CPSC 320 2017W2: Quiz 1

January 12, 2018

1 Looping Back to Asymptotic Analysis

Each row of the table below lists a problem posed for an array of n numbers. You are determining **good big-O bounds for the worst-case performance** of efficient algorithms for each problem. In the **left blank**, give a bound for an efficient algorithm if the **input array is not known to be sorted**. In the **right blank**, give a bound for an efficient algorithm if the **input array is known to be already sorted**.

Each bound is one of: $O(1), O(\lg n), O(n), O(n \lg n), O(n^2)$.

Note: throughout this problem, assume basic operations on numbers take constant time.

The problem is to find...	big-O bound (unordered)	big-O bound (known to be sorted)
... a given target value		
... the average value		
... a pair of values summing to a given target value		

For the individual portion only, finish these additional table entries:

The problem is to find...	big-O bound (unordered)	big-O bound (known to be sorted)
...the three smallest values		
...the number of values that are larger than the average value		
...whether any value is repeated (appears more than once)		

Finally, note that for some algorithms over arrays, even if the input is not **known** to be sorted, sorted arrays are a common case worth optimizing for.

For example, if the problem were to determine if any array contains any positive numbers, then we can start a linear scan for positive numbers from the right, returning `YES` as soon as we find a positive or `NO` when we finish the scan. On any array that happens to be sorted and that happens to have a positive number, this algorithm runs in constant time. In the worst case, it still takes linear time.

A friend suggests that sorted arrays are a case worth optimizing for when finding the three smallest values in an array of n numbers. Is there a correct and efficient algorithm for this problem that has an asymptotically better runtime in the case where the input *happens to be sorted* (but is not **known** beforehand to be sorted)?

Fill in the circle beside the *best* answer:

◯ Yes

◯ No

2 All Tied Up

SMP, as discussed in class, assumes that every woman and every man has a fully ordered preference list. In this problem (except the last question of the individual part), we consider the situation where a woman or man may have ties in their ranking. For instance, woman w_1 might have a preference list like $m_3, m_1 = m_4, m_2$, meaning she likes man m_3 best, followed by m_1 and m_4 in no particular order (that is, she does not prefer m_1 to m_4, nor m_4 to m_1), followed by m_2. In this case, we say that w_1 is *indifferent* between m_1 and m_4. It is possible for a woman or a man to be indifferent between more than two people and between multiple sets of people.

We will call this problem STP (the SMP with Ties Problem).

1. A *strong instability* in a perfect matching consists of a woman w and a man m such that w and m both (strictly) prefer each other to their current partner.

   Now, fill in the circle to indicate whether the following statement is **always** true (true for *every* situation matching the scenario), **sometimes** true (true for at least one situation matching the scenario but also false for at least one such situation), or **never** true.

   Scenario: I is an instance of STP.

   Statement: I has a valid solution (perfect matching) with no strong instability.

   ◯ Always

   ◯ Sometimes

   ◯ Never

2. What is the *largest* number of solutions without strong instabilities that any STP instance with n women can have? ☐

3. Continuing with the always/sometimes/never problem type above, fill in the circle next to the best answer for the following scenario and statement.

   Scenario: I is an instance of STP in which every woman is indifferent between m_1 and m_2. m_1 ranks w_1 first (tied with no one) while m_2 ranks w_1 last (tied with no one). P is a solution to I with no strong instabilities.

   Statement: m_2 marries w_1 in P.

   ◯ Always
   ◯ Sometimes
   ◯ Never

4. A *weak instability* in STP is one of:

   - a strong instability (i.e., every strong instability is also a weak instability),
   - a woman w and man m such that w prefers m to her partner and m is indifferent between w and his partner, or
   - a man m and woman w such that m prefers w to his partner and w is indifferent between m and her partner.

   Continuing with the always/sometimes/never problem type above, fill in the circle next to the best answer for the following scenario and statement.

   Scenario: I is an instance of STP.

   Statement: I has a valid solution (perfect matching) with no weak instability.

   ◯ Always
   ◯ Sometimes
   ◯ Never

5. **RETURNING TO THE STANDARD VERSION OF SMP** (with no ties in preference lists): Imagine we decide to resolve the "unfairness" of the Gale-Shapley algorithm (which gives optimal results to the proposing side and pessimal results to the other side) by alternating proposals between men and women:

```
procedure FAIR-AND-BALANCED-MARRIAGE(M, W)
    initialize all men in M and women in W to unengaged
    while an unengaged man with at least one woman on his preference list remains do
        let a woman make the first proposal, and after that
          let the opposite gender (of the last proposer) make the next proposal
        choose an engaged person of the proposing gender p
        propose to the next person p′ on the preference list of p
        if p′ is unengaged then
            engage p to p′
        else if p′ prefers p to their fiancee then
            break engagement of p′ to their fiancee
            engage p to p′
        end if
        cross p′ off p's preference list
    end while
    report the set of engaged pairs as the final matching
end procedure
```

   Continuing with the always/sometimes/never problem type above, fill in the circle next to the best answer for the following scenario and statement.

   Scenario: P is a solution produced by this "fair and balanced" marriage algorithm on an instance of SMP.

   Statement: P is stable.

   ◯ Always
   ◯ Sometimes
   ◯ Never

3 To Re Mi Pa So Ti La Do!

Anytime that we're exploring any solution space for a problem involving many different people with their own desires and preferences, we can define "strong Pareto optimality" as a useful criterion, often just called "Pareto optimality".

First, we need to define a "single step" from one solution to another. For example, given a perfect matching (valid but not necessarily stable solution) for an SMP instance, a single step might be to take any two married pairs (m, w) and (m', w') and swap them to get the two married pairs (m, w') and (m', w) (and a new perfect matching).

Then, a solution is "Pareto optimal" if no single step leaves everyone at least as well off as they were and at least one person better off than they were. (In other words, no single step would get at least some people supporting it and no one opposing it.) In our SMP example, only m, m', w, and w' could have their satisfaction change; so, we're asking if none of them gets a worse partner than before and at least one gets a better partner.

We'll explore this SMP definition of Pareto optimality in this problem.

1. Write in the subscripts (numbers) on the men and women in the list of matchings below to form a valid but unstable solution to the given SMP instance that **is** Pareto optimal.

w_1: 1 2	m_1: 1 2
w_2: 2 1	m_2: 1 2

   Write subscripts in the blanks on this solution: $(w\_\_, m\_\_), (w\_\_, m\_\_)$.

2. If we take a "single step" from this solution, which person gets a **worse** partner? Fill in the blank next to the answer.
   - ◯ w_1
   - ◯ w_2
   - ◯ m_1
   - ◯ m_2

3. Briefly explain why avoiding instability is probably a better metric to judge the quality of an SMP solution than Pareto optimality. (I.e., why might a Pareto optimal but not stable solution be a problem?)

4. Fill in the circle to indicate whether the following statement is **always** true (true for *every* situation matching the scenario), **sometimes** true (true for at least one situation matching the scenario but also false for at least one such situation), or **never** true.

   Scenario: P is a **stable** solution for an SMP instance with $n \geq 3$.

   Statement: P is Pareto optimal (for that instance).

   ◯ Always
   ◯ Sometimes
   ◯ Never

5. "Weak" Pareto optimality is like "strong" Pareto optimality defined above except that a single step that improves the solution has to make **everyone** better off (rather than making at least someone better off while making nobody worse off).

   In our SMP application of Pareto optimality, which of the following will always be a "weak" optimum? Fill in the square next to **every** correct answer.

   ☐ An arbitrary valid solution to any SMP instance.
   ☐ An arbitrary valid solution to any SMP instance with $n \geq 3$.
   ☐ An arbitrary stable solution to any SMP instance.
   ☐ An arbitrary stable solution to any SMP instance with $n \geq 3$.
   ☐ An arbitrary *strong* Pareto optimal solution to any SMP instance.
   ☐ None of these is guaranteed to be a weak optimum.

4 Knowing Your Structures

Each item below describes an operation on a tree data structure. Choose the tightest worst-case running time for a good implementation of the operation among: $O(1), O(\lg n), O(n), O(n \lg n), O(n^2)$. n represents the number of items (keys or key/data pairs) in the structure. Note: BST is binary search tree.

1. Find the successor of a node (the node with the next largest key) in an AVL tree.

   Worst-case bound:
   ◯ $O(1)$
   ◯ $O(\lg n)$
   ◯ $O(n)$
   ◯ $O(n \lg n)$
   ◯ $O(n^2)$

2. Delete a key from a (not necessarily balanced) binary search tree.

   Worst-case bound:
   ◯ $O(1)$
   ◯ $O(\lg n)$
   ◯ $O(n)$
   ◯ $O(n \lg n)$
   ◯ $O(n^2)$

3. Given two keys k_1, k_2, count the number of keys x such that $k_1 < x < k_2$ in a balanced BST.

   Worst-case bound:
   ◯ $O(1)$
   ◯ $O(\lg n)$
   ◯ $O(n)$
   ◯ $O(n \lg n)$
   ◯ $O(n^2)$

4. Determine if a binary tree that claims to be a heap satisfies the heap property.

   Worst-case bound:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

5. Return an array that contains the first 150 keys in a B+ tree (i.e., the 150 smallest keys).

   Worst-case bound:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

6. Build a maxHeap from a given array of integers.

   Worst-case bound:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

7. Sort the elements of an array in place using the standard quicksort algorithm.

   Worst-case bound:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

5 Choosing Your Structures

For each of the following, choose the data structure that most efficiently supports a solution of the options: **array**, **stack**, **queue**, **priority queue** (implemented as a binary heap), **balanced BST** (balanced binary search tree implemented as an AVL tree) and **dictionary** (dictionary/map implemented as a hash table). Choose the **best** answer in each case. If there are multiple best answers, just pick one.

1. Determine the next collision that will happen in a game where balls are bouncing around on a pool table. You may assume that a function to determine when two balls will collide (assuming they do not change direction beforehand) has already been written.

   Best option:
   - ◯ **array** ◯ **priority queue**
   - ◯ **stack** ◯ **balanced BST**
   - ◯ **queue** ◯ **dictionary**

2. Given a map of a cave (represented as a graph), a starting location (a vertex), and a magic spell that will affect all locations within a given distance from the starting location, determine which locations will be affected by the spell. Note that spells do not penetrate walls.

   Best option:
   - ◯ **array** ◯ **priority queue**
   - ◯ **stack** ◯ **balanced BST**
   - ◯ **queue** ◯ **dictionary**

3. Given the same cave map inputs as above, and a set of exits (vertices), find the path containing the least number of dangerous creatures from the start to an exit.

   Best option:
   ◯ **array** ◯ **priority queue**
   ◯ **stack** ◯ **balanced BST**
   ◯ **queue** ◯ **dictionary**

4. You are building an interpreter for a version of the C language that does not have pointers, and you need to keep track (by name) of the current value of each global and local variable.

   Best option:
   ◯ **array** ◯ **priority queue**
   ◯ **stack** ◯ **balanced BST**
   ◯ **queue** ◯ **dictionary**

5. You are a teaching assistant who is maintaining the current projected final grades of the students in the course. The projections are updated every time a new assigment, quiz or exam grades becomes known (whether a single updated or a group of simultaneous ones), and you want to be able to return efficiently a list of all students whose projected final grade lies within a given range (this range is not fixed, but varies with every query).

   Best option: ◯ **array** ◯ **priority queue** ◯ **stack** ◯ **balanced BST** ◯ **queue** ◯ **dictionary**

6. Given a string containing only lowercase alphabetic characters (a–z), determine which character occurs the most often in the string.

   Best option: ◯ **array** ◯ **priority queue** ◯ **stack** ◯ **balanced BST** ◯ **queue** ◯ **dictionary**

7. Given a mathematical expression, determine if it is parenthesized properly. For example, "(((2+3)*5))" is parenthesized properly, but "(2+4)) + (5" is not.

   Best option: ◯ **array** ◯ **priority queue** ◯ **stack** ◯ **balanced BST** ◯ **queue** ◯ **dictionary**

6 Tangling the Knot

A large organization has decided to ensure that each person in the organization has exactly one mentor and one mentee.[1] Both mentors and mentees are drawn from the same set, the employees. To make this work best, they decide to have each person rank each other person (first place, second place, etc.) as their choice of mentor and, separately, as their choice of mentee.

Now, they want to match mentor/mentee pairs up so that each person has exactly one mentor and each person has exactly one mentee. We call this problem MMP (the mentor/mentee problem). An instance of MMP may never have fewer than two employees.

1. Solve the following instance of MMP so that it is never the case that there are two employees a and b such that a would rather have b as their mentee than the mentee a was assigned, and b would rather have a as mentor than the mentor b was assigned.

Employee "name"	Ranked mentees	Ranked mentors
e_1	e_2, e_3	e_3, e_2
e_2	e_1, e_3	e_1, e_3
e_3	e_1, e_2	e_2, e_1

   Write subscripts in the blanks to fill in the solution. We've already given subscripts on the mentors (the first element of each pair): $(e_1, e\_\_), (e_2, e\_\_), (e_3, e\_\_)$.

2. A friend proposes solving MMP via the following reduction, assuming each employee has a unique employee ID:

   > Create an SMP instance as follows: Let the set of men M be the first half of the employees by ID. Let the set of women W be the second half of the employees by ID. Let the preference

[1] This is their Mentor/Mentee Organization Reneweal ProGram (or MMORPG) and they spend a lot of time playing around with it.

lists of men match the first half of employees' mentee preferences over the second half of employees. Let the preference lists of women match the second half of employees' mentor preferences over the first half of employees.

Given a solution to the SMP instance, create the solution to MMP as follows: For a pair (m, w), let the "first-half" employee corresponding to m be a mentor to the "second-half" employee corresponding to w (the mentee).

Fill in the blank next to each of the following critiques of this reduction that is accurate:

- ☐ For some instance of MMP, this does not produce a valid instance of SMP.
- ☐ This can produce an MMP solution with instabilities caused by instabilities in the SMP solution.
- ☐ This can produce an MMP solution in which some employee has no mentor.
- ☐ This can produce an MMP solution in which some employee has more than one mentor.
- ☐ None of these is accurate.

3. Fill in the circle to indicate whether the following statement is **always** true (true for *every* situation matching the scenario), **sometimes** true (true for at least one situation matching the scenario but also false for at least one such situation), or **never** true.

   Scenario: P is a valid solution to an instance of MMP.

   Statement: There is a path going from mentor to mentee in P (i.e., from a person to the person who is their mentee and then optionally continuing from that person to their mentee and so on) that is a cycle.

   ◯ Always
   ◯ Sometimes
   ◯ Never

4. A friend proposes solving MMP via this alternate reduction:

   > Create an SMP instance as follows: Let the set of men M be the set of employees. Let the set of women W be a second copy of the set of employees. Let the preference lists of men match the employees' **mentee** preferences. Let the preference lists of women match the employees' **mentor** preferences.
   >
   > Given a solution to the SMP instance, create the solution to MMP as follows: For a pair (m, w), let the employee corresponding to m be a mentor to the employee corresponding to w (the mentee).

   Which of the following should be addressed to complete this reduction? Fill in the blank next to each correct answer.

   ☐ Each of the set of men and the set of women is incomplete.
   ☐ The preference lists (of each of the men and the women) are incomplete.
   ☐ Two people may end up with the same mentor.
   ☐ A person may end up as their own mentor.
   ☐ None of these is accurate.