You must hand in this homework. You can work together on homeworks, but you must write up solutions by yourself. Getting solutions from the web is forbidden. Exams and homeworks are similar, so it is in your best interest to try to solve things by yourself.

This homework is the same as in the non-honors class, with the addition of (possibly) more challenging problems, indicated by (**).

1. **Recurrences**

Solve the following recurrences. You can use the Master Theorem, where applicable, but state which case you are using. If you cannot use the Master Theorem, try to explain your strategy. Big-O notation is fine.

(a) \( T(n) = 2T(n/3) + n \).
(b) \( T(n) = 2T(n/3) + 1 \).
(c) \( T(n) = 9T(n/2) + n^2 \).
(d) \( T(n) = T(n/4) + 3 \).
(e) \( T(n) = 3T(n/2) + 1 \).
(f) \( T(n) = 2T(n-1) + 1 \).
(g) \( T(n) = 9T(n/3) + \sqrt{n} \).
(h) \( T(n) = 64T(n/2) + 2^n \).
(i) \( T(n) = T(n/4) + T(3n/4) + 2n \).
(j) (** \( T(n) = T(\sqrt{n}) + \log \log n \).**

2. Design a divide and conquer algorithm for finding the first and second largest items in a list \( \{a_1, a_2, \ldots, a_n\} \) of \( n \) numbers. Analyze the running time of your algorithm.

3. **Stooge Sort**

Professor Randall thinks she has a new sorting algorithm. Here is the proposed algorithm. The input is a list \( A[1 \ldots n] \) of \( n \) numbers, where \( n \) is a power of 2.

\[
\text{RandallSort}(A) \\
1 \quad \text{if } \text{length}(A) = 1, \\
2 \quad \text{then return } A \\
3 \quad \text{RandallSort}(A[1 \ldots n/2]) \\
4 \quad \text{RandallSort}(A[n/2 + 1 \ldots n])
\]
for $i = 1 \rightarrow n/2$

if $A[i] > A[i + n/2]$

then Swap($A[i], A[i + n/2]$)

RandallSort($A[1 \ldots n/2]$)

RandallSort($A[n/2 + 1 \ldots n]$).

Swapping($k, \ell$)

$\text{temp} \leftarrow A(\ell)$

$A[\ell] \leftarrow A[k]$  

$A[k] \leftarrow \text{temp}$

(a) Analyze the running time of RandallSort by stating and solving the appropriate recurrence.

(b) Does the algorithm sort correctly? If yes, argue why. If no, give an example for which the above algorithm does not sort correctly.

4. You are consulting for a small investment company. They give you a price of Google’s shares for the last $n$ days. Let $p(i)$ represent the price for day $i$. During this time period, the company wanted to buy a fixed number of shares on some day and sell all these shares on some later day. The company wants to know when they should have bought and when they should have sold the shares in order to maximize the profit. If there was no way to make money during the $n$ days, you should report this instead.

For example, suppose $n = 3$, $p(1) = 9$, $p(2) = 1$, $p(3) = 5$. Then you should return buy on day 2, sell on day 3.

Your goal is to design a divide-and-conquer algorithm for this problem that runs in time $O(n)$.

(a) In one sentence, describe a simple algorithm for the problem that takes time $O(n^2)$.

(b) Use the same divide strategy as in Merge Sort. Consider two cases: (i) there is an optimal solution in which the investors are holding the stock at the end of day $n/2$; (ii) there no such optimal solution. For each case, express the optimal solution in terms of something you can compute on the two sublists.

(c) State your algorithm and analyze its running time.

5. You are given $n$ groundhogs of different sizes and $n$ corresponding holes (burrows). You are allowed to try to stick a groundhog into a hole, from which you can determine whether the groundhog is larger than the hole, smaller than the hole, or matches the hole exactly. However, there is no way to compare two groundhogs together (they squirm too much) or two holes together (you dont have a ruler with you). The problem is to match each groundhog to its hole. Design a randomized divide-and-conquer algorithm for this problem with expected efficiency $O(n \log n)$.

(a) State the algorithm.

(b) State the recurrence for the running time and solve it.
6. (***) A permutation $\sigma$ is a bijection from the set $\{1, 2, ..., n\}$ to itself where $\sigma(i)$ is the position that element $i$ is mapped to. For example, if $n = 5$, a permutation on the numbers $(1, 2, ..., 5)$ might be $(2, 1, 5, 3, 4)$, where $\sigma(1) = 4$ since element 1 moved to the fourth position. We say that elements $i$ and $j$ are neighbors in a permutation if $|\sigma(i) - \sigma(j)| = 1$. Every permutation can be decomposed into a smallest number of nearest neighbor transpositions which start at the identity $(1, 2, ..., n)$ and iteratively interchange two neighbors. We call the number of steps required the distance $d(\sigma)$. In our example, we can move $(1, 2, 3, 4, 5) \rightarrow (2, 1, 3, 4, 5) \rightarrow (2, 1, 3, 5, 4) \rightarrow (2, 1, 5, 3, 4)$, and this is the minimum number, so the $d(\sigma) = 3$. Write a divide and conquer algorithm that, given a permutation $\sigma$, outputs the distance $d(\sigma)$. What is the running time?

7. (***) Given an array of real numbers $A[1..n]$, we define the weight of a subarray $A[i..j]$ for $1 \leq i \leq j \leq n$ to be the sum of the elements contained in this subarray, $weight(i, j) = \sum_{k=i}^{j} A[k]$. The absolute weight is the absolute value of the weight, $|weight(i, j)|$. Notice that the numbers in the array are allowed to be negative, but the absolute weight is always non-negative. Your task is to design (and analyze) a divide-and-conquer algorithm that finds a subarray with the minimum absolute weight. Find a solution that runs in time $o(n^2)$.