Name ____

CSCI 332, Fall 2024
Exam 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 $O(g(n))$.

   - $f_1(n) = \sqrt[3]{n}$ (third root of n)
   - $f_2(n) = \frac{n!}{n}$ (n factorial divided by n)
   - $f_3(n) = n \log_2 n$ (n times log base two of n)
   - $f_4(n) = 3n$ (3 times n)
   - $f_5(n) = \log_2 n$ (log base 2 of n)

2. (5 points) Suppose you know that an algorithm has a worst-case runtime that is $O(n^2 \log n)$. 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 worst-case runtime is $\Omega(n^2)$. T, F, T or F
   - The algorithm's worst-case runtime is $\Omega(n^3)$. T, F, T or F
   - The algorithm's best-case runtime is $O(n^2)$. T, F, T or F
   - The algorithm's worst-case runtime is $O(n^3)$. T, F, T or F
   - The algorithm's worst-case runtime is $\Theta(n^2 \log n)$. T, F, T or F

The *1-dimensional closest points problem* takes in an array of values and looks for the distance between the two closest values in the array. For example, the array $[7, 0, -10, -8, 4, 9]$, contains values 7 and 4, which differ by 3. There is no other pair of values in the array that are closer, so the answer to the 1-dimensional closest points problem on this input is 3. We will assume that our array indices start at 1 for this problem.

```
1D-ClosestPoints(array $A$ of length $n$ indexed starting at 1):
    Let min = $\infty$
    For $i$ in 1 to $n$:
        For $j$ in 1 to $n$:
            If $i \neq j$:
                If $|A[i] - A[j]| <$ min:
                    min = $|A[i] - A[j]|$
    Return min
```

3. (2 points) Describe a worst-case input of size n for 1D-ClosestPoints.

4. (3 points) Give a function $f(n)$ such that the worst-case runtime of 1D-ClosestPoints is $\Theta(f(n))$.

Your friend notices that 1D-ClosestPoints unnecessarily compares all values in A twice and propopses the following algorithm instead.

```
1D-ClosestPoints-Improved(array $A$ of length $n$):
    Let min = $\infty$
    For $i$ in 1 to $n$:
        For $j$ in $i + 1$ to $n$:
            If $|A[i] - A[j]| <$ min:
                min = $|A[i] - A[j]|$
    Return min
```

5. (5 points) Give a function $g(n)$ such that the worst-case runtime of 1D-ClosestPoints-Improved is $\Theta(g(n))$.

Section 2 (Greedy Algorithms)

Suppose that you will drive your car for a long trip between New York City and San Francisco along a pre-specified path. In preparation for your trip, you have downloaded a map that contains the distances in miles between all the gas stations in your route. Assume that your car's gas tank, when full, holds enough gas to travel n miles. Assume that the value n is given, and that you want to make the minimum number of stops possible along the way, without running out of gas at any point. Your friend proposes a greedy algorithm for selecting which gas stations to stop at:

- Start your trip with a full tank.
- Check your map to determine the farthest away gas station in your route within n miles.
- at that gas station, fill up your tank and check your map again to determine the farthest away gas station in your route within n miles from this stop.
- Repeat the process until you get to San Francisco.

Let $s_g(j)$ denote the station where we make the j^{th} stop for the greedy algorithm. For example, if $s_g(2) = 7$ it means that we make the 2nd stop at the 7th gas station. Let $s_O(j)$ be the index of the gas station for the j^{th} stop for an optimal solution to the problem.

We know that if we can prove that the greedy algorithm "stays ahead" of the optimal, then we can prove that the greedy algorithm is, in fact, optimal.

6. (2 points) In words, what would it mean for the greedy algorithm to stay ahead of the optimal solution for this problem? (Your answer should probably use the phrase "gas station" at least once.)

7. (12 points) Fill in the blanks.

   We prove that that the greedy algorithm stays ahead using induction. Suppose that the greedy algorithm uses k stops and the optimal solution uses m stops. By definition $k \leq m$.

   Claim 1. *For all $r \leq k$, we have* ______________.

   Proof: Let $r \leq k$.

   Assume that ____.
   (This is the inductive hypothesis, IH.)

   There are two cases:

If $r = 1$, we know that __________________________ because the greedy algorithm chooses the farthest possible gas station for the first stop, so the optimal solution cannot use a closer gas station.

If $r > 1$, we know by the IH that __________________________. We want to show that the next gas station chosen by the greedy algorithm must be farther along than the next gas station in the optimal solution. By definition, the greedy algorithm picks the farthest gas station within n miles of $s_g(r-1)$. Since $s_O(r-1) \leq s_g(r-1)$, it must be that __________________________.

Because the claim is true in all cases, it holds for all $r \leq k$. □

Dijkstra's algorithm uses a greedy approach to compute the shortest distance from some node v to all other nodes in the graph.

Suppose that you only want the distance from v to some specific node u. Your friend proposes the following algorithm to find the shortest path from node u to node v, based on Dijkstra's algorithm.

```
Shortest_Path(directed graph $G$ with positive weights on edges, start node $v$, end node $u$):
  Let $P$ be an empty list to hold the chosen path
  Let $x$ be the current node and set $x = v$
  While current node $x$ is not equal to $u$:
      Choose the edge leaving $x$ with smallest edge weight; call its endpoint $y$
      Add $y$ to $P$
      $x = y$
  Return $P$
```

8. (6 points) Either briefly explain why this algorithm successfully finds the shortest path from v to u on all valid input graphs, or give an example where it fails to find the shortest path.

Section 3 (Divide and Conquer)

Recall the *1-dimensional closest points problem* from Section 1. Previously, you saw two iterative algorithms for that problem. You will now work to give a divide and conquer algorithm for it.

You notice that if you divide the array into the first half A and the second half B, then the minimum distance between points can come from three places:

- Pairs in A
- Pairs in B
- Pairs where one value comes from A and one value comes from B.

Just as in the significant inversions problem, you notice that you can calculate the minimum distance between pairs where one value comes from A and one value comes from B while you perform a merge of two sorted arrays.

```
Merge-and-Min-Distance(sorted arrays $A$ and $B$ of length $n$):
  Let $C$ be an empty array (to store the final sorted array)
  Let pointers $a_1$ and $b_1$ be pointers to first element of $A$ and $B$, respectively
  Let min $= \infty$ (to store the final minimum distance)
  While $i, j \leq n$:
      If $a_i < b_j$
          Append $a_i$ to $C$
          If distance between $a_i$ and $b_j$ less than min, update min accordingly.
          Move $a_i$ to $a_{i+1}$
      Else:
          Append $b_j$ to $C$
          If distance between $a_i$ and $b_j$ less than min, update min accordingly.
          Move $b_j$ to $b_{j+1}$
  Return $C$, min
```

a_1

2	9	12	25

Left half, A

b_1

-4	-1	8	15

Right half, B

-4							

9. (3 points) After the first iteration of the while loop in Merge-and-Min-Distance on A and B above, min=6.

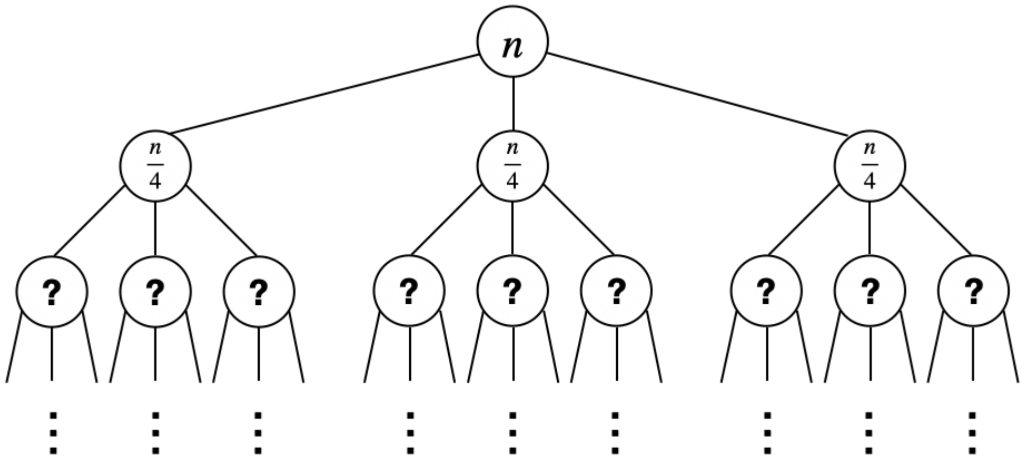
   Fill in the values that min has after the remaining 7 iterations. You may use the blank array above to help in your calculations if it is useful to you.

   min=<u>6</u>, min=___, min=___, min=___, min=___, min=___, min=___, min=___.

10. (7 points) Notice that Merge-and-Min-Distance takes $\Theta(n)$ time on an A, B of length n. Fill in the rest of a recursive algorithm to compute the minimum distance between two points in an array of length n. Your algorithm should take $O(n \log n)$ time on a length n array, should call Merge-and-Min-Distance, and should be recursive (that is, it should call itself at least once).

```
1D-ClosestPoints-Recursive(array $A$ of length $n$):
```

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)).

11. (3 points) What are a and b for this recurrence relation?

12. (3 points) What is the depth of the recursion tree, as a function of n?

13. (4 points) Assume that $f(n) = n^2$. 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

Section 4 (Dynamic Programming)

As snow piles up in the mountains (at least somewhere), you are thinking about ski season. You would like to be able to choose a route through a given ski jumping course that allows for the maximum total distance in the air. The input to your problem is a set of n jumps at positions $s_1, s_2, \ldots, s_n$ on the slope along with jump distances $d_1, d_2, \ldots, d_n$ which indicate how far each jump will send you. Some jumps may launch you over other jumps, but you can assume that you can always land safely and complete the next jump. You can also choose to skip a jump if doing so will allow you to take other jumps that give you more air.

14. (4 points) Suppose that the input is (in meters) $s_1 = 100, s_2 = 150, s_3 = 200, s_4 = 300, s_5 = 350$ and $d_1 = 60, d_2 = 10, d_3 = 70, d_4 = 110, d_5 = 30$. What is the optimal set of jumps, and what is your total air distance?

15. (6 points) For many dynamic programming problems, we have defined a value $p(j)$ for each input element j. We would like to define a similar value for this problem. In words, what does $p(j)$ mean for the ski jump problem? After defining it, compute $p(1), p(2), p(3), p(4)$ and $p(5)$ for the instance from (a).

16. (4 points) Give a recurrence relation for the maximum air distance.

$$OPT(j) = \begin{cases} \rule{5cm}{0.4pt} & \text{if } j = 0 \\ \rule{5cm}{0.4pt} & \text{if } j > 1. \end{cases}$$

17. (4 points) Give a polynomial time, recursive algorithm to compute the weight of the heaviest squared independent set using your recurrence relation above. You should fill in the following two functions so that Max_Jump_Distance returns the longest total jump distance over any set of compatible jumps in the input.

```
Max_Jump_Distance($s_1, s_2, \ldots, s_n, d_1, d_2, \ldots, d_n$):
```

```
Compute_OPT($j$):
```

18. (2 points) Analyze the runtime of your algorithm. That is, give a function $f(n)$ such that the worst-case runtime of Max_Jump_Distance on an input of size n is $\Theta(f(n))$.