Please enter your name and uID below.

Name:
uID:

Submission notes

- Due at 11:59 pm on Friday, September 11.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Big-Θ) Prove that $\log(n!) = \Theta(n \log(n))$.

2. (Party Planning) You are planning a (post-COVID) party for your friends. Even though everyone on the potential invite list is friends with you, not all of your friends are friends with each other (but we assume you do know the friendships among your friends). You want to invite as many friends as possible to the party, *but* you want to make sure everyone invited is friends with at least 5 other people on the guest list (so no one is lonely).

   Consider the following algorithm for determining the largest subset of your friends that you can invite without having to worry about lonely guests. The algorithm begins by assuming everyone will be invited, then iteratively "un-invites" anyone who doesn't have enough friends on the guest list.

Algorithm 1: Party Planning Algorithm

Input: List<Friendships> $friendships$
Result: Set<Friends>

$invited \leftarrow$ `initialize`$(friendships)$
while $\exists\ p \in invited$ *with* < 5 *friends* $\in invited$ **do**
  `uninvite`(p)
end
return `convert_to_set`$(invited)$

   (a) Prove this algorithm returns the largest possible invite list without any lonely friends (defined as those knowing less than 5 other people on the guest list).

   (b) Suppose there are n friendships among your friends. Which data structure (or system of data structures) should you use for *invited* so that the Party Planning Algorithm runs in $O(n)$ time? Argue your answer is correct. Note that you will have to describe which operations you can use for `initialize`, `uninvite`, and for checking if there is a person with less than 5 invited friends.

3. (Amortized ArrayList) Suppose you are working with an ArrayList implementation which increases its size by some huge constant amount c (e.g. one million) whenever it becomes full via `add_end` operations. That is, whenever it reaches its capacity n, it allocates a new array of size $n + c$ and copies all of its elements over.

   Prove that the runtime of k `add_end` operations is *not* $O(k)$ when using this ArrayList implementation.

4. (Sorting) Consider the following modification to mergesort, called `mergesort2`:

```
void mergesort2(int[] a, int left, int right) {
   if (left < right && !sorted(a, right)) {
      int mid = (left + right)/2;
      mergeSort2(a, left, mid);
      mergeSort2(a, mid+1, right);
      merge(a, left, mid, right);
   }
}

boolean sorted(int[] a, int right) {
   for (int i = 0; i < right-1; i++) {
      if (a[i] > a[i+1]) {
         return false;
      }
   }
   return true;
}
```

The idea behind this modification is that if a portion of the array is already sorted, then we don't need to recurse.

For full credit on this problem, prove the time complexity for `mergesort2` is $O(n^2)$. For partial credit, prove it is $O(n^2 \log(n))$.

5. (Erickson 1.37) For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. See the figure below for an example.

   (a) Provide pseudocode for a recursive algorithm to compute the largest complete subtree of a given binary tree. Your algorithm should return both the root and the depth of this subtree.

   (b) Prove your algorithm is correct.

   (c) Analyze your algorithm's worst-case asymptotic complexity.