Please enter your name and uID below.

Name:

uID:

Collaborators, if any, and how you collaborated:

Submission notes

- Due at 11:59 pm on Thursday, September 15.
- 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 this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. 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. (Parameter passing, 25pts) Throughout this class, we assume that parameter passing during method calls takes constant time, even if an N-element array is being passed. This assumption is valid in most systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three parameter-passing strategies.

   Consider the MergeSort algorithm in Figure 1.6 of the book. Let N be the size of the original array, and n be the size of a subarray. For each of the following parameter passing strategies, (i) give a recurrence $T(n)$ for the worst case running time of MergeSort on a subarray of size n, and (ii) derive a tight upper bound on the running time of MergeSort by describing the behavior of the recursion tree (hint: use facts about common cases of level-by-level series).

   Tips: Your final running time should only be in terms of N, but some of your recurrences will use n and N. Feel free to ignore ceilings and floors. It is not required, but helpful, to write the recursion tree depth, how much work is done at the leaves, and the level-by-level sums.

   (a) The array is always passed by pointer in $\Theta(1)$ time.

   (b) The array is always passed by copying the entire array in $\Theta(N)$ time.

   (c) Only the relevant subarray is passed by copying in $\Theta(n)$ time (n is the size of the subarray).

2. (Index fixed point, 25pts) The IndexFixedPoint problem is, given a sorted array of distinct integers $A[1..n]$, return whether there is some i where $A[i] = i$.

   So, $A = [-1, 2, 4]$ should return *true* ($A[2] = 2$), but $A = [-1, 10, 12]$ should return *false*.

   (a) Describe a $O(\log n)$ divide and conquer algorithm to solve IndexFixedPoint. Your description may optionally include pseudocode, but you must give a general written description of the algorithm.[1]

   (b) Your algorithm should *divide* the problem into smaller versions of the same problem (subproblems) and then *conquer* only one of the subproblems. Which problems can your algorithm ignore, and how do you know they're safe to ignore?

   (c) Give a recurrence for the worst case running time of your algorithm, and derive a tight upper bound on the closed form solution of the recurrence.

[1] For good examples of algorithm descriptions, see the first paragraph of the Mergesort, Quicksort, and Quickselect sections in the book.

3. (Weighted median, 20pts) WeightedMedian is a generalization of the median problem if each number also had some importance, or weight, associated with it.

   Suppose we are given a set S of n items, each with a distinct *value* and a positive *weight*. For any element $x \in S$, we define two subsets

   - $S_{<x}$ is the set of elements of S whose value is less than the value of x.
   - $S_{>x}$ is the set of elements of S whose value is more than the value of x.

   For any subset $R \subseteq S$, let $w(R)$ denote the sum of the weights of elements in R. The **weighted median** of S is any element x such that $w(S_{<x}) \leq w(S)/2$ and $w(S_{>x}) \leq w(S)/2$.

   (a) Describe a recursive algorithm to compute a weighted median in $O(n)$ time, using QuickSelect[2] as a subroutine. Your input is two arrays $S[1..n]$ and $W[1..n]$, where for each index i the ith element has value $S[i]$ and weight $W[i]$, and your output should be the index of a weighted median.

   (b) Give a recurrence for the worst case running time of your algorithm, and solve for a good upper bound on its closed form.

   Tips: Finding the median of the values m and then doing some extra work can help you recurse on only a portion of the whole array: figuring out what work to do, and what recursive call to make, is the key to figuring out this algorithm.

   When using Quickselect to find the median, it doesn't necessarily try to find the median *again in the recursive call. Depending on where the pivot is, it actually needs to solve a* different *problem, and that is it has the extra k parameter. For similar reasons, you may find it helpful to include extra parameters in your weighted median algorithm.*

   The PS1 Programming problem asks you to implement this algorithm – see its description for some concrete examples, as well as an alternative explanation of what this problem is asking for.

[2] Assume the $O(n)$ median of medians implementation