CPSC 320 2016W1: Midterm #1 Sample Solution

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]

SOLUTION: As bullet points so it's easier to discuss:

$\sqrt{n^2}$ **and** $\sqrt{n}$ Note that $\sqrt{n^2} = n$, which is a higher-order polynomial than $\sqrt{n} = n^{0.5}$. So, the **RIGHT** is in O of the left.

$n^{1.01}$ **and** $n^{1.99}$ By similar reasoning as above, the **LEFT** is in O of the right.

$\lg(n * (n-1))$ **and** $(\lg n)^2$ Note that $\lg(n*(n-1)) = \lg(n^2 - n) \leq \lg(n^2) = 2 \lg n \in O(\lg n)$. So, the left is upper-bounded by $\lg n$, while the right dominates that asymptotically (grows faster). So, the **LEFT** is in O of the right.

$n + \lg n$ **and** $n - \lg n$ The low-order terms on both of these drop out, leaving just $\Theta(n)$. These have the **SAME** asymptotic complexity.

$\log_{20}(n^2)$ **and** $\log_2(n^{20})$ Both sides can be similarly simplified, e.g.: $\log_{20}(n^2) = 2 \log_{20} n = \frac{2 \lg n}{\lg 20} \in O(\lg n)$. So, these have the **SAME** asymptotic complexity.

$n^6 + n^2$ **and** $3n^{5+2}$ The right function is just $3n^7$, which is a higher-order polynomial than the left. So, the **LEFT** is in O of the right.

$n!$ **and** 2^n We've seen this before; $n!$ dominates 2^n; the answer is that the **RIGHT** is in O of the left. If you're wondering why, consider $\frac{n!}{2^n}$. That's $n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$ and $2 \cdot 2 \cdot \ldots \cdot 2 \cdot 2$, both with the same number of terms. Except for the last term (which we can treat as a constant factor or we can balance out by "borrowing" from one of the earlier terms, e.g., noting that $4 = 2 \cdot 2$), each matching term on the top is at least as large as the term on the bottom.

$n \lg n$ **and** $\frac{n^2}{\lg n}$ As we've discussed, the $\lg n$ factors are dominated by even a small polynomial. So, the **LEFT** is in O of the right. You can also see this by simplifying their ratio: $\frac{n \lg n}{\frac{n^2}{\lg n}} = \frac{n(\lg n)^2}{n^2} = \frac{(\lg n)^2}{n}$.

So, these compare the same as $(\lg n)^2$ and n. (At that point, you can take derivatives to convince yourself further. If you do, use ln rather than lg to make your life easier!)

$n \lg k$ **and** $k \lg n$ This is a thorny one. Note that in the case when $k \in \Theta(n)$ (e.g., $k = \lfloor n/2 \rfloor$), these are clearly Θ of each other, but when $k = 2$ (or more appropriately for an asymptotic analysis, when $k << n$), the right is dominated by the left. ($k = 1$ is a bizarre case where the left one has supposed runtime of 0. We can safely ignore this, since algorithm runtimes don't come out to zero. This must be an artifact of the simplified runtime model we get when choosing these functions.)

So, if either one is in O of the other, it'll be the right being in O of the left. Otherwise, the answer will be neither is in O of the other.

Let's compare these in more detail using the form of the definition of O. Is $k \lg n \leq c \cdot n \lg k$ for some c, given that $k \leq n$? I'll set $c = 1$ and—since k and n are both positive—divide out the lg terms from both sides to get the question of whether $\frac{k}{\lg k} \leq \frac{n}{\lg n}$. These are the same function $f(x) = \frac{x}{\lg x}$ applied to different variables, asking is $f(k) \leq f(n)$? Since $k \leq n$, I'm really asking whether f is an increasing (technically, non-decreasing) function. Eyeballing the function, it seems to be. We can also check whether its derivative is non-negative. To keep things easier, I'll use $g(x) = \frac{x}{\ln x}$ instead. (Or, use $\ln x = \frac{\lg x}{\lg e}$.) I won't go through the steps of that derivative, but we get: $f'(x) = \frac{\ln x - 1}{(\ln x)^2}$, which is indeed positive for sufficiently large x (once $\ln x > 1$).

Long story very short: The **RIGHT** is in O of the left.

$n + k$ **and** n This one is **much** easier than the last one. We know from the problem statement that $0 < k < n$. Let's use that to investigate $n + k$. We can add n to all parts of the inequality: $n < n + k < 2n$. In other words, $n + k$ is bounded above and below by multiples of n.

These have the **SAME** asymptotic complexity.

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]**

   SOLUTION: Note that we're guaranteed that the last element has been increased by j (because $k \leq n$). So, $A[n] = n + j$. Solving for j: $j = A[n] - n$.

   In other words, we can find j in constant time by returning $A[n] - n$.

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

   SOLUTION: We'll try looking through all possible values of k from 1 to n. Then, k should be the index of the first element $A[i]$ of A that is equal to $i + j$ (which is greater than i, regardless of j's specific value) rather than i:

```
for k = 1 to n:
  if A[k] > k:
    return k
// Cannot reach here by guarantees in the problem statement
```

This is a single loop over n iterations with constant-time work inside. It stops when it completes k iterations; so, it takes $O(k) \subseteq O(n)$ time. (I.e., $O(k)$ is a better answer here than $O(n)$, although $O(n)$ is also correct.)

Note: many other tests can work in the conditional. This test `A[k] > A[k-1] + 1` does not quite work alone because the "jump" point may be at index 1. (Note that it's irrelevant whether you use 1- or 0-based indexing because the jump point is described in terms of the contents, not the index.)

To make that test work, you need to check $n - 1$ indexes (being careful of out-of-bounds accesses to the array) and, if you do not find a jump point, return $k = 1$.

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

   SOLUTION: Below.

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

FindK(A, left, right):

  if left > right:

      return left

  else:

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

      // Use the condition below or A[mid] != mid, or A[mid] == mid + (A[n] - n).
      // Or, negate one of these and swap the if and else branches.
      //
      // This test DOES NOT work because it doesn't tell you whether the jump
      // point is "to the left" or "to the right": A[mid] > A[mid-1] + 1.
      if A[mid] > mid:


          return FindK(A, left, mid-1)

      else:

          return FindK(A, mid+1, right)
```

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]**

   SOLUTION: See below.

Ordered list of women	Man	Woman	(Hidden) ordered list of men
1, 2	1	1	**does not matter**
2, 1	2	2	**does not matter**

   The second man **must** have the preference list $2, 1$. The women's preferences don't matter (as we know, since none of them ever go on a date, which means they never even "discover" any of their preferences).

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]**

   SOLUTION: The key line is marked with a comment. Both mentions on that line need to be circled.

```
1: procedure StableMarriage(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 ∈ M
5:         propose to the next woman w ∈ 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          ▷ THIS is the key line.
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
```

   Most of the solution is above. In addition, circling the one mention of m and one of m' **inside** of that particular branch of the conditional is fine. (Yes, the date must happen by those lines, but no they're not the first lines where it's required.)

   However, circling any of the other mentions of m—even the one at the bottom of the body of the loop—is incorrect. To see why, trace what happens for the example that answers the previous part!

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]**

   SOLUTION: n. The man must propose to his most-preferred woman, but he can only know who his most-preferred woman is by knowing (at minimum!) for that most preferred woman how she compares to every other woman. To do that, he needs to date everyone.

4 eXtreme True And/Or False [9 marks]

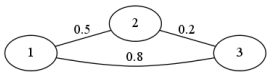
Each of the following statements may be **always** true, **sometimes** true, or **never** true. Select the best of these three choices and then:

- 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]
SOLUTION: We removed the options and just wrote in the answer in bold.

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.

   SOMETIMES

   Consider this graph: 

   (1) If we put $\{1, 3\}$ in one subset and just $\{2\}$ in the other, then the MST (which includes the edges weighted 0.5 and 0.2) will have two edges crossing from one subset to the other. (2) Either of the other two partitionings (e.g., $\{1, 2\}$ and $\{3\}$) has exactly one edge crossing from one subset to the other.

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.

   SOMETIMES

   (1) Using the notation above (but without hidden lists), consider this example:

	Man	Woman	
1, 2	1	1	1, 2
1, 2	2	2	2, 1

   The only stable matching has m_1 paired with w_1 (because they are each others' top choices) and therefore m_2 paired with w_2. (m_2 would like to switch to w_1, but w_1 does not want that switch.) Thus, w_1 is preferred to w_2 by all men, but w_2 gets her top choice.

   (2) In the example above, switch w_2's preferences to $1, 2$. The stable matching stays the same, and yet now w_2 does not get her top choice.

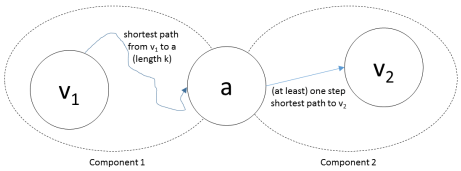
3. An articulation point in a simple, unweighted, undirected graph is diametric. (Recall that if the shortest path between a vertex i and another j is equal to the graph's diameter, we will say of i that it is a "diametric" vertex.)

   NEVER

(Note: we used a slightly different marking scheme for this part to put more weight on the proof sketch. We'll talk more about marking scheme elsewhere, but it seemed worthwhile to point this out!)

(1) Consider this simple graph: `A -- B -- C`. `B` is an articulation point dividing the graph into two disconnected subgraphs, one containing `A` and one `C`. However, the diameter stretches from `A` to `C`. (2) We'll prove that there's always a pair of vertices with a longer shortest-path than the articulation point paired with any node.

Imagine two nodes in a graph, one named a is an articulation point. The other named v_1 is some other vertex. The shortest path from a to v_1 is of length k. If we removed a, it would disconnect the graph into at least two components. Without loss of generality, we'll say that v_1 is in component 1. Component 2 has at least one node v_2. The shortest path from v_1 to v_2 must go through a (because otherwise removing a would **not** disconnect the two nodes). Furthermore, it must use the shortest path from v_1 to a to reach a (or we could make it shorter by swapping out that portion of the path for the shortest path from v_1 to a). Since that path from v_1 to v_2 goes at least one step beyond a to reach v_2, its length is at least $k + 1$.



Here's an illustration of this situation:

In other words, some other pair besides the articulation point and an arbitrary node will always have a longer shortest path, and the articulation point **cannot** be diametric.

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

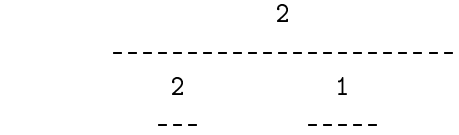
   (b) delete that event and all events that conflict with it from the list.

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.

   SOLUTION: Consider this one:



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

   SOLUTION: This algorithm first chooses the top event (because it has maximum value and the first start time). Then, it's out of events. So, it produces that single long 2-valued event as its solution set.

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

   SOLUTION: The optimal solution is to choose the two bottom events. Since the two solutions tie on the first event, we then compare the 1-valued event in the optimal solution against the "human interest stories" of 0 value left in the algorithm's solution. The optimal solution wins.

   CRITICAL NOTE: The metric for the value of a solution in this problem is **not** the total value of all events scheduled. How can I tell? Read the text above (identical to our quiz) that describes the metric! (I.e., comparing by highest-valued event, in case of ties by second-highest, etc.)

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.

SOLUTION: We give the algorithm in two pieces.

First, we take in an instance of ISP as n intervals with start times and durations: $\{(s_1, d_1), \ldots, (s_n, d_n)\}$ and return an instance of OSP as n events with start times, finish times, and values: $\{(s_1, f_1 = s_1 + d_1, 1), \ldots, (s_n, f_n = s_n + d_n, 1)\}$. Note that all events have the same value.

Next, after calling on the solution to OSP, we take a solution to OSP as a set of events $S = \{e_1, \ldots, e_k\}$ and return $|S| = k$.

You didn't need to argue for correctness here, but briefly: Since all values in the OSP instance are the same, the one with the most events will be optimal (because when others run out of events, they must show human interest stories). That means the OSP solution will produce the largest set of non-conflicting events. ISP and OSP have the same rules for conflict (except that ISP expresses the end time using a duration rather than directly). Thus the count of events in OSP's solution is the same as in the best solution to ISP.

ADDITIONAL NOTES: It **does** work to set the value to something like negative finish time in order to enforce an approach vaguely similar to the greedy solution to the standard greedy unweighted interval scheduling solution. Similarly, it works to set values to start times, which produces the symmetric greedy solution approach.

While it does also work to sort the events by start or finish time and then do something similar but using array indexes, this isn't as good an approach because of the unnecessary sorting (that dominates the runtime of the reduction).

NON-WORKING IDEAS: It is **not** correct to weight events based on having the most or fewest conflicts (and is much harder than the correct scheme above) or on greatest or least duration. See if you can see why.

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                                   ▷ This conditional takes O(1) time.
end if
                                  ▷ Initializing A takes O(n) time. Sorting it takes O(n lg n)
A ← an array 1...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                                                              ▷ O(1)
LastID ← None                          ▷ Assume None does not compare equal to any ID. O(1)
for i ← 2...n do                                                  ▷ n − 1 iterations
                                                  ▷ The whole loop body takes constant time.
    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                                   ▷ O(1)
```

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.

SOLUTION: Referring to the comments above, we have $O(n + n \lg n)$ time in initialization, various $O(1)$ time steps, and a loop that takes $O(n)$ time ($n - 1$ iterations each of constant time). All together, that's $O(n + n \lg n + 1 + n) = O(n \lg n)$.

(It is also plausible to say initialization of A also takes constant time if you see A as just a "view" of the log rather than actually a new array.)

7 BONUS: From the Cutting-Room Floor [2 BONUS marks]

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 midterm exam and toward the course's bonus point reward program—is your total score here, **rounded down**. We will be ridiculously harsh marking these. **Don't waste your time here!**

For your (in)convenience, we've rated—and sorted—the problems from "hard-ish" to "hardest-ish".

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.)

   SOLUTION: Left as a challenging exercise! However, note that $2^{\lg n} = n$ and that $(2n)!$ has n (and more!) terms that are at least n in value. We're not working with $(2n)!$—and $(2n)! \notin \Theta(2n-2)!$—but a similar argument will be helpful.

2. The algorithm given in Many Blocks That Used to Be a Log 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. (Hard-ish.)

   SOLUTION: Left as an exercise. However, try any log with a single entry in it and see what happens.

3. *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. (Harder-ish.)

   SOLUTION: Left as an exercise. However, remember from our previous work that the last woman is unengaged before the last iteration, at which point she accepts the first proposal made to her.

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

   SOLUTION: Left as a quite challenging exercise. Consider sorting the events by start time and then asking for each index i for the optimal solution in which that event is the one with the latest start time that we include.