Name:________________________________

WRITE YOUR NAME ABOVE

CHECK YOUR SCANTRON SHEET TO ENSURE THAT YOUR NAME (LAST NAME-SPACE-FIRST NAME), AND BANNER ID NUMBER ARE CORRECTLY AND FULLY FILLED

COMP 3270 — Introduction to Algorithms Fall 2021 Exam II

EXAM IS CLOSED TEXT AND NOTES

NO ELEXTRONIC DEVICES, INCLUDING LAPTOP, TABLET OR SMARTPHONE ARE ALLOWED.

A CALCULATOR IS ALLOWED.

A SMARTPHONE MAY NOT BE USED AS A CALCULATOR!

5 Problems 30 multiple choice questions 60 points 10% credit

Each question has exactly ONE correct answer.

Different questions have different difficulty levels.

Do not get stuck on any one question!

You have **75 minutes** to complete this test.

Time limit will be strictly enforced.

Begin this test only when you are instructed to do so.

When finished, return this exam with your scantron sheet inside

Problem 1: Consider the algorithm fragment below:

```
1. for i=1 to n–2
2.         A[i]=A[i]+1
3.         for j = 1 to i
4.                 A[j]=A[j]+1
5.                 for k = j to n
6.                         A[k]=A[k]+1
```

Answer the following questions regarding the approximate efficiency analysis of this fragment:

1. True or false? Steps 2, 4 & 6 have different Big-Oh complexities.

2. True or false? Under approximate analysis, steps 1, 3 & 5 have the same Big-Oh complexity.

3. What is the approximate Big-Oh complexity of this algorithm fragment?

A. O(n) B. O(n^2) C. O(n^3) D. O(n^4) E. none of these

Answer the following questions regarding the detailed efficiency analysis of this fragment:

4. True or false? Under detailed analysis, steps 2, 4 & 6 have different complexities, i.e., work done by them have different constant values.

5. What is the exact number of times step 1 will execute?

A.n–2 B. n–1 C. n D. n+1 E. none of these

6. What is the exact number of times step 3 will execute within each execution of the i-loop with a particular value of i?

A. n B. i–1 C. i D. i+1 E. none of these

7. What is the exact number of times step 5 will execute within each execution of the j-loop with a particular value of j?

A. n–j B. n–j+1 C. n–j+2 D. j E. none of these

8. What is the exact total number of times step 2 will execute?

A.n–2 B. n–1 C. n D. n+1 E. none of these

9. What is the correct summation that expresses the exact total number of times step 4 will execute?

A. $\sum_{i=1}^{i=n-2} i$ B. $\sum_{i=1}^{i=n-1} i$ C. $\sum_{i=1}^{i=n-2}(i+1)$ D. $\sum_{i=1}^{i=n-2} n$ E. none of these

10. What is the correct summation that expresses the exact total number of times step 6 will execute?

A. $\sum_{i=1}^{i=n-2}\sum_{j=1}^{j=i}(i+1)(n-j+1)$ B. $\sum_{i=1}^{i=n-2}\sum_{j=1}^{j=i} i(n-j+2)$

C. $\sum_{i=1}^{i=n-2}\sum_{j=1}^{j=i}(i+1)(n-j+2)$ D. $\sum_{i=1}^{i=n-2}\sum_{j=1}^{j=i} i(n-j+1)$ E. none of these

11. What is the degree of the polynomial T(n) for this algorithm fragment, i.e. highest exponent of n in it?

A. 2 B. 4 C. 1 D. 5 E. none of these

Problem 2: A recursive algorithm is characterized by the following two recurrence relations.

$T(n) = T(n/2) + T(n/4) + 2T(n/8) + 8n$; $T(1) = 8$

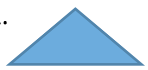
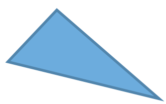
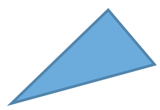
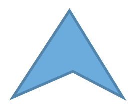
12. What is the branching factor (i.e., # of children of interior nodes) of its Recursion Tree?:

A. 1 B. 2 C. 3 D. 4 E. 5

13. True or false? In addition to the recursive calls, each execution of this recursive algorithm will do additional work that is linear in terms of the input size.

14. True or false? This is a recursive Divide & Conquer algorithm.

15. If you were to draw its Recursion Tree, with the children of each interior node ordered from left to right as the recursive call with half of the input, the recursive call with a fourth of the input and recursive calls with one-eights of the inputs, the shape of the Recursion Tree will be:

A.  B.  C.   D. E. none of these

Problem 3: Suppose we know the following about algorithms A1, A2 and A3, all of which correctly solve the same problem. A1 is $\Omega(n^2)$; A2 is $\Theta(n^2)$; A3 is $o(n^2)$. For each of the three statements below, decide if it is **True** (if so, mark A on Scantron), **False** (if so, mark B on Scantron) or **Can't Say** based only on the information given above (if so, mark C on Scantron).

16. A1 is the least efficient algorithm. C. Can't Say

17. A3 is the least efficient algorithm. B. False

18. A1 may be less or as efficient as A2. A. True

Now suppose you come to know that T(n) for A1 is $135n^3 + 2n + 1$ and T(n) for A2 is $3n^2 + 2n + 1$. Mark the following statements as **True (A)** or **False (B)** or **Can't Say (C)**.

19. A1 is $\Theta(n^3)$ A. True

20. A2 is $o(n^2)$ B. False

21. With everything stated above, you have all the information needed to rank the three algorithms A1-A3 from the most efficient to the least efficient. What is this rank ordering?

A. A1, A2, A3 B. A2, A1, A3 C. A3, A2, A1 D. A3, A1, A2

E. A2, A3, A1

Problem 4: Correctness proofs. Consider these two – one iterative and the other recursive – algorithms.

Factorial (n: integer>0)

1. product=1
2. for i=1..n do
3. product= product*i
4. return product

Binary-Tree-Height(T: Binary Tree Root Node) //A null (empty) tree is not a valid input

1. if T.left == NULL and T.right == NULL then return 0
2. if T.left != NULL and T.right != NULL then
3. return Larger(Binary-Tree-Height(T.left), Binary-Tree-Height (T.right)) + 1 //Larger(x,y) returns larger of x and y
4. if T.left != NULL then return Binary-Tree-Height(T.left)+1
5. if T.right != NULL then return Binary-Tree-Height(T.right)+1

22. What is the most accurate Loop Invariant to prove the correctness of **Factorial**?

A. Before the i-th execution of the for loop in steps 2-3, with i=1,2,3,..,n, product will contain the factorial of integer i.

B. Before the i-th execution of the for loop in steps 2-3, with i=1,2,3,..,n, product will contain the factorial of integer n.

C. Before the i-th execution of the for loop in steps 2-3, with i=1,2,3,..,n, product will contain the factorial of the variable product.

D. Before the i-th execution of the for loop in steps 2-3, with i=1,2,3,..,n, product will contain the factorial of integer (i–1).

E. None of these Loop Invariants are correct.

23. Which mathematical property of trees can be used in the Base Case Proof of the Inductive Correctness Proof of **Binary-Tree-Height**?

A. The height of a binary tree is the length of the longest path from the root to a leaf.

B. The height of a binary tree is the length of the longest path from the deepest leaf to the root.

C. The height of a binary tree with only a root node and no children is 0.

D. The height of a binary tree is the same as its depth.

E. None of these statements can be used in the Base Case Proof, unfortunately!

24. What is the most accurate Inductive Hypothesis to prove the correctness of **Binary-Tree-Height**?

A. Suppose the algorithm returns the correct height of a binary tree with no nodes.

B. Suppose the algorithm returns the correct height of a binary tree with one node, the root.

C. Suppose the algorithm returns the correct height of a binary tree with n nodes.

D. Suppose the algorithm returns the correct height of a binary tree with k nodes.

E. Suppose the algorithm returns the correct height of a binary tree with 1,2,3,...k nodes for some value of k>1.

25. True or False? More than one case need be considered by the Inductive Step of the Inductive Correctness Proof of **Binary-Tree-Height**.

Problem 5: Recurrence relations and detailed analysis of recursive algorithm efficiency

g(n: non-negative integer)

1. if $n \le 1$ then return n
2. else return (5 * g(n–1) – 6 * g(n–2))

26. What are the two recurrences of recursive algorithm **g**?

A. $T(n)=5$, $n\le1$; $T(n)=T(n-1)+T(n-2)+11$, $n>1$

B. $T(n)=4$, $n\le1$; $T(n)=T(n-1)+T(n-2)+10$, $n>1$

C. $T(n)=6$, $n\le1$; $T(n)=T(n-1)+T(n-2)+12$, $n>1$

D. $T(n)=1$, $n\le1$; $T(n)=T(n-1)+T(n-2)+1$, $n>1$

E. none of the above

MergeSort divides the array to be sorted into **two** equal halves, calls itself recursively on each half to sort that subarray, and then calls the Merge algorithm to merge the two sorted halves in linear time. This leads to its two recurrence relations $T(n)=2T(n/2)+cn$, $n>1$; $T(1)=c$. We solved these recurrences using the Recursion Tree method in class to show that $T(n)=cn\log_2 n+cn$ and therefore its complexity order is $\Theta(n\log_2 n)$. Now consider the question of whether Merge Sort will be faster if we split the array into **three** equal one-thirds, made a recursive call on each one-third to sort it and used a modified Merge algorithm to merge the three sorted subarrays into one wholly sorted array. Assuming that we can merge three sorted one-thirds in linear time, i.e., with the same cn work, the corresponding recurrences are:

$T(n)=3T(n/3)+cn$, $n>1$; $T(1)=c$.

27. What is the complexity order of this new MergeSort algorithm, that you obtain by applying the Master Theorem?

A. $\Theta(n)$ B. $\Theta(n\lg n)$ C. $\Theta(cn)$ D. $\Theta(n\log_3 n)$ E. Master Theorem cannot be applied

28. Now solve the recurrences of this modified MergeSort using the more precise Recursion Tree method. The corresponding table is shown on the next page. It can be seen from that table that total work done by all recursive executions = T(n) = cn * (_______________). What is the correct expression to fill in this blank?

A. $\lg n$ B. $\log_3 n$ C. $\log_3 n+1$ D. $\lg n+1$ E. none of these are correct

29. Based on this analysis, which of the statement below is most accurate (ignoring the system cost of keeping track of recursive calls using a system stack)?

A. The modified MergeSort algorithm will take the same time to execute in practice as the original MergeSort algorithm for the same input size.

B. The modified MergeSort algorithm will take less time to execute in practice than the original MergeSort algorithm for the same input size.

C. The modified MergeSort algorithm will take more time to execute in practice than the original MergeSort algorithm for the same input size.

D. The modified MergeSort algorithm will take less time to execute in practice than the original MergeSort algorithm for the same input size only if the constant c is the same for both algorithms.

E. We do not have the information to determine which algorithm will run faster in practice.

30. Solve $T(n)=T(n-1)+1$; $T(0)=1$ by forward or backward substitution to obtain T(n). What is it?

A. $T(n)=n$ B. $T(n)=n^2-1$ C. $T(n)=n^2+1$ D. $T(n)=n-1$ E. $T(n)=n+1$

Recursion Tree Method Table for solving T(n)=3T(n/3)+cn, n>1; T(1)=c.

level	Level number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each recursive execution, excluding the recursive calls	Total work done by the algorithm at this level = column3*column5
0	0	3^0	$n/3^0$	$c(n/3^0)$	cn
1	1	3^1	$n/3^1$	$c(n/3^1)$	cn
2	2	3^2	$n/3^2$	$c(n/3^2)$	cn
The level just above the base case level	$(\log_3 n\text{-}1)$	$3^{(\text{logntobase3-1})}$ = $(n^{\text{log3tobase3}})/3 = n/3$	$n/3^{(\text{logntobase3-1})}=3$	c(3)	cn
Base case level	$\log_3 n$	$3^{\text{logntobase3}}$ = $n^{\text{log3tobase3}}$ = n	$n/3^{\text{logntobase3}}=1$	c(1)	cn