COMP 7270 Assignment 2 No late submissions accepted!

Upload your submission well before this deadline. Even if you are a few minutes late, as a result of which Canvas marks your submission late, <u>your assignment may not be accepted</u>.

Instructions:

1. This is an individual assignment. You should do your own work after discussing with at most two partners. **Write out whom you discussed with. Any evidence of copying will result in a zero grade and additional penalties/actions.**
2. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. **Think carefully; formulate your answers, and then write them out concisely** using English, logic, mathematics and pseudocode (<u>no programming language syntax</u>).
4. **Type your answers in this Word document and submit it. If that is not possible, use a word processor to type your answers as much as possible (you may hand-write/draw equations and figures), turn it into a PDF document and upload**.

1. Use the Recursion Tree Method to show that $T(n)=2(2^n)–1$ for a recursive algorithm characterized by the recurrences $T(n)=2T(n-1)+1$; $T(0)=1$. You must fill in the table below <u>for the first three levels of the Recursion Tree and for the base case level and the level above</u>. Then write out the expression T(n) that you get by adding all values in the last column and simplifying using results from Appendix A to show the above. This simplification <u>must</u> be shown to get any credit.
<u>Finally, state the most accurate complexity order of this algorithm</u>:

Level #	# of recursive executions at this level <u>as a function of</u> level #	Input size to each execution	Additional work done by each execution	Total work done at this level <u>as a function of</u> level #
0	1(0)	T(n)	0	1(0)
1	2(1)	T(n-1)	2T(n-1)	$2^1(1)$
2	4(2)	T(n-2)	4T(n-2)	$2^2(2)$
3	8(3)	T(n-3)	8T(n-3)	$2^3(3)$
base	$2(2^n)$-1(base)	T(n/k)	aT(n-k)	$2(2^n)$-1(base)

T(n)=

2T(n-1) + 1

$T(2) = 2T(1) + 1 = 3 * 1 = 4 * 1 – 1 = 2^{2 * 1 – 1}$

$T(3) = 2T(2) + 1 = 7 * 1 = 8 * 1 – 1 = 2^{3 * 1 - 1}$

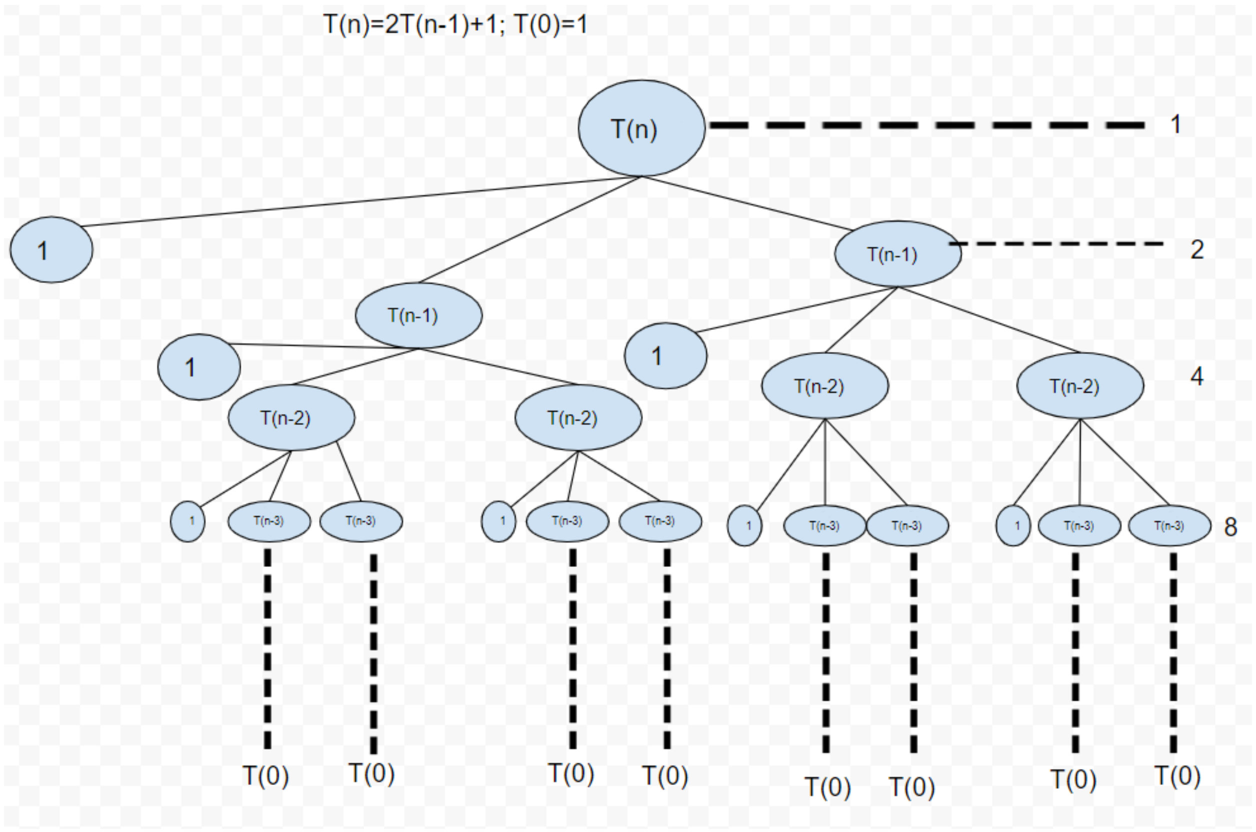
$T(4) = 2T(3) + 1 = 15 * 1 = 16 * 1 – 1 = 2^{4 * 1 - 1}$

$T(5) = 2T(4) + 1 = 31 * 1 = 32 * 1 – 1 = 2^{5 * 1 - 1}$

$T(6) = 2T(5) + 1 = 63 * 1 = 64 * 1 – 1 = 2^{6 * 1 – 1}$

$T(n) = 2^{n1-1}$

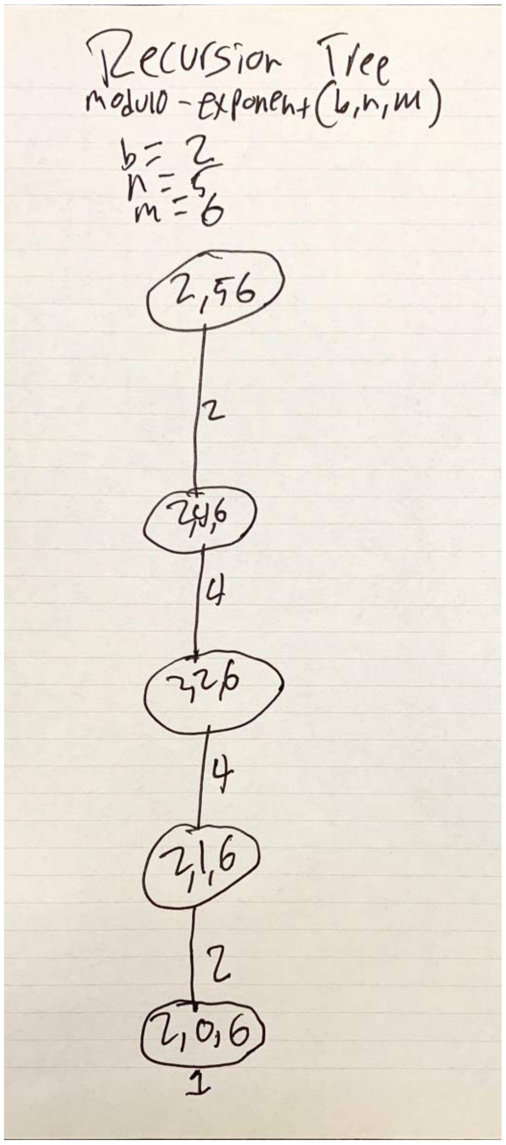
Complexity Order = $O(2^n)$



2. Understand this recursive algorithm for computing (b^n mod m). Then draw the **<u>Recursion Tree</u>** of it computing (2^5 mod 6).

<u>Note: At each node of the tree you should show the inputs and outputs of that execution clearly to receive credit.</u>

```
Modulo-exponent(b,n,m: integers such that m≥2, n≥0)
1.  if n==0 then return 1
2.  else
3.      temp = Modulo-exponent(b,floor(n/2),m)
4.      temp = temp*temp
5.      if n is even then
            return temp mod m
6.      else
7.          return (temp mod m)*(b mod m) mod m
```



3. Use the Detailed Method to determine the precise T(n) of the following iterative maximum subsequence sum (MSS) algorithm. You must show your work below to get any credit. The algorithm is as below.

```
        MSS Algorithm-1 (A:array[p..q] of integer)
        sum, max: integer
1       sum = max = 0
2       for i = p to q
3           sum = 0
4           for j = i to q
5               sum = sum + A[j]
6               if sum > max then
7                   max = sum
8       return max
```

Line #	Step	Single execution cost	# times executed
1	sum = max = 0	4	1
2	for i = p to q	1	n+1
3	sum = 0	1	n
4	for j = i to q	$(n+1)^2$	n(n+1)
5	sum = sum + A[j]	4	n^2
6	if sum > max then	8	n^2
7	max = sum	1	n or n/2
8	return max	1	1

$T(n) = 1(4) + (n+1)1 + n(1) + n(n+1)(n+1)^2 + n^2(4) + n^2(8) + n(1) + 1(1)$

$= 4 + (n+1) + n + n(n+1)(n+1)^2 + 12n^2 + n + 1$

$= 6 + 3n + n(n+1)(n+1)^2 + 12n^2$

$= 6 + 3n + 12n^2 + (n^2+n)(n+1)^2$

$= 7 + 4n + 14n^2$

$= O(n^2)$

4. Develop, state and solve the recurrence relations of the Recursive Divide & Conquer iterative algorithm as follows by answering the following questions.

```
MSS Algorithm-2 (A:array[p..q] of integer)
1         if p=q then
2                   if A[p] > 0 then
3                             return A[p]
4                   else return 0
5         left-partial-sum = right-partial-sum = max-right = max-left = left-max-sum = right-max-sum = 0
6         center = floor((p+q)/2)
7         max-left = Algorithm-2(A[p..center])
8         max-right = Algorithm-2(A[center+1..q])
9         for i from center downto p do
10                  left-partial-sum = left-partial-sum + A[i]
11                  if left-partial-sum > left-max-sum then
12                            left-max-sum = left-partial-sum
13        for i from center+1 to q do
14                  right- partial-sum = right-partial-sum + A[i]
15                  if right- partial-sum > right-max-sum then
16                            right-max-sum = right- partial-sum
17        if max-left≤max-right then
18                  if max-right≤left-max-sum+right-max-sum then
19                            return left-max-sum+right-max-sum
20                  else return max-right
          else
21                  if max-left<eft-max-sum+right-max-sum then
22                            return left-max-sum+right-max-sum
23                  else return max-left
```

You must state costs in terms of n with numerical coefficients, and not using a complexity order notation, to get credit. You may assume that the for loops on lines 9-12 and 13-16 are executed n/2 times.

Which statements are executed when the input is a base case (provide line #s)? 1-4

What is the total cost of these? 8+8+1+1=18

Which statements are executed when the input is not a base case (provide line #s)? 5-20 or 5-17,21-23

What is the total cost of these? 18+6+4+5+1+8+10+7+2+8+10+7+5+9+6+3=109 OR 18+6+4+5+1+8+10+7+2+8+10+7+5+9+7+3=110

Provide the complete and precise two recurrence relations characterizing the complexity of MSS Algorithm-2:

$T(n)$ = running time of MSS on input of size n $T(n) = 1$ when $n=1$

$T(n)$ = $T(n) = 2T(n/2) + O(n)$ when $n>1$

Now simplify the recurrence relations by:

1. If your recurrence relation for the non-base case input has multiple terms in it besides the term representing the recursive calls, keep only the largest n-term from them and drop the others; if your recurrence relation for the non-base case input has only one other term besides the term representing the recursive calls, keep it. $O(n\log n)$

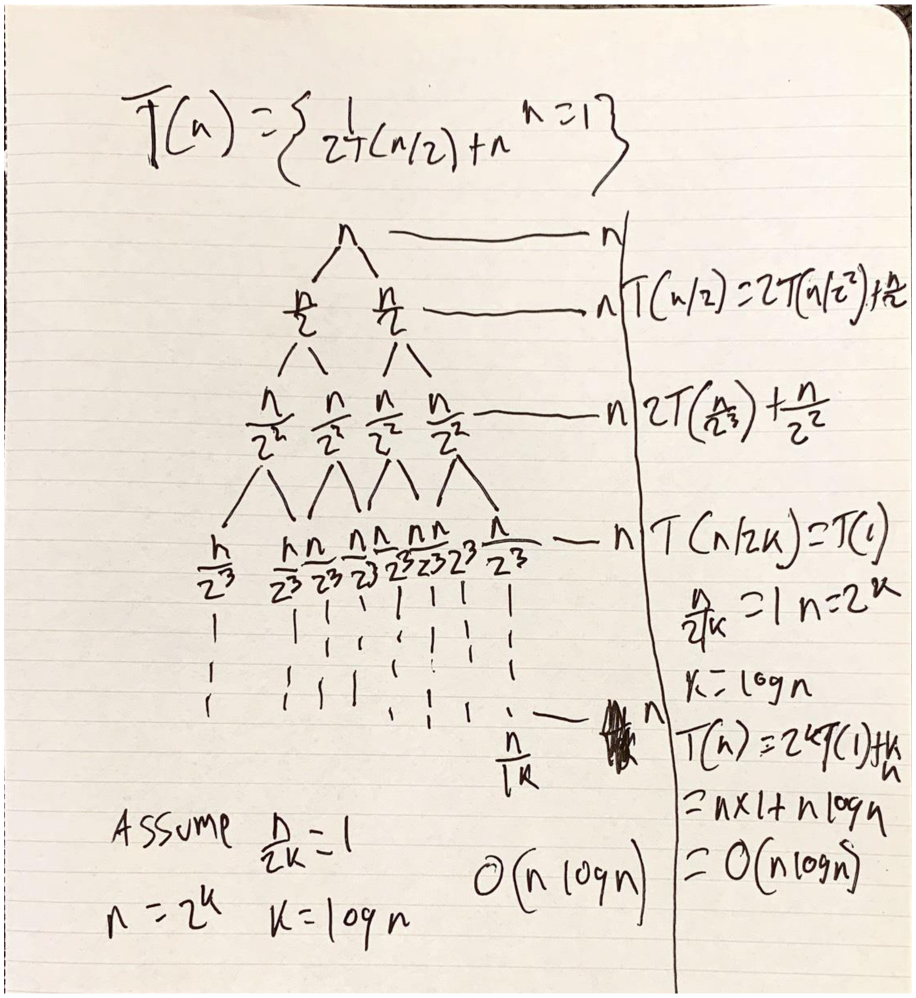
2. Take the largest numerical coefficient of all terms (excluding the term representing the recursive calls) in your two recurrence relations, round it up to the next integer if it is not an integer, and replace the numerical coefficients of all other terms (excluding the term representing the recursive calls) with this coefficient.

Provide the simplified recurrence relations below.

T(n) = O(nlogn) when n=1

T(n) = n when n>1

Solve these recurrence relations using the Recursion Tree method, determine and state the T(n) of the algorithm. You must show your work below to get any credit.



$T(n) = \left\{ \begin{matrix} 1 & n=1 \\ 2T(n/2)+n \end{matrix} \right\}$

n — n
$\frac{n}{2}$ $\frac{n}{2}$ — n
$\frac{n}{2^2}$ $\frac{n}{2^2}$ $\frac{n}{2^2}$ $\frac{n}{2^2}$ — n
$\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ $\frac{n}{2^3}$ — n
$\frac{n}{1k}$ — n

$T(n/2) = 2T(n/2^2) + \frac{n}{2}$

$2T(\frac{n}{2^3}) + \frac{n}{2^2}$

$T(n/2k) = T(1)$

$\frac{n}{2k} = 1$ $n = 2^k$

$k = \log n$

$T(n) = 2^k T(1) + \frac{k}{n}$

$= n \times 1 + n \log n$

$= O(n \log n)$

Assume $\frac{n}{2k} = 1$

$n = 2^k$ $k = \log n$

$O(n \log n)$

5. Let Result[1..2] be an array of two integers. Modify steps of the Iterative MSS algorithm as in Question 3 to return the starting and ending indexes of the optimal (maximum) subsequence it found in its last line instead of the maximum sum value. Provide your modified algorithm below. Make only the minimum number of modifications necessary. You will need to add additional steps.

```
        MSS Algorithm-1 (A:array[p..q] of integer, Result:array[1..2])
        sum, max: integer
1       sum = max = 0
2       for i = p downto q
3               sum = 0
4               for j = i to q
5                       sum = sum + A[j]
6                       if sum > max then
7                               result[1] = i
8                               result[2] = j
9       return result
```

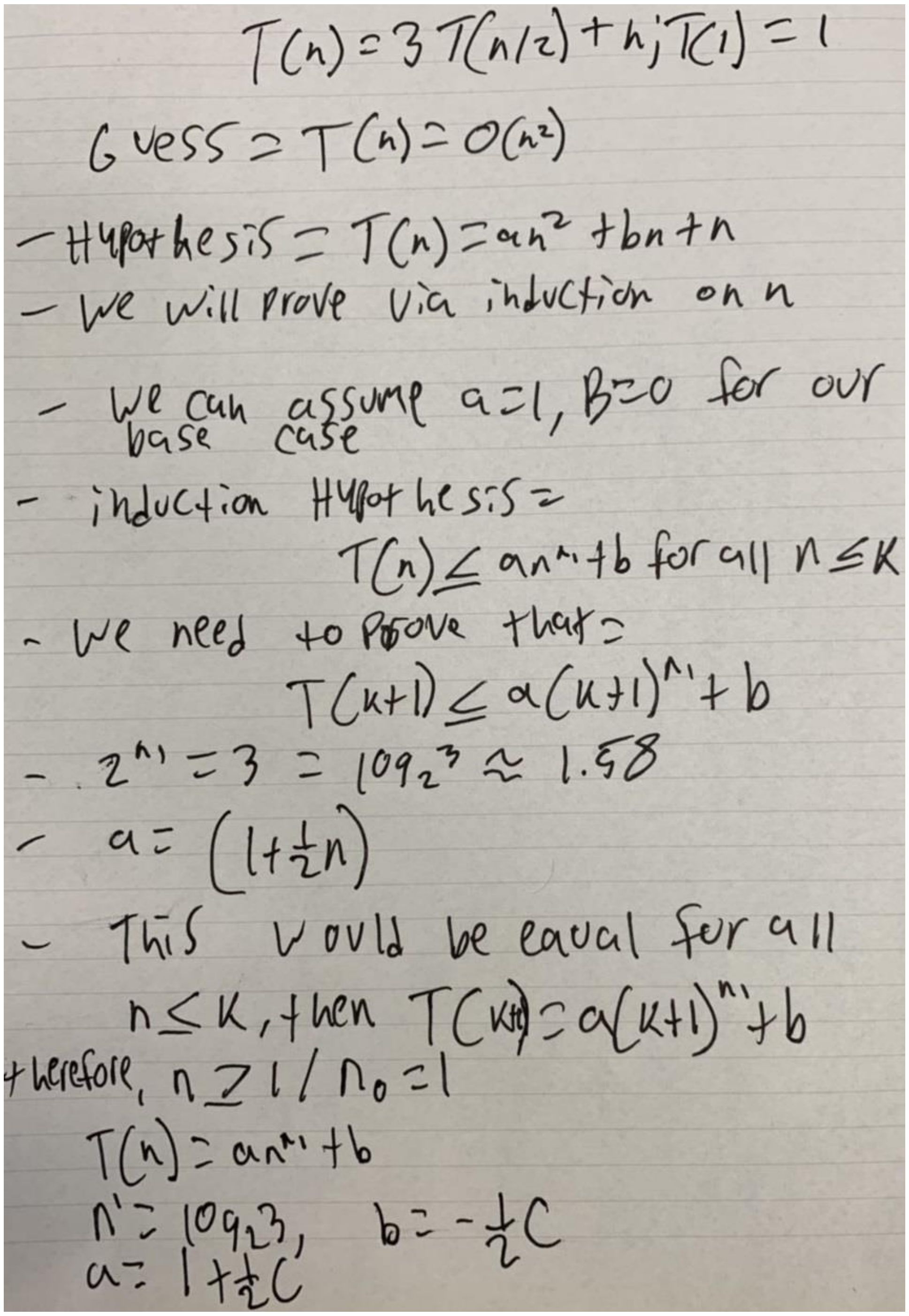
6. Solve the following three recurrences using the Master Method and state the order of complexity of the corresponding recursive algorithm. If you simply state the complexity order without showing your work, and the case that applies, you will not get any credit even if the complexity order is correct.

(a) $T(n)=3T(n/2)+n$ = for this example, we have $a=3$, $b=2$, and $f(n)=n$. so then we have $n^{logba} = n^{log3}$ and this would be an example of case 1. The final answer would be $\Theta(n^{log3})$

(b) $T(n)=3T(n/2)+n^{(\text{log of 3 to the base 2})}$ = for this example, we have $a=3$, $b=2$, and $f(n)=n^{(\text{log of 3 to the base 2})}$. so then we have $n^{logba} = n^{log3}$ and this would be an example of case 2 because neither one would be considered larger and they would be equal. The final answer would be $\Theta(n \lg n)$

(c) $T(n)=3T(n/2)+n^3$ = for this example, we have $a=3$, $b=2$, and $f(n)=n^3$ we can conclude that $3 < 2^3$ so that then case 1 would apply resulting in $\Theta(n^2)$

7. Suppose a recursive algorithm is characterized by these recurrences: T(n)=3T(n/2)+n; T(1)=1. Let a guessed solution be T(n)=O(n^2). Prove that this guess is correct using the Substitution Method. <u>State values of the constants n_0 and c that you determine as part of the proof</u>. <u>You must clearly show the three parts of the inductive proof</u> to get credit. Hints: (i) The goal of simplification in the Inductive Step is to get to a form T(n)≤(the expression you are trying to prove)–(another term) so that you can show that T(n)≤(the expression you are trying to prove) when (another term)≥0. (ii) Note that the value of n in any expression is such that n≥n_0.



$T(n) = 3T(n/2) + n; T(1) = 1$

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

- Hypothesis = $T(n) = an^2 + bn + n$
- we will prove via induction on n
- We can assume $a = 1$, $B = 0$ for our base case
- induction Hypothesis =

  $T(n) \le an^{n_1} + b$ for all $n \le K$
- We need to prove that =

  $T(K+1) \le a(K+1)^{n_1} + b$
- $2^{n_1} = 3 = \log_2 3 \approx 1.58$
- $a = (1 + \frac{1}{2}n)$
- This would be eaual for all $n \le K$, then $T(K+1) = a(K+1)^{n_1} + b$

therefore, $n \geq 1 / n_0 = 1$

$T(n) = an^{n_1} + b$

$n' = \log_2 3$, $b = -\frac{1}{2}C$

$a = 1 + \frac{1}{2}C$