Problem Set 8

There's no sense in being precise
when you don't even know what you're talking about.
— *John von Neumann*

- This problem set is due **at 9:00am** on **April 7, 2025** .
- This problem set comprises 3 problems.
- Submit early. Do **not** wait till the last moment.
- Each solution should start on a new page.
- We will give full credit **only** for correct solutions that are described clearly and convincingly.

Problem 8-1. Just in Time [25 points]

Let $T(n)$ be the time complexity of an algorithm to solve a problem of size n as discussed in class. For these problems, assume $T(n)$ is $O(1)$ for any n less than 3. Solve the following recurrence relations for $T(n)$. Please show your work in deriving your solutions.

(a) [7 points] $T(n) = T(n-3) + 2n$

(b) [8 points] $T(n) = 5T(\sqrt[3]{n}) + \log\log n$

(c) [10 points] $T(n) = 3T(n/5) + 2T(n/7) + \Theta(n)$

Problem 8-2. Comparison is the Thief of Joy [15 points]

Consider the following functions. Within each group, sort the functions in asymptotically increasing order, showing strict orderings as necessary. For example, we may sort $n^3, n, 2n$ as $2n = O(n) = o(n^3)$.

(a) [5 points] $\log n$, $\sqrt[3]{n}$, $\log_7 n$, $\log n^3$, $\log\log n^{10}$, $\log^3 n$.

(b) [5 points] $n^{5/3}$, $n \log n^2$, $n^2 \log n$, 2^n, $\log(n!)$.

(c) [5 points] $(\log n)^n$, $(\log n)^{n-5}$, e^n, $n^{10 \log n}$.

Problem 8-3. All are Equal before the Laws of Mathematics [60 points]

Among the sorting techniques we have learned in class, the `mergesort` and `quicksort` algorithms are perhaps the most practical. However, it is worth our time to think more deeply about why breaking a problem into two subproblems is the best option: why not three? or eight? Is it that writing the code becomes too hard, or does one not really gain that much?

We shall explore this by modifying both algorithms to take an additional input: the number of subproblems/parts in each layer of the recursion! Write the following code in Python, and *make sure you comment your code* clearly and in detail so that the graders can see what you're trying to do.

(a) [25 points] A variant of mergesort we will call "$k$mergesort" with the same signature as the version taught in class, with one addition: an integer k, which defines the number of sub-problems. This might look like

```
from typing import List
def kmergesort(A: List[int], k: int) -> List[int]:
    # code
```

I.e., the array is to be broken down into k subarrays of (approximately) equal size, each of which is then sorted individually using k-mergesort, and then these k subarrays are merged into a single array which is returned as output. You may assume that $2 \leq k \leq n$, where n is the size of the array.

(b) [5 points] Calculate the worst-case time complexity of your $k$mergesort technique.

(c) [25 points] A variant of quicksort we will call "$k$quicksort". In a manner similar to the $k$mergesort case above, we shall first select $k-1$ pivot elements (you may pick these in any manner you wish), and partition the array into k subarrays using these pivots. Each subarray is then sorted individually using the same method. You may assume that $2 \leq k \leq n$, where n is the size of the array. *Hint: This problem may be easier to code up than you think!*

(d) [5 points] Calculate the worst-case time complexity of your $k$quicksort technique.