CPSC 320 2016W1: Midterm #1 (Group Version)

2016-10-20 Thu

1 O'd to a Pair of Runtimes [10 marks]

Consider the following pairs of functions (representing algorithm runtimes), expressed in terms of the positive variables n and occasionally k, where $k < n$. For each pair, write between them the **best** choice of:

- **LEFT** to indicate that the left one is big-O of the right one
- **RIGHT** to indicate that the right one is big-O of the left one
- **SAME** to indicate that the two are Θ of each other, or
- **NONE** to indicate that none of the previous relationships holds

Notes: You are choosing the "faster" one. Do not write **LEFT** or **RIGHT** if **SAME** is true. The first one is filled in for you.

[1 mark per problem]

Left Function	**LEFT**, **RIGHT**, **SAME**, or **NONE**	Right Function
n	**LEFT**	n^2
$\sqrt{n^2}$		$\sqrt{n}$
$n^{1.01}$		$n^{1.99}$
$\lg(n * (n-1))$		$(\lg n)^2$
$n + \lg n$		$n - \lg n$
$\log_{20}(n^2)$		$\log_2(n^{20})$
$n^6 + n^2$		$3n^{5+2}$
$n!$		2^n
$n \lg n$		$\frac{n^2}{\lg n}$
$n \lg k$		$k \lg n$
$n + k$		n

2 The DnC: Trumped-Up Values [10 marks]

An array A of integers of length n is of the form $[1, 2, \ldots, k-1, k+j, k+j+1, \ldots, n+j]$, where $1 \leq k \leq n$ and j is positive. That is, it is the integers 1 through n in order, except that at some point all the remaining values (those with indexes k and up) increase by j. So, it might look like $[1, 2, 3, 4, 7, 8]$ for $n = 6$, $k = 5$, and $j = 2$.

1. Give an efficient algorithm to determine j—given A and n—and a good asymptotic bound on its runtime. **[3 marks]**

2. Give a good brute force algorithm to determine k—given A and n—and its runtime. **[3 marks]**

3. Complete the following pseudocode algorithm so that it efficiently (in $O(\lg n)$) determines k, given A and n. **[4 marks]**

```
FindK(A, n):
  return FindK(A, 1, n) // assumes 1-based indexing

FindK(A, left, right):

  if left > right:

      return ______________________________

  else:

      mid = floor((left + right) / 2)


      if _________________________________________:


          return FindK(A, ________________, _________________)

      else:

          return FindK(A, ________________, _________________)
```

3 Misanthropic Marriage [6 marks]

Imagine the stable marriage problem solved using the original Gale-Shapley algorithm (below), but while all men have ranked all women, **none of the women rank any of the men until the matching process begins**.

The women only discover, for any pair of men, which one they prefer after they've gone on a date with each man, and they leave *that* experience to as late as possible. (Note: one date with a man is enough for a woman to compare him with any number of other men with whom she's also had a date.) We say they "discover" rather than "decide" because we assume each woman effectively still has a hidden, complete ranking of the men. She and the algorithm simply don't know what it is until she has dated each man.

1. Complete the following example with two men and women so that the women never date anyone at all. (We filled in the first man's preference list with woman 1 ranked 1*st* and woman 2 ranked 2*nd*.) **[2 marks]**

Ordered list of women	Man	Woman	(Hidden) ordered list of men
1, 2	1	1	
	2	2	

2. Circle each mention of a man in the algorithm below where woman w must have dated that man by the time the line involving the mention of him completes. **[2 marks]**

```
1: **procedure** STABLE-MARRIAGE($M$, $W$)
2:     initialize all men in $M$ and women in $W$ to unengaged
3:     **while** an unengaged man with at least one woman on his preference list remains **do**
4:         choose such a man $m \in M$
5:         propose to the next woman $w \in W$ on his preference list
6:         **if** $w$ is unengaged **then**
7:             engage $m$ to $w$
8:         **else if** $w$ prefers $m$ to her fiancé $m'$ **then**
9:             break engagement of $m'$ to $w$
10:            engage $m$ to $w$
11:        **end if**
12:        cross $w$ off $m$'s preference list
13:    **end while**
14:    report the set of engaged pairs as the final matching
15: **end procedure**
```

3. Imagine we also left the men's choices until after they date—at the last possible moment—the women they must choose between. There are $|M| = |W| = n$ men and women. How many women must a man date before he makes his first proposal? **[2 marks]**

4 eXtreme True And/Or False [9 marks]

Each of the following statements may be **always** true, **sometimes** true, or **never** true. Select the best of these three choices and then:

- If the statement is **always** true, (1) give and very briefly explain an example instance in which it is true and (2) sketch the key points of a proof that it is always true.
- If the statement is **never** true, (1) give and very briefly explain an example instance in which it is false and (2) sketch the key points of a proof that it is never true.
- If the statement is **sometimes** true, (1) give and very briefly explain an example instance in which it is true and (2) give and very briefly explain an example instance in which it is false.

Note that you will always select a choice and then give two answers. There is space for this below.
[3 marks per part]

1. After partitioning a weighted, undirected graph into two (non-empty) subsets of nodes, two edges in a minimum spanning tree cross from one of the subsets to the other.

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

2. In a SMP problem (with $n > 1$) in which every man prefers woman w_i to woman w_j, w_j marries her first choice.

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

3. An articulation point in a simple, unweighted, undirected graph is diametric. (Recall that if the shortest path between a vertex i and another j is equal to the graph's diameter, we will say of i that it is a "diametric" vertex.)

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

THE REST OF THIS PAGE IS INTENTIONALLY BLANK.
USE IT FOR SCRATCH/EXTRA SPACE IF NEEDED.

5 Olympic Scheduling Revisited: Value Neutral [9 marks]

We revisit the Olympic Scheduling problem, but removing a **different** assumption from our quiz.

RECALL the Olympic Scheduling problem: You are in charge of a live-streaming YouTube channel for the Olympics that promises never to interrupt an event. (So, once you start playing an event, you must play only that event from the time it starts to the time it finishes.) You have a list of the events, where each event includes its: **start time, finish time (which must be after its start time), and expected audience value**. Your goal is to **make a schedule to broadcast the most valuable complete events**. **The best schedule is the one with the highest-valued event**; **in case of ties, compare second-highest valued events, and so on**. (So, for example, you obviously **will** include the single highest-valued event in the Olympics—presumably the hockey gold medal game—no matter what else it blocks you from showing.)

(Times when you're not broadcasting events will be filled with "human interest stories" that have zero value; so, they're irrelevant.)

UPDATE: Previously, you went on to assume that all start times and finish times were distinct, and all event values were distinct. Now, you make **NONE** of these assumptions.

NOTE: If event j starts at the finish time of event i (i.e., $s_j = f_i$), they do **not** overlap, and can both be broadcast.

5.1 An In-Valuable Assumption [5 marks]

Recall the old naïve algorithm for Olympic Scheduling.

1. Begin with an empty result set and a list of all the events.
2. Until there are no events left in the list:
   (a) add to the result set the highest valued event in the list, and
   (b) delete that event and all events that conflict with it from the list.

A friend claims that this algorithm will still work if we simply add a tie-breaker: "add to the result set the highest valued event in the list, breaking ties by earliest start time and then by earliest finish time (and arbitrarily if value, start, and finish are all tied)".

The resulting algorithm is **not** optimal in general.

1. Draw a small instance for which this algorithm gives a suboptimal result.

2. State what solution this algorithm produces on this instance and briefly explain why.

3. Briefly explain why this is not an optimal solution to this instance, including giving the optimal solution.

5.2 An ACTUALLY Unrelated Reduction Problem [4 marks]

NOTE: This problem can be solved entirely independently of the previous one.

Give a good, correct reduction **from** the Interval Scheduling Problem **to** this new Olympic Scheduling problem. (So, assume you can call on a solution to the new Olympic Scheduling problem, and use it to solve the Interval Scheduling Problem.)

For our purposes: An instance of the Interval Scheduling Problem is n intervals—each with a **start time** and **positive duration**. Intervals are unweighted (i.e., all have the same value). A solution is **the size** of the largest possible set of **non-conflicting** intervals.

Hint: Remember that in the new Olympic Scheduling problem, (1) values (and times) no longer need to be distinct and (2) a solution with **some** event scheduled is better than an empty solution.

**THE REST OF THIS PAGE IS INTENTIONALLY BLANK.
USE IT FOR SCRATCH/EXTRA SPACE IF NEEDED.**

6 Many Blocks That Used to Be a Log [6 marks]

RECALL the web server log problem: Logs from a web server include one line per access to the system (ordered by time of access) with a user ID (a string) on each line (plus other fields we don't care about). Unusual accesses may suggest security concerns. In this problem we are identifying the first user ID that only ever accessed the system once.

We will use n—the total number of entries in the log—to describe problem size. Note: **assume** that comparing two user IDs (strings) for equality or order takes constant time.

NOW: Consider the following algorithm that attempts to solve the web server log problem by sorting the log (preserving the original index position of each entry) and then scanning the sorted array for a runs of a single user ID while tracking the one that originally appeared earliest:

```
if the logs are empty then
    return None
end if
A ← an array $1 \ldots n$, where A[i] is the pair (user ID from log entry $i$, index $i$)
sort A with an efficient sorting algorithm, comparing by ID and breaking ties by index
BestIndex ← None
LastID ← None                                   ▷ Assume None does not compare equal to any ID
for $i \leftarrow 2 \ldots n$ do
    if A[i]'s ID ≠ A[i-1]'s ID and A[i-1]'s ID ≠ LastID then
        if BestIndex = None or A[i-1]'s index < BestIndex then
            BestIndex ← A[i-1]'s index
        end if
    end if
    LastID ← A[i-1]'s ID
end for
return the ID in the log entry at BestIndex
```

Now give and **briefly** justify—including annotating the algorithm above—a reasonable asymptotic bound on the algorithm's worst-case runtime in terms of n.

7 UPDATED BONUS: From the Cutting-Room Floor [3 BONUS marks]

THERE'S ONE NEW, TRIVIAL PROBLEM AT THE END WORTH 1 BONUS MARK!

Except for the last problem, this is a section filled with problems that are **too hard for the amount of points they're worth**. Each is worth 0.5 bonus points. Your total earned bonus points—on this group stage of the midterm exam and toward the course's bonus point reward program for each group member—is your total score here, **rounded down**. We will be ridiculously harsh marking these. **Don't waste your time on anything but the last problem!**

DO THE LAST PROBLEM. Maybe. . . skip the rest.

1. Compare $(2n-2)!$ and $2^{n \lg n}$ asymptotically, simplifying each as much as possible **but not more**, stating which—if either—dominates the other, and sketching the key pieces of a proof of your answer. (Hard-ish.)

2. *Follow the eXtreme True And/Or False rules on:* In a SMP instance with $n > 1$ solved using the original Gale-Shapley algorithm, two men both receive their last choice of woman. (Hard-ish.)

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

3. Continuing the problem from Many Blocks That Used to Be a Log: The algorithm given is **incorrect**. Give the *smallest* example on which the algorithm fails, explain what solution the algorithm produces and why, and give the correct solution. Then, provide a fixed version of the algorithm. (Harder-ish.)

4. Give a good, clear, polynomial-time solution to Olympic Scheduling Revisited: Value Neutral. Note: we are not looking for a reduction, but a complete solution. (Hardest-ish.)

5. Draw a picture that relates some of the best elements of CPSC 320 to each other (algorithms, analysis, problem-solving, wolves, Squirrel Girl, . . .). **Any effort** will receive credit. **Don't leave me blank!** Worth **[1 BONUS mark]**, unlike the other parts of this problem. (Not-so-hard-ish.)

(We might even give a second course bonus point—not exam bonus point—to awe-inspiring creations.)