HW2

Baran Hasan BOZDUMAN
171044036

GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 2
Upload Due date: March 13 2020 – 12:00

(You will both bring your handwritten solutions to Nur Banu Albayrak (118) and upload the scanned versions to Moodle. You can bring the handwritten solutions due to the end of March 13.)

PART 1 : Analyze the following algorithms. Write worst-case, average-case, base-case analysis if significant. Express your results using most proper asymptotic notation. Explain your solutions. For the 1st to the 4th algorithms use table method and show table.

1.

somefunction(rows, cols)	step/exec	freq.	total
{			
for(i = 1; i <= rows; i++)	1	row+1	row+1
{			
for(j = 1; j <= cols; j++)	1	row(cols+1)	row(cols+1)
print(*)	1	row.cols	row.cols
print(newline)	1	row	row
}			
		T(row,cols)=	2row.cols +3row +1
}			

for the general case $T(row,cols) = \Theta(row,cols)$ Best case is $\Theta(row.cols)$

worst case is $\Theta(rows.cols)$

There is no condition to interrupt the loops so for every case the loops gonna repeat.

2.

somefunction(a, b)	Step/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++)	1	b	b
{			
for(j = 1; j < a; j++)	1	a	b(a-1)
{			
answer += increment	1	1	1
}			
increment = answer	1	1	1
}			
return answer	1	1	1 ★
}			
		T(a,b)	ba -2b +b

In Best case the function works until 3. first return. and it takes constant time

In the Worst case the function works until the last return and that takes a.b.

Best case is $\Theta(1)$

Worst case is $\Theta(a.b)$

$T(a,b) =$ we can say that $O(a.b)$ and $\Omega(1)$

```
somefunction(arr[], arr_len)
{
    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]
    if (val >= 0)
        return 1
    else
        return -1
}
```

step exec	frequency	total
1	1	1
1	arrlen/2 + 1	arrlen/2 + 1
1	1	1
1	arrlen/2 + 1	arrlen/2 + 1
1	1	1
1	1	1
1	1	1
1	1	1
		arrlen + 7

Best case $\theta(arr\_len)$

Worst case $\theta(arr\_len)$

We can express $T(arr\_len) = \theta(arr\_len)$

4.

we have two without nestedloops so the for loops and repeat until arr_len/2 but for the analysis we can say this arr_len

```
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 exec	frequency	total
1	1	1
1	n^2+1	n^2+1
1	$n^2(n+1)$	n^3+n^2
1	$n^2 \cdot \frac{(n^2+1)(n+1)}{2} \cdot 1$	n^4+n^3
1	1	1
1	1	1
		$n^4+2n^3+2n^2+4$

we can think the for loops as a loop the first loop changes according to first for its $\frac{n(n+1)}{2}$ and $2n$

$\frac{n(n+1)}{2}$ works for the two loop

Best case $\theta(n^4)$

Worst case $\theta(n^4)$

$T(n) = \theta(n^4)$

because of in worst and best case

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])
    }
}
```

constant time

arr_len

$\frac{arr\_len.(arr\_len+1)}{2}$ $\cdot (arr\_len)^2 + arr\_len$

$T(arr\_len) = \theta(arr\_len^2)$

$\theta(1)$

we can think that the for loops of a loop so from formula $\frac{n(n+1)}{2}$

Best case is $\theta(arr\_len^2)$

Worst case is $\theta(arr\_len^2)$

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
}

somefunction(arr, arr_len)
{
    for i = 0 to arr_len):
        for j = i to arr_len): n%i^2
            if otherfunction(arr[i], 2) == arr[j]:
                print(arr[i], arr[j])
            elif otherfunction(arr[j], 2) == arr[i]:
                print(arr[j], arr[i])
}
```

constant time $\Theta(1)$

a.b $\Theta(a.b)$

Best case $\Theta(1)$
Worst case $\Theta(a.b)$

$T(a,b) = O(a.b)$

→ when we take the mod of arr_len it gives us a sum of sequence but not ordered which is goes until n.

$\frac{n.(n+1)}{2}$ → (the rule almost)

and that gives n^2
we can say $T(n) = O(n^2)$ (?)

7.

```
otherfunction(X, i)
{
    s = 0
    for(j = 0; 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
}
```

$\Theta(1)$

$\log n$ → because it increases multiplied by 2

$\Theta(1)$

$\log 1 + \log 2 + \cdots \log arr\_len = \log arr\_len!$
$\log 1.2.3 \ldots arr\_len$

Best case $\Theta(\log n!)$
Worst case $\Theta(\log n!)$

we can say that $T(arr\_len) = \Theta(\log arr\_len!)$

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
}
```

constant time Θ(1)

constant time Θ(1)

Θ(1) for the best case
the function can finish first return if the n value is smaller than 10.

constant time Θ(1)

PART 2 : Design an algorithm for each of the problems. Write your algorithms in pseudo code. Obtain the complexities of the algorithms. Write worst-case, average-case, base-case analysis if significant. Express your results using most proper asymptotic notation. Explain your solutions.

1. Assume you have an array of points in 2d space. Find the closest point in the array to a given point.
2. The i^{th} 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.
3. Find if a given array of integers contains two numbers whose sum is a given number b.
4. A sequence of positive integers in increasing order, a1, a2,…,an is called a "Sum Chain of Length n" if for all k ($1 < k \leq n$), there exist i, j ($1 \leq i \leq j \leq k$) such that ak=ai+aj

   Example: {1, 2, 3, 5, 10, 13, 15} : (2=1+1, 3=2+1, 5=3+2, 10=5+5, 13=10+3, 15=10+5)

   Find if a given sequence of n numbers is a "Sum Chain of Length n". Use the algorithm you design for the third question in this part.

1) 171044036

```
Find closestPoint ( Point point, Point[] pointArray, array-len ) {
    x = 0, y = 0 } 1
    result = 999...
    for i=0 to array-len                                   } n
        if arraylen != i+1
            x = power ((point.x - pointArray[i+1].x), 2);
            y = power ((point.y - pointArray[i+1].y), 2);
            if (result > sqrt(x+y)
                result = sqrt(x+y)
}
```

1 //constant time $\Theta(1)$

$\Theta(n)$ for the worst case

$\Theta(n)$ for the best case

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

2)

a)

```
Find local_min ( int[] array, array length
    if array-length > 2
        for (i=1; i<arrlength; i++)
            if ( arrayLi-1] <= arrayLi] && arrayLi] <= arrayLi+1])
                return arrayLi]

    return -9999
```

$\Theta(n)$ for the worst case

$\Theta(1)$ for the best case

after it finds the number it returns for best case it finds once lookup and returns for worst case it looks until the end of array.

$T(n) = O(n)$
$\Omega(1)$

b)

```
find_all local min (int[] array, array length
    ArrayList<Integer> array
        if array_length > 2
            for i=1 to array_length
                if (arrayLi-1] <= arrayLi] && arrayLi] <= arrayLi+1])
                    array.add (arrayLi]);
```

$\Theta(n)$ for the worst case

$\Theta(1)$ for the best case

for the two situation it goes until the end of array

$T(n) = O(n)$

3) Find the integers (int given , int[] array, int array_len)

```
for (i=0; i<array_len; i++)                    n
    for (j=0; i<array_len; j++)                n^2
        if (j!=i)
            if (array[j]+array[i] == given)
                arrayij.add (array[j])
                arrayij.add (array[i])          } constant
                return true;
return false
```

$\Theta(1)$ for the best case

$\Theta(n^2)$ for the worst case

$T(n) = O(n^2)$ or $\Omega(1)$

for the best case it will breaks the loop after it find out
for the worst case it will look up for every situation

4) Find_ifchainSum (issumchainarray[] array_length)

```
result;
    for i=0; to , array_length                (n)
        result = findtheintegers( issumchainarray[i+1] , issumchainarray[], (i+1))
        if result == false
            return false

return true;
```

for the worst case $\Theta(n^3)$ $T(n) = O(n^3)$ or $\Omega(n)$

for the best case $\Theta(n)$

it sends the array expand in each step and it have to check until the end of the array so in best case it takes n time. for the worst case it repeat n for the sum of chain array and n^2 times for the find integer function