Homework Assignment #4
Due: February 14, 2025 at 5:00 p.m.

The same rules apply as for Assignment 1. (In particular, you can work in pairs, where each pair submits just one paper.)

[4] **1.** In the first lecture, we saw an algorithm that checks whether there is some value that appears more than $n/2$ times in an (unsorted) array $A[1..n]$. That algorithm runs in $O(n)$ time.

In a recent lecture, we saw an algorithm $\textsc{Select}(A[1..n], k)$ that finds the kth smallest element of an (unsorted) array $A[1..n]$ in $O(n)$ time. If A contains duplicates, this algorithm returns the value that would appear in position k of the sorted version of array A.

Now consider the problem of deciding whether some value appears more than $n/5$ times in an (unsorted) array $A[1..n]$. Give a *simple* algorithm to solve this problem in $O(n)$ time. Briefly explain why your algorithm is correct. (You do not have to give a formal proof.)

Hint: Your algorithm should call $\textsc{Select}$ a few times to find a few candidate values that you can then check.

2. Some scientists repeatedly do an experiment and take a measurement. They want to find out the median value of the measurement.

They maintain a hash table to keep track of the measurements. The hash table stores a set of pairs $(val, freq)$, where $freq$ represents the number of times the scientists have seen the measurement val so far. Each time they repeat the experiment, and get a measurement of val, they look up val in the hash map and increment its corresponding $freq$ by 1 (or insert $(val, 1)$ into the hash table if val has never been seen before).

At the end of their experiment they iterate across the hash table and collect all the $(val, freq)$ pairs into two arrays $Val[1..n]$ and $Freq[1..n]$. Here, n is the number of different values that were seen as measurements. The arrays are not sorted, since the hash function scrambles keys.

Let $r = \sum_{i=1}^{n} Freq[i]$ be the total number of repetitions of the experiment. Note that r could be much larger than n. We want to find a median measurement m such that $\sum_{i:Val[i]<m} Freq[i] \le r/2$ and $\sum_{i:Val[i]>m} Freq[i] \le r/2$. (In other words, at most half the repetitions obtained a measurement less than m and at most half obtained a measurement greater than m.) We wish to do this in $O(n)$ time.

[4] **(a)** To solve the problem, we generalize it a little. Given f, we want to design a $\textsc{Find}$ function to find the value v_f such that $\sum_{i:Val[i]<v_f} Freq[i] < f$ and $\sum_{i:Val[i]\le v_f} Freq[i] \ge f$. (Intuitively, if you wrote down all the results of measurements one by one in sorted order, v_f would appear in the fth position.)

Fill in the missing parts of the following algorithm to do this.

```
Find(Val[1..n], Freq[1..n], f)
    Precondition: n ≥ 1 and 1 ≤ f ≤ Σ_{i=1}^{n} Freq[i]
    if n = 1 then return Val[1]
    else
        v_mid ← Select(Val[1..n], ⌊n/2⌋)          ▷ algorithm from text Section 9.3
        create arrays Val_low[1..⌊n/2⌋], Freq_low[1..⌊n/2⌋], Val_high[1..⌈n/2⌉], Freq_high[1..⌈n/2⌉]
        index_low ← 1
        index_high ← 1
        total_low ← 0                              ▷ will store total frequency in low half
        for i ← 1..n do                            ▷ split arrays into two halves by value
            if Val[i] ≤ v_mid then
                Val_low[index_low] ← Val[i]
                Freq_low[index_low] ← Freq[i]
                index_low ← index_low + 1
                total_low ← total_low + Freq[i]
            else
                Val_high[index_high] ← Val[i]
                Freq_high[index_high] ← Freq[i]
                index_high ← index_high + 1
        if ________________________ then
            return Find(________________________)
        else
            return Find(________________________)
```

Give a brief explanation of how you designed your solution.

You do *not* have to prove your algorithm is correct. (However, if you want to check your solution, you can prove it is correct; just don't hand in that proof.)

[2] **(b)** Show that your solution to part (a) runs in $O(n)$ time.

[1] **(c)** How would you use the algorithm in (a) to find the median value m described above?