Homework #3
Introduction to Algorithms/Algorithms 1
601.433/633
Spring 2018

Due on: Tuesday, February 27th, 5.00pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: Gradescope.

Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

March 8, 2018

Problem 1 (10 points)

You are given an array A of n integer. All numbers except for $O(n^{1-\varepsilon})$ are in the range $[1, 1000n^2 - n]$. Find an algorithm that sorts an array A in $O(n)$ time in the worst case given that $\varepsilon \in (0, 1)$. Provide running time analysis and correctness proof for your algorithm.

Solution

Our algorithm:

1. Create two arrays: array R and array M ;
2. Go through array A and if $A[i]$ is not from the range (at most two comparisons), put it into array M, if its from the range put it into the array R;
3. Sort array M using merge sort;

4. Sort array R using radix sort;
5. Merge two sorted arrays R and M back into array A;
6. output array A.

Correctness proof for radix sort, merge sort and merge step was given in the class (or book). In fact, due to given conditions every element will go either to array M or to array R, neither will go to both, thus $A = R \cup M$ and $R \cap M = \emptyset$. Thus if we sort R and M and merge them, then we get sorted A. Second step of the algorithm takes $O(n)$ time, while running time for radix sort was proved to be linear in the class(or book) (when $A[i] \in \{1, \ldots, n^{\alpha}\}$ and $\alpha = O(1)$ Merge sort works in time $O(n_0 \log n_0)$, where n_0 is the size of the array M . But conditions of the problem state that $n_0 = O(n^{1-\varepsilon})$, thus running time of the merge sort is

$$T_M(n) = O(n_0 \log n_0) \rightarrow T_M(n) = O(n^{1-\varepsilon}(logn^{1-\varepsilon})) \rightarrow T_M(n) = O(n)$$

. Running time for the merge sort was also proved to be linear in the class. Thus total running time is $O(n)$.

Problem 2 (10 points)

In class we considered the algorithm SELECT, which determines the ith smallest element in the array of size n in $O(n)$ time for the worst case input. The first step of this algorithm is division into $n/5$ groups of 5 elements each. Consider other two versions of this algorithm: the first one uses division into $n/3$ groups of 3 elements each and the second one uses division into $n/7$ groups of 7 elements each. Otherwise both algorithms implement the same routine as SELECT. Which algorithm is **asymptotically** faster, the one with division into groups of 3, groups of 5 (SELECT) or groups of 7? Prove your statement.

Solution

First, let us find lower bound for the asymptotic runtime of division into $n/3$ groups of 3 elements each. The number of elements that are for sure less than the median-of-medians is at most

$$2 * (\left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil) \leq \frac{n}{3} + 3 \qquad (1)$$

So, the recurrence is then

$$T(n) \geq T(\left\lceil \frac{n}{3} \right\rceil) + T(\frac{2n}{3} - 3) + \Omega(n) \qquad (2)$$

Let us guess that

$$T(n) > cn \log n \tag{3}$$

$$T(n) > c\frac{n}{3}\log\frac{n}{3} + c\left(\frac{2n}{3} - 3\right)\log\left(\frac{2n}{3} - 3\right) + an \tag{4}$$

For large n we can conclude that $\log(\frac{2n}{3} - 3) > \log(\frac{n}{2})$, then substitute it:

$$T(n) > c\frac{n}{3}\log n - \frac{cn}{3}\log 3 + c\frac{2n}{3}\log\frac{n}{2} - 3c\log\frac{n}{2} + an \tag{5}$$

$$T(n) > cn\log n + n\left(a - \frac{c}{3}\log 3 - \frac{2c}{3} - 3c\frac{\log n}{n} + \frac{3}{n}\right) \tag{6}$$

Thus for small enough c we can conclude that $T(n) > cn\log n$, thus $T(n) = \Omega(n\log n)$.

Now, let us prove the asymptotic runtime of division into $n/7$ groups of 7 elements each. The number of elements that are less than the median-of-medians is

$$4 * (\left\lceil \frac{1}{2}\left\lceil \frac{n}{7} \right\rceil \right\rceil - 2) \geq \frac{2n}{7} - 8 \tag{7}$$

So, we now have the relation:

$$T(n) = \begin{cases} O(1) & \text{if } n < n_0 \\ T(\lceil \frac{n}{7} \rceil) + T(\frac{5*n}{7} + 8) + O(n) & \text{otherwise} \end{cases}$$

Let us guess that

$$T(n) \leq cn \tag{8}$$

$$T(n) \leq c\left\lceil \frac{n}{7} \right\rceil + c(\frac{5n}{7} + 8) + an \tag{9}$$

$$T(n) \leq c\left\lceil \frac{n}{7} \right\rceil + c(\frac{5n}{7} + 8) + an \tag{10}$$

$$T(n) = c\frac{6n}{7} + 9c + an \tag{11}$$

$$T(n) \leq cn \tag{12}$$

By choosing $n_0 = 126$ and $n \leq n_0$ and $c \geq 14a$, our proof holds. We conclude that $n/7$ groups provides $O(n)$ runtime. We know from class that $n/5$ groups provides $O(n)$ runtime. We also now that any median search algorithm at least need to read all elements, thus any median search algorithm should have lower bound of $\Omega(n)$. Thus we can conclude that algorithms with groups of 5 and groups of 7 have complexity $\Theta(n)$, and assymptotically faster than algorithm with groups of 3 which is slower as it has lower bound of $\Omega(n\log n)$.

Problem 3 (10 points)

Professor asked Bob to find the median of an integer array A, of size n, which is stored on the lab server. Bob has access to the server, however, his rights are very limited: he can only read data from the server, but cannot write to the server.
The array is so large that Bob can not just copy it to his machine. Bob's computer has only $O(\log n)$ memory. Help Bob to develop an efficient algorithm which finds the median of A. Provide a correctness proof and running time analysis. Full score will be given for $O(n \log n)$ expected time algorithm.

Solution

Here is a version of this algorithm that does not use recursion, although this algorithm can easily be implemented recursively as well.

```
FIND-ARRAY-MEDIAN(array $A$, size $n$)
  1. $l = -\infty$
     //the lower bound of our search.
  2. $L = 0$
     //the number of array elements less than $l$
  3. $r = -\infty$
     //the upper bound of our search. The median is found if $l = r$
  4. $R = 0$
     //the number of array elements greater than $r$
  5. while $l \neq r$ and $L < \lfloor \frac{n}{2} \rfloor$:
     (a) $e = n - (L + R)$
         //$e$ is the number of elements in the inclusive range $[l, r]$
     (b) $i =$ random in $[1, e]$
     (c) $g =$ the $i^{th}$ element of $A$ in the range $(l, r)$
         //scan $A$ in order and select random element $g$ s.t. $l \leq g \leq r$ as our guess
     (d) $c_l = 0$
     (e) $c_r = 0$
```

```
(f) for each element in $a \in A$ :
    //count the number of elements in $A$ that are less than or greater than $g$, our guess
    i. if $a < g$, increment $c_l$ by one.
    ii. if $a > g$, increment $c_r$ by one.
(g) if $c_l + L < \lfloor \frac{n}{2} \rfloor$:
    //There are less than $\lfloor \frac{n}{2} \rfloor$ items less than $g$
    i. $l = g$
       //The median is at least $g$, so update the search range
    ii. $L = L + c_l$
       //update the number of items outside the search range
(h) if $c_r + R > \lfloor \frac{n}{2} \rfloor$:
    //There are more than $\lfloor \frac{n}{2} \rfloor$ items greater than $g$
    i. $r = g$
       //The median is at most $g$, so update the search range
    ii. $R = R + c_r$
       //update the number of items outside the search range
6. return $l$
```

The loop invariant condition for (5) is that the median of A is greater than or equal to l and less than or equal to r. Before the first loop is executed, this condition holds because we assume all elements are greater than $-\infty$ and less than ∞. After each execution:

1. l is only updated to some new value l' if there are less than $\lfloor \frac{n}{2} \rfloor$ items less than l'. The median cannot be less than l', because if the median were some $g < l'$, than more than half of the elements in A would be greater than g, a contradiction. Therefore the loop invariant is maintained.

2. r is only updated to some new value r' if there are less than $\lfloor \frac{n}{2} \rfloor$ items greater than r'. The median cannot be greater than r', because if the median were some $g > r'$, than more than half of the elements in A would be less than g, a contradiction. Therefore the loop invariant is maintained.

The algorithm terminates in one of two conditions:

1. $l = r$ in this case, the median is $l = r$ and this value is returned.

2. In the case where we have an even-size array A, with the two distinct center elements $A\left[\left\lfloor \frac{n}{2} \right\rfloor\right] \neq A\left[\left\lceil \frac{n}{2} \right\rceil\right]$, it will not be the case that we find $l = r$, because l and r will be our two distinct center elements. In this edge case, we terminate when we have selected the element such that $L \geq \left\lfloor \frac{n}{2} \right\rfloor$, that is, the right center element, which is one of two medians.

Because iterations of the while loop repeatedly narrow the search space until the median is found, we expect that the algorithm will terminate and produce a correct result.

Runtime:
All steps but (5) take trivial time. All substeps of (5) except (c) and (f) take trivial time. Step (c) and (f) both scan the array taking $O(n)$ time. Therefore, each iteration of the while loop takes $O(n) + O(n) = O(n)$ time.

We can modify the time analysis from the book (for randomised select algorithm). Let's define the size of the problem m to be number of elements between l and g. Then we can rewrite a recurrence:

$$T(m) = \sum_{k=1}^{m} X_k \left(T(max(k-1, m-k)) + O(n)\right),$$

where X_k is an indicator random variable that pivot has rank k among elements between l and g. Size of the problem goes down, but the we still spend $O(n)$ on each run. Further we will just rewrite formulas from the book which holds due to linearity of expectation and uniformity of distribution of our random pivot.

$$E(T(m)) = \frac{1}{m} \sum_{k=1}^{m} E(T(max(k-1, m-k))) + O(n)$$

$$E(T(m)) < \frac{2}{m} \sum_{k=m/2}^{m-1} E(T(k)) + O(n)$$

Suppose that $O(n) < c_1 n$. Now we will check by substitution that $E(T(m)) < c_2 n \log m$:

$$E(T(m)) < \frac{2}{m} \sum_{k=m/2}^{m-1} c_2 n \log k + c_1 n$$

$$E(T(m)) < \frac{2c_2 n}{m} \sum_{k=m/2}^{m-1} \log k + c_1 n$$

$$E(T(m)) < c_2 n \log m$$

Where the last transition we leave for students as an exercise (find small enough c_2 for which it works).

At the very begining the size of the problem m is equal to n, thus our expected running time is $O(n \log n)$.

Because we need only store our bound, guess, and count variables, and we examine the array one element at a time, we have sufficient memory.

Problem 4 (20 points)

Resolve the following recurrences. Use Master theorem, if applicable. In all examples, assume that $T(1) = 1$. To simplify your analysis, you can assume that $n = a^k$ for some a, k.

1. $T(n) = 2T(n/8) + n^{\frac{1}{5}} \log n \log \log n$
2. $T(n) = nT(n/6)$
3. $T(n) = 16T(n/2) + \sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i})$
4. $T(n) = 8T(n/4) + n$
5. $T(n) = 4T(n-2) + 2^{2n}$
6. $T(n) = 8T(n/2) + n^3$
7. $T(n) = T(n/2) + \log n$
8. $T(n) = T(n-1) + T(n-2)$

Solution

1. $T(n) = 2T(n/8) + n^{\frac{1}{5}} \log n \log \log n$

   $\forall n \geq n_0$, $n^{\frac{1}{5}} \log n \log \log n \leq n^{\frac{1}{5}} n^{\frac{1}{100}} n^{\frac{1}{100}} \leq n^{\frac{1}{4}} \leq cn^{\log_8 2 - \epsilon} = cn^{\frac{1}{3} - \epsilon}$ for $0 < \epsilon < \frac{1}{100}$ and sufficiently large c, n_0. Therefore, $f(n) \in O(n^{\frac{1}{3} - \epsilon})$. Therefore, $T(n) = \Theta(n^{\frac{1}{3}})$ by case 1 of the master theorem.

2. $T(n) = nT(n/6)$

   Assume $n = 6k$, then

   $$T(n) = 6^k \cdot 6^{k-1} \cdot 6^{k-2} \cdot \ldots T(6k/6k) = 6^{\frac{k^2+k}{2}}$$

thus $T(n) = n^{\frac{\log_6 n+1}{2}}$

3. $T(n) = 16T(n/2) + \sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i})$

   First of all let's estimate $f(n) = \sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i})$.
   We know that $1 < \ln(e + 1/i) < 2$ then
   $\sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i}) > \sum_{i=1}^{n} i^3 > \sum_{i=n/2}^{n} i^3 > \frac{n}{2}(\frac{n}{2})^3 > C_1 n^4$
   and
   $\sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i}) < 2\sum_{i=1}^{n} i^3 < 2n^4 < C_2 n^4$. Thus $f(n) = \Theta(n^4) = \Theta(n^{\log_b^a})$, thus it's case 2 from Master Theorem. Thus $T(n) = \Theta(n^4 \log n)$.

4. $T(n) = 8T(n/4) + n$

   $a = 8, b = 4, log_b a > 1$. By applying the case 1 of Master Theorem, $T(n) = \Theta(n^l og_4 8) = \Theta(n^{3/2})$

5. $T(n) = 4T(n-2) + 2^{2n}$

   $n = \log a$
   $T(n) = T(\log a) = 4T(\log a - \log 4) + 2^{2\log a} = 4T(\log \frac{a}{4}) + a^2$
   $T(n) = T(\log a) = S(a) = 4S(\frac{a}{4}) + a^2$.
   Case 3 of Master theorem: $a^2 = \Omega(a^{1+\varepsilon})$, and for $C = 1/2$:
   $4f(\frac{a}{4}) = \frac{4n^2}{16} = \frac{n^2}{4} \leq Cf(n) = Ca^2$ Thus $S(a) = \Theta(f(n)) = \Theta(a^2)$. But at the same time $T(n) = \Theta(2^{2n})$

6. $T(n) = 8T(n/2) + n^3$

   $\forall n \geq n_0$, $0 \leq c_1 n^{log_2 8} = c_1 n^3 \leq n^3 - 8n\log n \leq c_2 n^{log_2 8} = c_2 n^3$ for $c_1 = .5$, $c_2 = 1$, and sufficiently large n_0. Therefore, $f(n) \in \Theta(n^3)$. Therefore, $T(n) = \Theta(n^3 \log n)$ by case 2 of the master theorem.

7. $T(n) = T(n/2) + \log n$

   $n = 2^k$
   $T(2^k) = T(2^k/2) + k = T(2^{k-1}) + k$
   $T(2^k) = S(k) = S(k-1) + k$
   Thus $T(n) = S(k) = \sum_{i=1}^{k} i = \Theta(k^2) = \Theta((\log n)^2)$.

8. $T(n) = T(n-1) + T(n-2)$

Let $T(n) \leq ca^n$ hold for smaller values of n. Then,
$T(n) = T(n-1) + T(n-2) \leq ca^{n-1} + ca^{n-2} = ca^n + (\frac{c}{a} + \frac{c}{a^2} - c)a^n$
where $(\frac{c}{a} + \frac{c}{a^2} - c)a^n$ is a positive term iff for some positive a, $a^2 - a - 1 \leq 0 \rightarrow a \leq \frac{1+\sqrt{5}}{2}$.
Therefore $T(n) = O((\frac{1+\sqrt{5}}{2})^n)$

Let $T(n) \geq ca^n$ hold for smaller values of n. Then,
$T(n) = T(n-1) + T(n-2) \geq ca^{n-1} + ca^{n-2} = ca^n + (\frac{c}{a} + \frac{c}{a^2} - c)a^n$
where $(\frac{c}{a} + \frac{c}{a^2} - c)a^n$ is a negative term iff for some positive a, $a^2 - a - 1 \geq 0 \rightarrow a \geq \frac{1+\sqrt{5}}{2}$.
Therefore $T(n) = \Omega((\frac{1+\sqrt{5}}{2})^n)$
Therefore $T(n) = \Theta((\frac{1+\sqrt{5}}{2})^n)$
From $T(1) = 1$ we can conclude that constant $C = \left(\frac{1+\sqrt{5}}{2}\right)^{-1}$.