Homework 4

Winston (Hanting) Zhang

Friday, September 30, by 23:00

[**Important Note: To allow us to return sample solutions two days before the midterm, this assignment cannot be submitted more than three (3) days late, i.e., no later than Monday, October 3, by 23:00.**]

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete, so please allow yourself plenty of time to work on it. (This assignment may be a bit more substantial than the third; in particular, Question 3 is not easy.) We highly recommend reading the entire assignment right away — you never know when inspiration will strike. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time.

Collaboration. Please carefully check the collaboration policy on the course website. When in doubt, ask an instructor.

Consulting outside sources. Please carefully check the policy regarding outside sources on the course website. Again, when in doubt, ask an instructor.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. Starting with this homework, only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

Recommended practice problems (do not hand in): KT Problems 5.1, 5.3, 5.5, 5.6.

Problem 1. [1+1+1+2+2=7]

Solve the following recurrences. In all cases, assume $T(1)$ is some positive constant. You should be able to find a matching upper and lower bound on $T(n)$. In other words, your answer should be of the form $T(n) = \Theta(\cdots)$. You do not need to provide formal proofs, just a brief justification for your answer. As we often due in class, you may sacrifice formality by ignoring issues with rounding integers; in all these recurrences, those issues will not matter.

We apply a generous amount of the Master Theorem.

1. $T(n) = 9T(n/3) + c_1 \cdot n^2$

   $a = 9$, $b = 3$, $c = 2$. We have $\log_b a = \log_3 9 = 2 = c$. This is case 2, i.e. $\boxed{T(n) = \Theta(n^2 \log n)}$.

2. $T(n) = 10T(n/5) + c_2 \cdot n^{1.5}$

   $a = 10$, $b = 5$, $c = 1.5$. We have $\log_b a = \log_5 10 \approx 1.43 < c$. This is case 3, and we need to check the regularity condition. We have $af(n/b) = 10c_2(n/5)^{1.5}$. Since $5^{1.5} \approx 11$, $10/5^{1.5} < 1$. Hence we can choose some $10/5^{1.5} < k < 1$ such $af(n/b) \le kf(n)$, and regularity holds. Thus $\boxed{T(n) = \Theta(n^{1.5})}$.

3. $T(n) = 3T(n/4) + c_3 \cdot \sqrt{n}$

   $a = 3$, $b = 4$, $c = 0.5$. We have $\log_b a = \log_4 3 \approx 0.79 > c$. This is case 1. i.e. $\boxed{T(n) = \Theta(n^{0.79})}$.

4. $T(n) = T(n/3) + T(\frac{2}{3}n) + c_4 \cdot n$

   Consider the recursion tree. On the 0^{th} layer, we do $c_4 n$ work on a input of size n and split into two subproblems. On the 1^{th} layer, we have a total input size of $n/3 + 2n/3 = n$ and hence do a total of $c_5 n$ work again. On the 2^{th} layer, the total input size stays n, and we do $c_5 n$ work yet again. At each layer in the tree, the subproblem size skrinks by $2/3$ and $1/3$. This means there can be a maximum $\log_{3/2} n$ layers, giving a upper bound of $c_5 n \log_{3/2} n = O(n \log n)$ work. However, note that since we generate unbalanced subproblems, this tree is not full, as the $1/3$ branches will shrink faster and hit the base case before $\log_{3/2} n$ layers. We do know, however, that the tree will have a minimum of $\log_3 n$ layers. Hence we are lower bounded by $c_5 n \log_3 n = \Omega(n \log n)$. Combining these two bounds gives $T(n) = \boxed{\Theta(n \log n)}$.

5. $T(n) = T(n/4) + T(n/2) + c_5 \cdot n$

   Consider the recursion tree. On the 0^{th} layer, we do $c_5 n$ work on a input of size n and split into two subproblems. On the 1^{th} layer, we have a total input size of $n/4 + n/2 = 3n/4$ and hence do a total of $c_5 3n/4$ work. On the 2^{th} layer, the total input size skrinks by $3/4$ again, and we do $c_5(3/4)^2 n$ work. This pattern continues to hold with layer k having $c_5(3/4)^k n$ work. Thus the total work on the tree is the geometric series given by: $T(n) = \sum_{k=0} c_5 \left(\frac{3}{4}\right)^k n = c_5 n \frac{1}{1-3/4} = \boxed{\Theta(n)}$.

Problem 2. [3+7=10]

Consider an array of integers $A = [a_1, \ldots, a_n]$ with $a_1 < a_n$.

1. Show that there exists an index $i \in \{1, \ldots, n-1\}$ such that $a_i < a_{i+1}$. In other words, there exist two consecutive entries of the array of which the first is smaller than the second.

   Proof. We prove the contrapositive: Suppose for every $i \in \{1, \ldots, n-1\}$, $a_i \ge a_{i+1}$. Then $a_1 \ge a_n$. This is clearly true by the transitivity of $\ge$. □

2. Give and analyze an algorithm which finds such a pair of entries in $O(\log n)$ time. Your algorithm should output $i \in \{1, \ldots, n-1\}$ such that $a_i < a_{i+1}$. You may assume that the array A has already been loaded into memory. You may also assume that comparing two entries of the array is a primitive operation (i.e., takes constant time).

Proof. We proceed by strong induction on n. The base case $n = 2$ is true, since finding the consecutive entries is trivial, taking $\log 2 = 1$ iteration.

Now assume that our claim is true for all $k < n$. We want to show that our claim is true for n. Indeed, we query index $a_{\lfloor n/2 \rfloor}$ and do binary search. If $a_1 < a_{\lfloor n/2 \rfloor}$, then our previous result guarantees there exists an index $\{1, \ldots, \lfloor n/2 \rfloor - 1\}$ such that $a_i < a_{i+1}$. By our induction hypothesis, finding such an index will take $\log(n/2)$ iterations. If instead $a_1 > a_{\lfloor n/2 \rfloor}$, then we conclude that $a_{\lfloor n/2 \rfloor} < a_n$. Hence our previous result and induction hypothesis again guarantees that we can find an index $\{\lfloor n/2 \rfloor - 1, \ldots, n-1\}$ such that $a_i < a_{i+1}$ in $\log(n/2)$ iterations. Thus in total, we can find the desired index for n in $\log(n/2)+1 = \log n - 1 + 1 = \log n = O(\log n)$ iterations. Each iteration takes constant time, so the total runtime is $O(\log n)$. ☐

[Hint: The first part may be useful here.]

Problem 3. [10]

Consider the following situation. At the end of the semester, David has two arrays a, b of weighted average scores of the students in the two sections of CSCI 270. Let's say that both sections have the same size n, which you may assume is a power of 2 if it simplifies things for you.

He has already sorted the students by score, so $a[1] \leq a[2] \leq \cdots \leq a[n]$, and $b[1] \leq b[2] \leq \cdots \leq b[n]$. However, he has not yet done any comparison of students between the two sections: it is possible that all numbers in the a array are larger, or all numbers in the b array are larger, or they are mixed in any possible way.

David now wants to know how the "most middle" student performed. That is, he wants to find a student i (out of the $2n$ total) such that at least n students had scores no higher than i, and at least n students had scores no lower than i. There may be multiple such students; for instance, if all students achieved the same score, then any student would do. Give an algorithm that always runs in time $O(\log n)$ and finds one such student, by outputting the array (a or b) and the index i where the student occurs in that array.

Definition 1. *Let a be an array of even length $\ell = 2n$. Student i is a* ***moodian*** *of a if at least n students have scores no higher than $a[i]$, and at least $n+1$ students have scores no lower than $a[i]$.*

Remark. We need this definition because the given definition of a median is slightly too ambiguous for our needs. Note that any moodian of a is also a median of a.

Proposition 1. *Let a be an array of even length $\ell = 2n$. A moodian of a exists.*

Proof. Sort a so that $a[1] \leq a[2] \leq \ldots \leq a[2n]$. Then $a[n]$ is no higher than at least $a[1]$ to $a[n]$, a.k.a. n students, and also no lower than at least $a[n]$ to $a[2n]$, a.k.a. $n+1$ students. Hence $a[n]$ is a moodian. ☐

Algorithm 1 MOODIAN-FIND

```
Require: Sorted arrays $a$, $b$; $a$.length = $b$.length = $n$
  procedure MOODIAN-FIND($a$, $b$)
    let $n \leftarrow a$.length
    if $n = 1$ then
      return smaller of $a[1]$ and $b[1]$
    if $a[n/2] = b[n/2]$ then
      return student at $a[n/2]$
    else if $a_m < b_m$ then
      return MOODIAN-FIND($a[n/2+1 : n], b[1 : n/2]$)
    else
      return MOODIAN-FIND($a[1 : n/2], b[n/2+1 : n]$)
```

Proposition 2. MOODIAN-FIND*(a, b) finds the moodian of a and b in $O(\log n)$ time.*

Proof. Suppose that $n = 2^k$ for some k, to make the proof cleaner.

We proceed by strong induction on n to show that it takes $O(\log n)$ iterations to find the moodian. The base case $n = 1$ is trivial, since we simply take the smaller of $a[1]$ and $b[1]$, taking 1 iteration.

Now assume that our claim is true for all $k < n$. We want to show that our claim is true for n. Indeed, we query index $a[n/2]$ and $b[n/2]$. Now there are three cases:

1. $a[n/2] = b[n/2]$: Then we know there are at least $n/2$ students with scores no higher than $a[n/2] = b[n/2]$ from both a and b, for a total of at least n students no higher than $a[n/2]$. For the other side, we know there are at least $n/2 + 1$ students with scores no lower than $a[n/2] = b[n/2]$ from both a and b, for a total of at least $n + 2$ students no lower than $a[n/2]$. Thus $a[n/2]$ is a moodian and we end the search in $1 = O(\log n)$ iterations.

2. $a[n/2] < b[n/2]$: Consider the subarrays $a' = a[n/2 + 1 : n]$ and $b' = b[1 : n/2]$, so that $a'[i] = a[n/2 + i]$ and $b'[i] = b[i]$. We claim that the moodian of a' and b' is the moodian of a and b.

   Indeed, let m be the moodian of a' and b'. We claim that $a[n/2] \leq m \leq b[n/2]$. Without loss of generality assume that $m \in a'$; if $m \in b'$ simply switch $a' \iff b'$. We can deduce the LHS of the inequality we want using the fact that a is sorted: $a[n/2] \leq a[n/2 + 1] = a'[1] \leq m$.

   For the RHS of the inequality, assume for the sake of contradiction that $b[n/2] < m$. Then all of the elements of b' are lower than m. Hence everything no lower than m must be from a'. Since a' has $n/2$ elements, this implies that there can be a maximum of $n/2$ student no lower than m. This contradicts the fact that m is a moodian, since by definition, we need at least $n/2+1$ students to be no lower than m. Thus $m \leq b[n/2]$. Note: this slight difference between $+1$ for the median/moodian is the difference maker; if we did not have the slightly stricter condition, and had the ambigutiy of which median to choose, the given algorithm doesn't work.

   Since m is the moodian of a' and b', at least $n/2$ students are no higher than m. Since $a[n/2] \leq m$, we also have $n/2$ students from $a[1 : n/2]$ no higher than m. By definition $a[1 : n/2]$ and a' are disjoint, so in total we can say that there are at least $n/2 + n/2 = n$ students in a and b no higher than m. On the other side, there are $n/2+1$ students no lower than m in a' and b'. Since $m \leq b[n/2]$, we also have $n/2$ students from $b[n/2+1 : n]$ no lower than m. By definition $b[n/2 + 1 : n]$ and b' are disjoint, so in total we can say that there are at least $n/2 + 1 + n/2 = n + 1$ students in a and b no lower than m. These two conditions prove that m is indeed the moodian of a and b.

   By our induction hypothesis, finding the moodian of a' and b' takes $\log(n/2)$ iterations. Thus in total, we can find the desired index for n in $\log(n/2)+1 = \log n - 1 + 1 = \log n = O(\log n)$ iterations. Each iteration takes constant time, so the total runtime is $O(\log n)$.

3. $a[n/2] > b[n/2]$: By symmetry, this is the same case as above, just switch a and b.

We have shown the correctness of MOODIAN-FIND and that it has $O(\log n)$ runtime, thus we are done. $\square$

Problem 0. [0]

Chocolate Problem: 1 chocolate bar

Reminder: If you solve a chocolate problem (which you can do in groups of size up to 3), please e-mail David with the solution — do not submit it on Gradescope. Also, feel free to list preferences or dietary restrictions for/against particular types of chocolate.

In discussion section, you saw how to generalize the idea of Binary Search to trees: each tree has a node v such that when you query that node and the answer points to one of the subtrees from that node, there are at most $n/2$ nodes in that subtree. Thus, repeating this querying at most $\log_2(n)$ times, you can find any node in a tree from the answers to such queries.

Here, we want to generalize the idea further, from trees to arbitrary undirected graphs. In non-tree graphs, it does not make sense to talk about "the subgraph containing the node". For instance, if your graph is a cycle, then no matter which node you query, you can go around the cycle in either direction to get to any other node. So we will clarify the answer as follows: when a node v is queried, and the correct answer is t, the answer will reveal an edge out of v that lies on a *shortest* path from v to t. If there are multiple shortest paths from v to t (with different edges out of v), then any of them could be returned.

You now play the following game: both you and your opponent know the undirected graph G. Your opponent picks a node $t \in G$, without telling you what it is. In each round, you get to point to a vertex v. If $v = t$, then the game is over. Otherwise, your opponent reveals an edge e incident on v that lies on a shortest path from v to t. The game repeats until you've found t. Your goal is to get there with few queries. Prove the following:

Lemma 3. *For every graph $G = (V, E)$, there exists a vertex v you can query such that never mind which edge e incident on v your opponent reveals, the set of remaining vertices that are consistent with this answer shrinks by at least a factor 2. That is, if S is the set of all nodes t such that e is on a shortest path from v to t, then $|S| \leq |V|/2$.*

To round out the answer, show how to use the lemma to guarantee that you can find the node t in at most $\log_2 n$ queries — this part is basically trivial once you have proved the lemma.