Homework 4

Math 182, Winter 2025

When designing an algorithm, you must include the following:

1. A short (1-2 sentence) sketch of the main idea of the algorithm.
2. The algorithm itself, written in pseudo-code.
3. The runtime of the algorithm.
4. A proof that the algorithm is correct.
5. A proof that the runtime is correct.

If you wish to sort a list, assume that you can do so in $O(n \log n)$ time; you do not need to write out the details of the sorting algorithm. If you wish to use an algorithm we have discussed in class (eg BFS/DFS, Interval Scheduling, Dijkstra's Algorithm, etc) you may do so without elaboration; if you need to modify the algorithm, you should provide sufficient details for the modification in your pseudocode.

Question 1:

Consider the following algorithm.

The Stoogesort algorithm (by Jeff Erickson)

```
Sort(A):
  Stoogesort(A, 1, length(A))

Stoogesort(A, i, j):
  if j - i = 1 and A[i] > A[j]:
    Swap(A, i, j)
  if j - i > 1:
    t = ⌊(j - i+1)/3)⌋
    Stoogesort(A, i, j-t)
    Stoogesort(A, i+t, j)
    Stoogesort(A, i, j-t)

Swap(A, i, j):
  temp = A[i]
  A[i] = A[j]
  A[j] = temp
```

(a) Does the algorithm work correctly? If so, prove it. If not, find a list which it does not sort.

(b) What is the running time of the algorithm?

Question 2:

Every day, you and a friend walk from your apartments or dorms to your first class. You are both busy and don't often see each other, so it is important that your routes cross, so that you have a chance to see each other on the walk. But being busy, you want the path to class to be as short as possible.

In mathematical terms: assume you are given an undirected weighted graph $G = (V, E)$, where the vertices are intersections of streets in the city, edges represent roads between those intersections, and the weight of each edge is the time it takes to walk down that road. You are given vertices s_1, t_1 representing your home and first class, and s_2, t_2 representing the home and first class of your friend.

Design an efficient algorithm to find paths from s_1 to t_1 and from s_2 to t_2 such that the paths meet in at least one vertex v, and the sum of the weights of the two paths is as small as possible.

For convenience, you only need to report the lengths of the paths (rather than the paths themselves), although you should think about how to find the actual paths as well. If no such paths exist, your algorithm should return ∞. For full credit, your algorithm should run in $O((|V| + |E|) \log(|V|))$ time.

Question 3:

You are trying to focus a microscope. Assume that there are n focus settings on a dial, one of which has the least blurriness. If you turn the dial too far forward or too far back from that setting, the image will become increasingly blurry. Design an algorithm to efficiently find the setting with the best focus.

Mathematically, represent the settings by $1, \ldots, n$. There is a function `Blurriness(i)` that reports an integer representing the blurriness of the image when the knob is at setting i. There is a unique number $k \in \{1, 2, \ldots, n\}$ that minimizes this function. The function `Blurriness(i)` is strictly decreasing for $i \leq k$, and strictly increasing for $i \geq k$. Design an algorithm to find k. For full credit, this algorithm should run in $O(\log n)$ time.

Question 4:

Given a list L, an *inversion* is a pair $i < j$ such that $L[i] > L[j]$. An inversion is *special* if $L[i] > 2L[j]$. Design an algorithm to count the number of special inversion in a list. (As a starting point, look at the algorithm for counting the number of inversions in a list given in Kleinberg & Tardos. You will need to modify the merging step.)