Exercise 4: Algorithm Design, Sorting

Due date: August 9, 11:59pm

Instructions:

Submit a typed or neatly handwritten scan of your responses to the "Exercise 4" assignment on Gradescope here: https://www.gradescope.com/courses/52323. Make sure to log in to your Gradescope account using your UW email to access our course.

For more details on how to submit, see
https://courses.cs.washington.edu/courses/cse401/18au/hw/submitting_hw_guide.pdf.

These problems are meant to be done **individually**. If you want to discuss these problems with a partner or group, make sure that you're writing your answers individually later on. Check our course's collaboration policy if you have questions.

1. Sorting algorithms

Answer each of the following in English (not code or pseudocode). **Each subpart requires at most 2-4 sentences to answer.** Answers significantly longer than required <u>will not receive full credit</u>.

(a) Recall that merge sort works by taking an array, splitting it into two pieces, recursively sorting both pieces, then combining both pieces in $\mathcal{O}(n)$ time. Suppose we split the array into *three* equally sized pieces instead of two. And after we finish recursively sorting each of the three pieces, we first merge two of the three pieces together, then we merge that with the third piece to get the final sorted result.

(i) Write the recurrence for this variation of merge sort. Do not worry about finding the exact constants for the non-recursive term (for example if the running time is A(n) = 4A(n/3) + 25n, you need to get the 4 and the 3 right but you don't have to worry about getting the 25 right).

(ii) Use the master theorem to give an asymptotic bound for this variation of merge sort. Show your work.

(iii) Do you think this variation of merge sort is better or worse than the two-split merge sort? If necessary, explain which you expect to have better constant factors and why.

(b) You're given an array where each element is an (age, name) pair representing guests at a DC-Marvel Universe party in Shire. You have been asked to print each guest at the party in ascending order of their ages, but if more than one guests have the same age, only the one who appears first in the array should be printed. For example, if the input array is

    [(23, Noah), (2000, Gandalf), (50, Frodo), (47, Emma), (23, Sophia), (83200, Superman), (23, Alice), (47, Madison), (47, Mike), (150, Dumbledore)]

the output should be

    (23, Noah) (47, Emma) (50, Frodo) (150, Dumbledore) (2000, Gandalf) (83200, Superman)

Describe a solution that takes $\mathcal{O}(1)$ space in addition to the provided input array. Your algorithm may modify the input array. This party has some very old guests and your solution should still work correctly for parties that have even older guests, so your algorithm can't assume the maximum age of the partygoers. Give the worst-case tight-$\mathcal{O}$ time complexity of your solution.

2. Sorting Design Decisions

For each of the following scenarios, choose the best sorting algorithm from the list below. Justify your choice in 2 - 3 sentences and be sure to explain how your sort addresses the problems of particular prompt.

insertion sort, selection sort, heap sort, merge sort, quick sort

(a) You are a game programmer in the 1980s who is working on displaying a sorted list of enemy names that a player has encountered during their gameplay. Since it is a game, you want to display the names of the enemies fast as possible, but because it is the 1980s, your customers are used to and will be okay with occasional slow loading times. Additionally, the game is intended to run normally on consoles that don't have much memory available.

(b) Imagine that you are sorting a **small set** of computer files by their file name. You realize, however, that each computer file is huge and takes up a lot of disk space, so you do not want to copy excessively when sorting. In fact, even just moving and rearranging these large files is expensive, so you don't want to move them often. Hint: You may find it useful to refer to visuals of the sorting algorithms. Lectures slides and `https://visualgo.net/en/sorting` are good resources for remembering these general ideas.

(c) Imagine that you are a future NASA software engineer. You're assigned a task to sort data you receive from a probe on Mars, in which each piece of data includes time and temperature. The sensors on this probe capture very large amounts of data. The data is already given to you in sorted order of earliest to latest time, but you want to sort them by temperature, where ties in temperature are then sorted by time.

3. Analyzing topKSort

In this problem, we'll think about two possible ways to use a heap to implement `top-`k`-sort`. For simplicity, we'll assume that our input is an array of `int`s. The clients for your code do not want you to change the array they give you – you are allowed to copy elements out of it, but not to alter the input array in any way (your `top-`k`-sort` from the programming project also satisfies this requirement).

Algorithm 1 is the top-k sort algorithm you implement in Project 3. Recall that this algorithm runs in $\Theta(n \log k)$ time in the worst case.

Note: you will need to have a general idea of how you implemented this in Project 3 for part (c) below, but should be able to answer parts (a) and (b) independently.

Algorithm 2 is described below. `buildMaxHeap` works like the buildHeap from lecture, except that a max-heap is built instead of a min-heap. Note that **Algorithm 2** runs in time $\Theta(n + k \log n)$ in the worst case.

```
int[] algorithm2(int[] input, int k) {
    MaxHeap h = buildMaxHeap(input); // copies input into a new array.
    int[] output = new int[k];
    for(int i = k - 1; i >= 0; i = i - 1) {
        output[i] = h.removeMax();
    }
    return output;
}
```

The analysis of some of the running times will be easier knowing this fact: Here's a useful fact about how logs and polynomials grow:

Theorem 1. *If α and β are positive constants then n^α dominates $(\log(n))^\beta$.*

(a) Suppose the relationship between n and k is: $k = \log(n)$. What would be the simplified big-Θ running times of **Algorithm 1** and **Algorithm 2** be under this relationship. Your answers should **only** include n (not k).

(b) Suppose the relationship between n and k is: $k = n/\log(n)$. What would the simplified big-Θ running times of **Algorithm 1** and **Algorithm 2** be under this relationship. Your answers should **only** include n (not k). Hint: you may want to use a log identity.

(c) The big-Θ running time for **Algorithm 1** is never better than the one for **Algorithm 2**. Despite that, there are reasons to choose **Algorithm 1** over **Algorithm 2** — describe one reason why you might choose **Algorithm 1**.

4. Algorithm Design

Suppose you are designing a search aggregator, which for a given query fetches search results from two different search engines and presents an intersection of the two search results. Here is a simplified version of this problem: Given two sorted integer arrays of lengths m and n, return a new array with elements that are present in both input arrays. For example, if the input arrays are [17, 23, 35, 43, 47, 69, 78, 80, 84, 86] and [23, 35, 50], the output array should be [23, 35].

Assume that each input array **will not** contain duplicates within itself. For example [17, 17] and [1, 2] would not be a valid input because one of the arrays has duplicates in it.

(a) Describe a brute force solution. What is the worst-case tight big-$\mathcal{O}$ time complexity of this brute solution?

(b) Describe a solution that has worst-case tight-$\mathcal{O}(m \log n)$ time complexity.

(c) Describe a solution that has worst-case tight-$\mathcal{O}(m + n)$ time complexity.

(d) Between subparts (b) and (c) above, is one solution always better than the other (in terms of runtime)? If so, which one and why? If not, describe the situation (in terms of m and n) when one is better than the other.