Name _____

CSCI 332, Fall 2024

Exam 2—Practice 2

Note that this exam has four sections. The first section covers algorithm analysis (20 points), the second section covers greedy algorithms (20 points), the third section covers divide and conquer algorithms (20 points), and the fourth section covers dynamic programming algorithms (20 points). If you need more space, develop your solution on scratch paper before copying your final answer to the exam paper.

Good luck!

Section 1 (Algorithm Analysis)

1. (5 points) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

   - $f_1(n) = n!$ (n factorial)
   - $f_2(n) = n^2$ (n squared)
   - $f_3(n) = 2^{n!}$ (2 to the power of n factorial)
   - $f_4(n) = 2^3 n$ (two to the power of 3 times n)
   - $f_5(n) = \log_2 n^2$ (log base 2 of n squared)

2. (5 points) Suppose you know that an algorithm has a worst-case runtime that is $\Omega(n)$. For each of the following, decide whether it is definitely true, definitely false, or could be true or false and circle your choice.

   - The algorithm's best-case runtime is $\Omega(\log n)$. T, F, T or F
   - The algorithm's best-case runtime is $\Omega(n \log n)$. T, F, T or F
   - The algorithm's worst-case runtime is $O(n^2)$. T, F, T or F
   - The algorithm's worst-case runtime is $O(n)$. T, F, T or F
   - The algorithm's worst-case runtime is $\Omega(\log n)$. T, F, T or F

Recall Dijkstra's algorithm to compute the shortest path distance from one node s to all other nodes in a directed graph with non-negative edge weights:

> Dijkstra(directed graph $G = (V, E)$ with non-negative edge weights; node s)
> Let S be the set of explored nodes
> For each $u \in S$, we store a distance $d(u)$
> Initially $S = \{s\}$ and $d(s) = 0$
> While $S \neq V$:
> Select a node $v \notin S$ with at least one edge from S for which
> $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
> Add v to S and define $d(v) = d'(v)$

3. (3 points) How many times *exactly* does the while loop run? (This should be a function of m, the number of edges, and/or n, the number of nodes.)

4. (3 points) Suppose that we implement the while loop naively by examining every $v \notin S$, computing $\min_{e=(u,v):u \in S} d(u) + \ell_e$, and then using those values to find the next node to add to S. Describe a a worst-case input for this implementation.

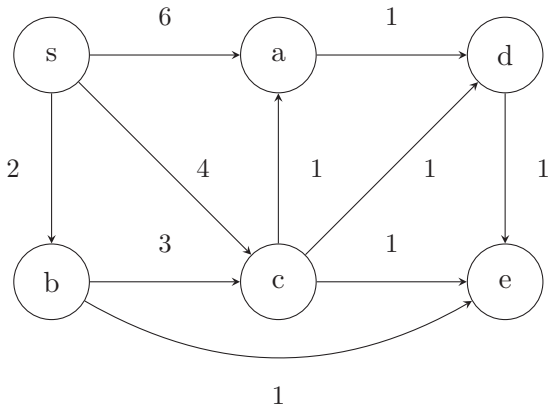
5. (4 points) Assuming this naive implementation of the while loop, give a $f(n)$ such that the worst-case runtime of Dijkstra's algorithm on a graph with n nodes is $\Theta(f(n))$.

Section 2 (Greedy Algorithms)

Dijkstra's algorithm is copied here again for your convenience:

> Dijkstra(directed graph $G = (V, E)$ with non-negative edge weights; node s)
> Let S be the set of explored nodes
> For each $u \in S$, we store a distance $d(u)$
> Initially $S = \{s\}$ and $d(s) = 0$
> While $S \neq V$:
> Select a node $v \notin S$ with at least one edge from S for which
> $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
> Add v to S and define $d(v) = d'(v)$

6. (20 points) Use the table below to trace through the execution of Dijkstra's algorithm on the following graph. Note that it continues on the next page! You may not need all of the rows.

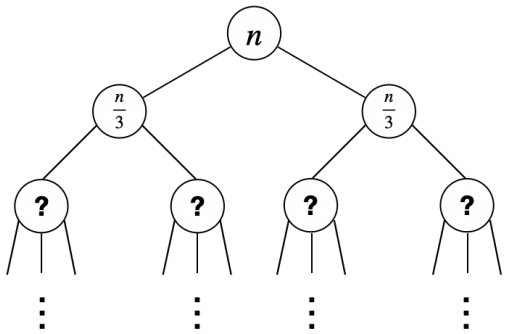


	current S	current $d(u)$ values for $u \in S$	all $v \notin S$ with at least one edge from S	values of $d'(v)$	v to add to S
set up					
while loop run 1					
while loop run 2					

	current S	current $d(u)$ values for $u \in S$	all $v \notin S$ with at least one edge from S	values of $d'(v)$	v to add to S
while loop run					
while loop run					
while loop run					
while loop run					
while loop run					
while loop run					

Section 3 (Divide and Conquer)

Consider the recursion tree below, which represents the recursive calls (each node) and size of the input to the recursive calls (text inside each node).



In general, we can write a recurrence relation as

$$T(n) = aT(n/b) + f(n)$$

where $T(n)$ is the runtime of the algorithm on an input of size n and $f(n)$ is the non-recursive part of the algorithm (i.e., any preprocessing before or postprocessing after the recursive call(s)).

7. (3 points) What are a and b for this recurrence relation?

8. (3 points) What is the depth of the recursion tree, as a function of n?

9. (4 points) Assume that $f(n) = n \log n$. At the following levels of the recursion tree, how many computations are being done over all calls at that level?

   (i) Level 0

   (ii) Level 1

   (iii) Level 2

   (iv) Level 3

The pseudocode to binary search for the location after which to insert a new element t into a sorted array A, indexed starting at 0, is shown below. You may assume that A will always have at least one element.

```
Binary_Search(sorted array A, new element t):
    min = 0
    max = length of A - 1
    make A and t global
    return Binary_Search_Indices(min, max)
```

```
Binary_Search_Indices(index min, index max):
  If mid = max:
        If t < A[min]:
              Return min -1
        Else:
              Return min
  Else:
        mid = ⌊ min + max / 2 ⌋
        If t < A[mid]:
              return Binary_Search_Indices(min, mid - 1)
        Else
              return Binary_Search_Indices(mid + 1, max)
```

10. (4 points) Suppose you have the length 16 array $A =$ (with indices labeled above)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	5	6	12	100	105	108	133	229	277	280	551	559	601	603

and new element $t = 295$ and call Binary_Search(A, t). Exactly how many calls to Binary_Search_Indices will be made? (You should give a number here.)

11. (3 points) What is the recurrence relation for the runtime of Binary_Search_Indices on an array of length n?

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ \underline{\qquad\qquad\qquad\qquad\qquad\qquad} & \text{if } n > 1. \end{cases}$$

12. (3 points) What is the worst-case runtime of Binary_Search given above?

Section 4 (Dynamic Programming)

Suppose that you are the booking agent for breakout pop star Temple Raye. The possible gigs that you are considering can be divided into *arena* venues or *club* venues, and you have the option of booking an arena or a club every weekend. Arenas generally pay much more than clubs. However, the prep time for an arena show is longer. In particular, if you schedule Temple to play an arena, she can't play any concert (arena or club) the previous weekend. On the other hand, if you schedule her in a club, it's okay if she plays in arena or a club the weekend before.

Given a sequence of n weekends, a *plan* is a choice of club, arena, or none for the gigs for each of the n weekends. A plan is only valid if, for any weekend where an arena is chosen, none is chosen for the previous weekend. The revenue from the plan is the sum of the revenues from the chosen gigs. (Choosing none adds \$0 to the revenue.) Your goal is to find an optimal plan—that is, one that is both valid and has maximal revenue.

For weekend i, we write c_i to mean the revenue from the club show that you can book that weekend, and a_i to mean the revenue from the arena show you can book that weekend.

Here is an example input to the problem.

	weekend1	**weekend2**	**weekend3**	**weekend4**	**weekend5**
c	10	8	2	10	24
a	11	100	40	5	50

So, for this example, c_2 is 8 and a_4 is 5. An optimal plan for this input is to choose no show for weekend 1, the arena show for weekend 2, the club show for weekend 3, no show for weekend 4, and arena show for weekend 5, with a total revenue of $0 + a_2 + c_3 + 0 + a_5 = 0 + 100 + 2 + 0 + 50 = 152$.

13. (6 points) Give a recurrence relation for the revenue from the best possible concert schedule up to week j.

$$OPT(j) = \begin{cases} \underline{\qquad\qquad\qquad\qquad} & \text{if } j = 0 \\ \underline{\qquad\qquad\qquad\qquad} & \text{if } j = 1 \\ \underline{\qquad\qquad\qquad\qquad} & \text{if } j > 1. \end{cases}$$

14. (7 points) Give a polynomial time, recursive algorithm to compute the revenue of the best possible concert schedule using your recurrence relation above. You should fill in the following two functions so that Max_Revenue returns the maximum revenue of the best possible concert schedule.

Max_Revenue($a_1, a_2, \ldots, a_n, c_1, c_2, \ldots, c_n$):

Compute_OPT(j):

15. (3 points) Fill in the rest of a recursive algorithm to find the optimal set of gigs to choose, given a full set of OPT values for $j = 0$ through $j = n$.

Nodes(j):