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.

	step/exec.	freq	total
somefunction(rows, cols) {			
for(i = 1; i <= rows; i++)	2	rows +1	2rows +2
{			
for(j = 1; j <= cols; j++)	2	rows.(cols+1)	2rowscols +2rows
print(*)	1	rows cols	rows cols
print(newline)	1	rows	rows
}			3.rows cols + 5rows +2 → O(rows.cols)
}			

$T_{best} = T_{worst} = \Theta(rows.cols)$. Because of there isn't statement which breaks loops.

$T(rows, cols) = \Theta(rows.cols)$

2.

	step/exec.	freq.	total
somefunction(a, b) {			
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	2b
{			
for(j = 1; j < a; j++)	2	(b-1).a	2ab -2a
{			
answer += increment	1	(b-1)(a-1)	ab -b -a +1
}			
increment = answer	1	b-1	b-1
}			
return answer	1	1	1
}			3ab -3a +2b +5 → O(a.b)

$T_{best} = \Theta(1)$, $T_{worst} = \Theta(a.b)$

3. If b equals 0, function ends. So best case is 1. There is no break to terminate function. So worst case is a.b.

$T(a,b) = O(a.b) = \Omega(1)$

	step/exec.	freq.	total
somefunction(arr[], arr_len) {			
val = 0	1	1	1
for (i = 0; i < arr_len / 2; i++)	3	arr_len/2 +1	3arr_len/2 +3
val = val + arr[i]	2	arr_len/2	arr_len
for (i = ~~n~~ arr_len / 2; i < arr_len; i++)	2	arr_len/2 +1	arr_len +2
val = val - arr[i]	2	arr_len/2	arr_len
if (val >= 0)	1	1	1
return 1	1	1	1
else			
return -1	1	1	1
}			9 arr_len/2 +9 → O(arr_len)

$T_{best} = T_{worst} = \Theta(arr\_len)$. because of there is no break to teminate function and loops.

$T(arr\_len) = \Theta(arr\_len)$

4.

	step/exec.	freq.	total
somefunction(n) {			
c = 0	1	1	1
for (i = 1 to n*n)	2	$n*n$	$2n^2$
for (j = 1 to n)	2	$n*n*n$	$2n^3$
for (k = 1 to 2*j)	2	$n*n*n*(n+1)$	$2n^4+2n^3$
c = c+1	2	$n*n*n*n$	$2n^4$
return c	1	1	1
}			$4n^4+4n^3+2$ → $O(n^4)$

$\sum_{j=1}^{n} 2j = 2+4+\cdots+2n = n.(n+1)$ $\quad\sum_{j=1}^{n} 2j-1 = 1+3+\cdots+2n-1 = n^2$

5. $T_{best} = T_{worst} = \Theta(n^4)$ because of there is no break to terminate function and loops.

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

```
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])
    }
}
```

otherfunction: $O(1) = \Omega(1) = \Theta(1)$

somefunction: $T(arr\_len) = \Omega(arr\_len) = \Theta(arr\_len) = O(arr\_len)$

inner for: $T(arr\_len) = \Theta(arr\_len - i)$

otherfunction: $\Theta(1)$

$\Theta(N^2)$

$T_{best} = T_{worst} = \Theta(N^2)$ because of there is no break to terminate function and loops.

$T(N) = \Theta(N^2)$

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):
         if otherfunction(arr[i], 2) == arr[j]:
            print(arr[i], arr[j])
         elif otherfunction(arr[j], 2) == arr[i]:
            print(arr[j], arr[i])
}
```

$O(1) = \Omega(1) = \Theta(1)$

$O(a \cdot b) = \Omega(a \cdot b) = \Theta(a \cdot b)$

$\rightarrow \Theta(1)$

$T = \Theta(a \cdot b)$

$\sum_{j=1}^{arr\_len} (arr\_len - i) = \frac{arr\_len(arr\_len + 1)}{2}$

$= n \cdot \frac{(n+1)}{2}$

$T_{worst} = T_{best} = T(n) = \sum_{i=1}^{\frac{n \cdot (n+1)}{2}} 2 \, (n \% i)$

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
}
```

$\Theta(\log(n))$

$\sum_{i=1}^{n} \log(i) = \log 1 + \log 2 + \ldots + \log n = \log(n!)$

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

$T_{best} = T_{worst} = \Theta(\log(n!))$

There is no break to terminate function and loops. So best case and worst case are equal.

8.

```
somefunction(n)
{
      res = 0
      j = 1                                   } Θ(1)

      if(n < 10)
            return n + 10

      for(i = 9; i >= 1; i--)                 → Θ(1)
            while (n % i == 0)
                  n = n / i
                  res = res + j * i           } O(logn)
                  j *= 10

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

$T_{best} = \Theta(1)$, $T_{worst} = \Theta(\log n)$, $T(n) = O(\log n)$

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.

Part 2.1 :

```
calcDistance (P1, P2)
    return   sqrt(pow(P1.x - P2.x, 2) + pow(P1.y - P2.y, 2))
```
} Θ(1)

```
closestPoint (Points[], size, point)
    if (size < 1)
        return -1
min = calcDistance (Points[0], point)
distance = 0
```
} Θ(1)

```
for (i=1 ; i < size ; ++i)
    .distance = calcDistance (Points[i], point)
     if (distance < min)
            min = dist
return min
```
} Θ(n)

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

Part 2.2.a :

```
firstLocalMin (A[], size)
    if (size > 2)
        for (i=1 ; i < size-1 ; ++i)
            if (A[i] <= A[i+1] && A[i] <= A[i-1])
                return A[i]
    return -1
```
} $T_{best} = T_{worst} = T(n)$, $T(n) = \Theta(n)$

Part 2.2.b:

```
allLocalMin(A[], size)
    if(size > 2)
        for(i=1; i<size-1; )
            if(A[i] <= A[i+1] && A[i] <= A[i-1])
                print(A[i])
                i += 2
            else
                ++i
    return -1
```

for loop → $\Theta(n)$

return -1 → $\Theta(1)$

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

Part 2.3:

```
isContain(A[], size, num)
    for(i=0; i<size; ++i)
        for(j=i; j<size; ++j)
            if(A[i] + A[j] == num)
                return true
    return false
```

if block → $\Theta(1)$

$\sum_{j=0}^{size} size - j = size + size-1 + \dots + 0$

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

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

$T_{best} = \Theta(1) \quad T_{worst} = O(n^2)$

Part 2.4

```
sumChainOfLength(A[], size)
    for(i=1; i<size; ++i)
        if(!isContain(A, i, A[i]))
            return false
    return true
```

if(!isContain(A, i, A[i])) → $O(n^2)$

return false → $\Theta(1)$

loop body → $O(n^2)$

$\sum_{i=1}^{size} i^2 = 1+3+ \dots + size^2$

$= \frac{size.(size+1).(2size+1)}{2}$

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

$T_{best} = \Theta(1) \quad T_{worst} = O(n^3)$