len)

Since the steps depend on arr_len, best and worst times are same and equal to arr_len.

4. somefunction (n)

```
{
    c=0                  → 1
    for(i=1 to n*n)      → 2
      for(j=1 to n)      → 3
        for(k=1 to 2*j)  → 4
          c=c+1          → 5
    return c             → 6
}
```

Step No	Steps	Frequency	Total
1	1	1	1
2	2	n^2+1	$2n^2+2$
3	2	$(n^2).(n+1)$	$2n^3+2n^2$
4	2	$(n^2).n.((n+1).n+1)$	$2n^5+2n^4+2n^3$
5	2	$n.(n+1).n^3$	$2n^5+2n^4$
6	1	1	1

$+$

$$4n^5+4n^4+4n^3+4n^2+4$$

$$T_{worst}(n) = O(n^5)$$

$$T_{best}(n) = \Omega(n^5) = \Theta(n^5) = O(n^5)$$

Best and worst times are same because it depends on n. The 4^{th} line is change by j. So it is little confusing but every time, it runs $n.(n+1)$ times.

5. otherfunction (xp, yp)

```
{
    temp = xp
    xp = yp
    yp = temp
}
```

$T_{best}(xp, yp) = T_{worst}(xp, yp) = \Theta(1)$

```
somefunction (arr[], arr_len)
{
  for (i=0 ; i< arr_len ; 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])
  }
}
```

In this function there are 2 loops and 1 function call. But the function which is called (otherfunction) has constant time notation.

So $T_{best}(arr\_len) = T_{worst}(arr\_len) = \Theta(arr\_len^2)$

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
}
```

In other function, there is nested loop. So the worst case $T_{worst}(a,b) = O(a.b)$
But in the best case, the function returns 1 just second line.
So $T_{worst}(a,b) = O(a.b)$
$T_{best}(a,b) = \Omega(1)$

```
somefunction (arr, arr-len)
{
    for i=0 to arr-len
        for j=i to arr-len
            if otherfunction (n%i, 2) == arr[i]
                print (arr[i], arr[j])
}
```

In some function, other function is called $(arr\text{-}len)^2$ times and other function's time notation is a.b. In some function, b is always 2, so it is constant. Generalize;

$$T_{worst} = T_{best}(arr\_len) = O(arr\_len^2 \times a \times \underset{\downarrow \atop 2}{b}) = O(arr\_len^2 \times a)$$

7) otherfunction (X, i)

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

j increases like 1,2,4,8,16

So $T(n) = \Theta(\log_2 i)$

```
somefunction (arr[], arr_len)
{
    for (i=0 ; i ≤ arr_len-1; i++)
        A[i] = otherfunction (arr, i) / i+1
    return A
}
```

In this function, loop is called arr_len + 1 and other function is also called arr_len. And the total time is:

$$T(arr\_len) = \Theta((arr\_len + 1) . \log_2 (arr\_len - 1)) = \Theta(arr\_len . \log_2 arr\_len)$$

Scanned with CamScanner

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
}
```

In this function, I though like that while turns prime factors. (For instance if n=20 while turns 3 times for 1,4,5). I tried so much times to worst case. Then, I decided the n divided by i every time if the mode is zero. So it depends on n. The frequency time is not greater than $\log_2 n$. Because the number is decreasing after every step.

For the best case, if the first condition is true, it returns. So this time is constant.

$T_{worst}(n) = O(\log_2 n)$

$T_{best}(n) = \Omega(1)$

Scanned with CamScanner

PART 2

1) some function (row, col)

```
{
    for (i=0; i<row; ++i) {
        for (j=0; j<col) {
            if ( arr[i][j] != NULL && (i != row || j != col) {
                tempMin = sqrt ((row-i)*(row-i) + (col-j)*(col-j))
                if (tempMin < min) {
                    min = tempMin
                    x = i
                    y = j
                }
            }
        }
    }
}
```

Firstly I initialized min to 9999. I didn't write in pseudo code. Then I check the array is empty or not. After that, I calculate the distance between points. During the loop, determine the minimum and initialize its coordinates to x and y. $T(row, col) = O(row.col) = \Theta(row.col)$ the best and worst is same.

Scanned with CamScanner

2.a) some function (arr[], n)

```
some function (arr[], n)
{
    for ( i=1; i<n-1; ++i) {
        if (arr[i] <= arr[i+1]  &&  arr[i] <= arr [i-1]
            return  arr[i]
    }
}
```

When we find local minimum which is less than or equal to prev and next element, this function returns this. It needs linear time because it depends on n. Tworst and Tbest are equal to O(n) (Θ(n))

2.b)

```
some function ( arr[], n, target)
{
    count = 0

        for (j=1; j<n-1; ++j) {
            if (arr[i] <= arr[i+1]  &&  arr[i] <= arr[i-1] {
                arr2 [count] = arr[i]
                count ++
            }
        }

}
```

This function is similar 2.a. function. Only difference is in 2.b, we put to the arrays. So all local minimums are in the arr2 arrays. It also needs linear time. Tworst and Tbest are equal to O(n) (Θ(n))

3.) some function (arr[], n, target)

```
{
    for (i= 0; i<n; ++i)
        for (j=0; j<n; ++j)
            if (arr[i] + arr[j] == target) return 1
    return 0
}
```

In this function, I check all the 2 numbers which is given target number. If it is found, return 1. Tworst and Tbest are equal to n^2. $T_{best} = T_{worst} = O(n^2) = \Theta(n^2)$

4.) some function (arr[], n)

```
{
    k=1
    for (i=0; i<n; ++i)
        for (j=0; j<n; ++j)
            if ( arr[i] + arr[j] == arr[k]   k++
    if (k==n) return 1
    return 0
}
```

4^{th} function is similar to 3^{rd}. Also, we can say 4^{th} extends from 3^{rd}. The difference is, in this function we check if all this members in array, is sum of before this elements. For instance, if k is 4, we control that to the 4^{th} elements 0, 1, 2, 3^{rd} elements' sum is 4^{th} element (But just two numbers sum). Finally $T_{best} = T_{worst}(n) = O(n^2) = \Theta(n^2)$

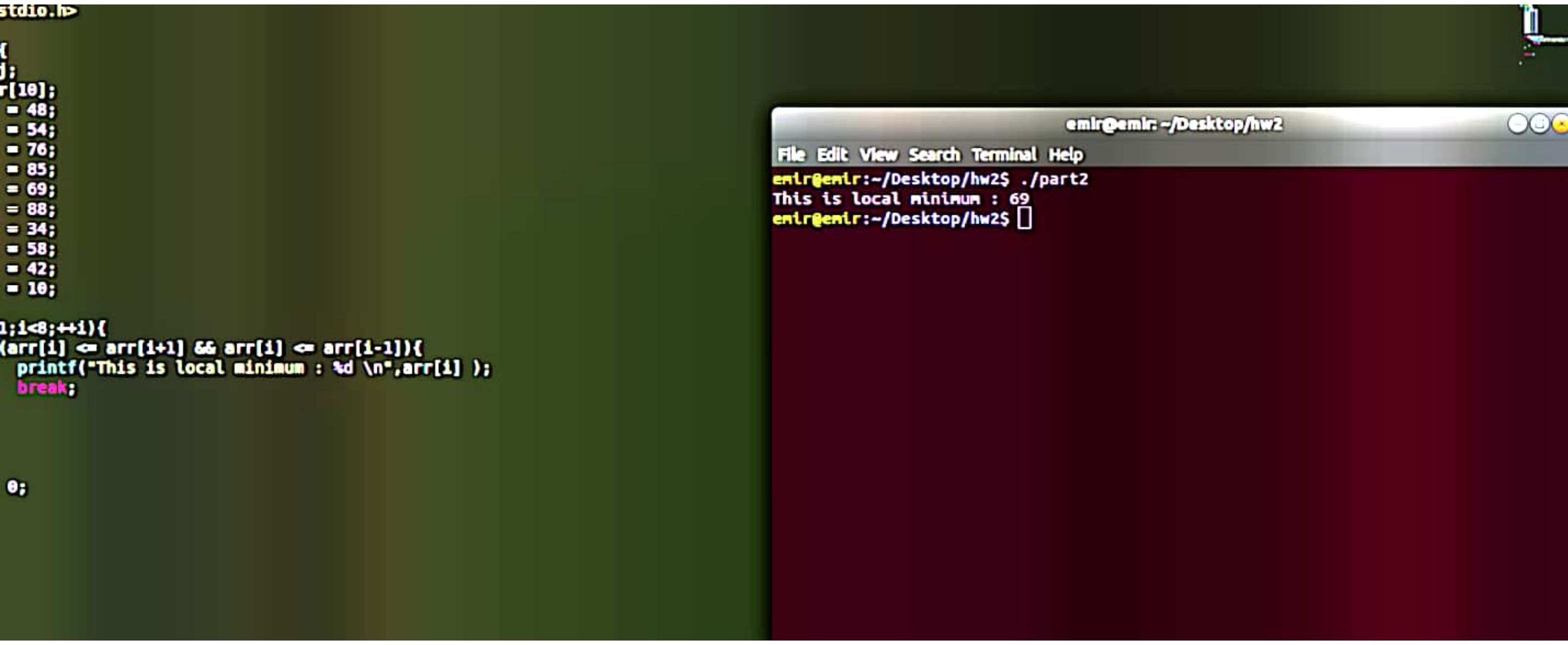
part3.c part4.c part1.c part2.c part2b.c

```c
#include <stdio.h>
#include <math.h>

int main(){
    int i,j;
    int arr[5][6];
    int x = 0 ;
    int y = 0;
    float tempMin ;
    float min = 9999.0;
    for(i=0;i<5;++i){
        for(j=0;j<6;++j){
            arr[i][j] = 0 ;
        }
    }
    arr[3][4] = 88;
    arr[2][5] = 77;
    arr[1][3] = 99;
    arr[1][5] = 106;

    for(i=0;i<5;++i){
        for(j=0;j<6;++j){
            if(arr[i][j] != 0 && (i!=2 || j!=5)){
                tempMin = sqrt((2-i)*(2-i) + (5-j)*(5-j));
                if(tempMin < min) {
                    min= tempMin;
                    x = i;
                    y = j;
                }
            }
        }
    }
    printf("The closest point's coordinates are : %d and %d \n",x,y);
```

emir@emir: ~/Desktop/hw2

File Edit View Search Terminal Help

```
emir@emir:~/Desktop/hw2$ ./part1
The closest point's coordinates are : 1 and 5
emir@emir:~/Desktop/hw2$
```


stdio.h>
{
j;
r[10];
= 48;
= 54;
= 76;
= 85;
= 69;
= 88;
= 34;
= 58;
= 42;
= 10;
1;i<8;++i){
(arr[i] <= arr[i+1] && arr[i] <= arr[i-1]){
printf("This is local minimum : %d \n",arr[i]);
break;
0;
emir@emir: ~/Desktop/hw2
File Edit View Search Terminal Help
emir@emir:~/Desktop/hw2$ ./part2
This is local minimum : 69
emir@emir:~/Desktop/hw2$

```c
#include <stdio.h>

int main(){
	int i,j;
	int arr[10];
	int arr2[10];
	int count = 0;
	arr[0] = 48;
	arr[1] = 54;
	arr[2] = 76;
	arr[3] = 85;
	arr[4] = 69;
	arr[5] = 88;
	arr[6] = 34;
	arr[7] = 58;
	arr[8] = 42;
	arr[9] = 10;

	for(i=1;i<9;++i){
		if(arr[i] <= arr[i+1] && arr[i] <= arr[i-1]){
			arr2[count] = arr[i];
			count++;
		}
	}

	printf("The local minimums are : \n");
	for(i=0;i<2;++i){
		printf("%d\n",arr2[i] );
	}

	return 0;



}
```

emir@emir: ~/Desktop/hw2

File Edit View Search Terminal Help

```
emir@emir:~/Desktop/hw2$ gcc -c part2b.c
emir@emir:~/Desktop/hw2$ gcc -o part2b part2b.o
emir@emir:~/Desktop/hw2$ ./part2b
The local minimums are :
69
34
emir@emir:~/Desktop/hw2$
```

part3.c | part4.c | part1.c | part2.c | part2b.c

```c
#include <stdio.h>


int find_sum(int arr[],int given){
	int i,j;
	for(i=0;i<10;++i){
		for(j=0;j<10;++j){
			if(arr[i] + arr[j] == given) return 1;
		}
	}
	return 0;
}


int main(){
	int arr[10];
	arr[0] = 48;
	arr[1] = 54;
	arr[2] = 76;
	arr[3] = 85;
	arr[4] = 69;
	arr[5] = 88;
	arr[6] = 34;
	arr[7] = 58;
	arr[8] = 42;
	arr[9] = 10;
	int b = 112;
	printf("This is the result : %d\n",find_sum(arr,118));
}
```

emir@emir: ~/Desktop/hw2

File Edit View Search Terminal Help

```
emir@emir:~/Desktop/hw2$ gcc -c part3.c
emir@emir:~/Desktop/hw2$ gcc -o part3 part3.o
emir@emir:~/Desktop/hw2$ ./part3
This is the result : 1
emir@emir:~/Desktop/hw2$
```

part3.c part4.c part1.c part2.c part2b.c

```c
#include <stdio.h>


int find_sum(int arr[]){
    int i,j,k;
    k = 1;
    for(i=0;i<7;++i){
        for(j=0;j<k;++j){
            if(arr[i] + arr[j] == arr[k]) k++;
        }
    }
    if(k == 7)  return 1;
    return 0;
}


int main(){
    int arr[7];
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    arr[3] = 5;
    arr[4] = 10;
    arr[5] = 13;
    arr[6] = 15;
    int b = 112;
    printf("This is the result : %d\n",find_sum(arr));
}
```

emir@emir: ~/Desktop/hw2

File Edit View Search Terminal Help

```
emir@emir:~/Desktop/hw2$ gcc -c part4.c
emir@emir:~/Desktop/hw2$ gcc -o part4 part4.o
emir@emir:~/Desktop/hw2$ ./part4
This is the result : 1
emir@emir:~/Desktop/hw2$
```