Homework #3
Introduction to Algorithms
601.433/633
Spring 2020

Mou Zhang

Due on: Tuesday, February 25th, 12pm
Format: Please start each problem on a new page.
Where to submit: On Gradescope, please mark the pages for each question

1 Problem 1 (24 points)

Recall that when using the QuickSort algorithm to sort an array A of length n, we picked an element $x \in A$ which we called the *pivot* and split the array A into two arrays A_S, A_L such that $\forall y \in A_S, y \leq x$ and $\forall y \in A_L, y > x$.

We will say that a pivot from an array A provides $t|n-t$ separation if t elements in A are smaller than or equal to the pivot, and $n-t$ elements are strictly larger than the pivot.

Suppose Bob knows a secret way to find a good pivot with $\frac{n}{3}|\frac{2n}{3}$ separation in constant time. But at the same time Alice knows her own secret technique, which provides separation $\frac{n}{4}|\frac{3n}{4}$, her technique also works in constant time.

Recall that in the QuickSort algorithm, as per Section 7.1 in CLRS, the PARTITION subroutine picks a pivot x from A by simply picking the first element in the array. Alice and Bob's subroutines are subroutines for picking the pivot x in the PARTITION subroutine for QuickSort.

Alice and Bob applied their secret techniques as subroutines in the QuickSort algorithm to pick pivots. Whose algorithm works **asymptotically** faster? Or are the

runtimes **asymptotically** the same? Prove your statement.

Solution:
At each step, the time complexity for reorder the elements around the pivot is O(n). So for Bob, his time complexity has the relation $T_B(n) = T_B(\frac{n}{3}n) + T_B(\frac{2n}{3}n) + O(n)$. For Alice we have $T_A(n) = T_A(\frac{n}{4}n) + T_A(\frac{3n}{4}n) + O(n)$. Since for the separation $\frac{n}{2}|\frac{n}{2}$ we have the relation that $T(n) = 2T(\frac{n}{2}) + O(n)$. The time complexity for this relation is $T(N) = \Theta(n \log n)$. So I suggest that the runtimes for T_B and T_A are asymptotically the same, which are both equal to $\Theta(n \log n)$. The prove is shown as below.

Proof.
1.ForT$_B$:
(1) We try to prove that for large enough C_B, n_0 and $\forall n \geq n_0$, $T_B(n) \leq C_B n \log n$.
Base case: If $n = 1$, then $T_B(n) = 0 \leq C_B 1 \log 1 = 0$, True.
Induction hypothesis: $\forall k < n$, $T_B(k) \leq C_B k \log k$.
Inductive Step:$T_B(n) = T_B(\frac{n}{3}) + T_B(\frac{2n}{3}) + Cn \leq C_B \frac{n}{3} \log(\frac{n}{3}) + C_B \frac{2n}{3} \log(\frac{2n}{3}) + Cn = C_B n \log n - (C_B \frac{\log 3}{3} + C_B \frac{2 \log \frac{3}{2}}{3} - C)n \leq C_B n \log n$. The last step holds for $C_B \geq \frac{3C}{\log 3 + 2 \log \frac{3}{2}}$. True.
Thus, $T_B(n) = O(n \log n)$.
(2)We try to prove that for small enough positive C_B and large enough n_0, and $\forall n \geq n_0$, $T_B \geq C_B n \log n$.
Base case: If $n = 1$, then $T_B(n) = 0 \geq C_B 1 \log 1 = 0$, True.
Induction hypothesis: $\forall k < n$, $T_B(k) \geq C_B k \log k$.
Inductive Step:$T_B(n) = T_B(\frac{n}{3}) + T_B(\frac{2n}{3}) + Cn \geq C_B \frac{n}{3} \log(\frac{n}{3}) + C_B \frac{2n}{3} \log(\frac{2n}{3}) + Cn = C_B n \log n + (C - C_B \frac{\log 3}{3} - C_B \frac{2 \log \frac{3}{2}}{3})n \geq C_B n \log n$. The last step holds for $C_B \leq \frac{3C}{\log 3 + 2 \log \frac{3}{2}}$. True.
Thus, $T_B(n) = \Omega(n \log n)$.
Therefore, $T_B(n) = \Theta(n \log n)$

2. For T_A:
(1) We try to prove that for large enough C_A, n_0 and $\forall n \geq n_0$, $T_A(n) \leq C_A n \log n$.
Base case: If $n = 1$, then $T_A(n) = 0 \leq C_A 1 \log 1 = 0$, True.
Induction hypothesis: $\forall k < n$, $T_A(k) \leq C_A k \log k$.
Inductive Step:$T_A(n) = T_A(\frac{n}{4}) + T_A(\frac{3n}{4}) + Cn \leq C_A \frac{n}{3} \log(\frac{n}{4}) + C_A \frac{3n}{4} \log(\frac{3n}{4}) + Cn = C_A n \log n - (C_A \frac{\log 4}{4} + C_A \frac{3 \log \frac{4}{3}}{4} - C)n \leq C_A n \log n$. The last step holds for $C_A \geq \frac{4C}{\log 4 + 3 \log \frac{4}{3}}$. True.

Thus, $T_A(n) = \Omega(n \log n)$.
(2)We try to prove that for small enough positive C_A and large enough n_0, and $\forall n \geq n_0$, $T_A \geq C_A n \log n$.
Base case: If $n = 1$, then $T_A(n) = 0 \geq C_A 1 \log 1 = 0$, True.
Induction hypothesis: $\forall k < n$, $T_A(k) \geq C_A k \log k$.
Inductive Step:$T_A(n) = T_A(\frac{n}{4}) + T_A(\frac{3n}{4}) + Cn \geq C_A \frac{n}{4} \log(\frac{n}{4}) + C_A \frac{3n}{4} \log(\frac{3n}{4}) + Cn = C_A n \log n + (C - C_A \frac{\log 4}{4} - C_A \frac{3 \log \frac{4}{3}}{4}) n \geq C_A n \log n$. The last step holds for $C_A \leq \frac{4C}{\log 4 + 3 \log \frac{4}{3}}$. True.
Thus, $T_A(n) = \Omega(n \log n)$.
Therefore, $T_A(n) = \Theta(n \log n)$
Since $T_B = \Theta(n \log n) = T_A$, T_B and T_A are asymptotically the same. $\square$

2 Problem 2 (13 points)

Resolve the **asymptotic complexity** of the following recurrences, i.e., solve them and give your answer in Big-Θ notation. 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.

Your final answer should be as simple as possible, i.e., it should not contain any sums, recurrences, etc.

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

   Solution:
   $\forall n \geq n_0$ and large enough c, n_0, $n^{\frac{1}{5}} \log n \log\log n \leq n^{\frac{1}{5}} \log \frac{1}{100} \log \frac{1}{100} \leq n^{\frac{1}{4}} \leq cn^{\log_8 2-\epsilon} = cn^{\frac{1}{3}-\epsilon}$ for $0 < \epsilon < \frac{1}{12}$. So $f(n) = O(n^{\frac{1}{3}} - \epsilon) = O(n^{\log_8 2} - \epsilon) = O(n^{\log_b a} - \epsilon)$. Owing to case 1 of the master theorem, $T(n) = \Theta(n^{\frac{1}{3}})$.

2. $T(n) = 8T(n/2) + n^3 - 8n \log n$

   Solution:
   $\forall n \geq n_0$ for a large enough n_0 , $0 \leq c1n^3 \leq n^3 - 8n \log n \leq c2n^3$ for $c1 = 0.5$ and $c2 = 1$. So $f(n) = \Theta(n^3) = \Theta(n^{\log_2 8}) = \Theta(n^{log_b a})$. Owing to case 2 of the master theorem, $T(n) = \Theta(n^3 \log n)$

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

   Solution:
   Assume $n = 2^k$, then we have $T(2^k) = T(2^k/2) + k = T(2^{k-1}) + k$. Assume $S(k) = T(2^k)$, then we have $S(k) = S(k-1) + k$. So $T(n) = S(k) = \sum_{i=1}^{k} i = \Theta(k^2) = \Theta((\log n)^2)$

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

   Solution:
   Assume that $T(n) \leq ca^n$for values smaller than n. $T(n) = T(n-1)+T(n-2) \leq ca^{n-1}+ca^{n-2} = ca^n+(\frac{c}{a}+\frac{c}{a^2}-c)a^n$. To make $(\frac{c}{a}+\frac{c}{a^2}-c)a^n$ positive, we have $a^2-a-1 \leq 0$, which means $a \leq \frac{1+\sqrt{5}}{2}$. Thus, $T(n) = O((\frac{1+\sqrt{5}}{2})^n)$

   Similarly, assume that $T(n) \geq ca^n$for values smaller than n. $T(n) = T(n-1) + T(n-2) \geq ca^{n-1} + ca^{n-2} = ca^n + (\frac{c}{a} + \frac{c}{a^2} - c)a^n$. To make $(\frac{c}{a} + \frac{c}{a^2} - c)a^n$ negative, we have $a^2 - a - 1 \geq 0$, which means

$a \geq \frac{1+\sqrt{5}}{2}$. Thus, $T(n) = \Omega((\frac{1+\sqrt{5}}{2})^n)$
Therefore, $T(n) = \Theta((\frac{1+\sqrt{5}}{2})^n)$

5. $T(n) = 3T(n^{\frac{2}{3}}) + \log n$

   Solution:
   Assume $n = a^k$, we have $T(n) = T(a^k) = 3T((a^k)^{\frac{2}{3}}) + \log a^k$
   Assume $Q(k) = T(a^k)$, then we have $Q(k) = 3Q(\frac{2}{3}k) + \log a^k$
   Since $k^{\log_{\frac{3}{2}} 3-\epsilon} \approx k^{2.7-\epsilon}$ for some ϵ, $\log a^k = k \log a = O(k^{\log_{\frac{3}{2}} 3-\epsilon})$.
   Thus, $T(n) = S(k) = \Theta(k^{\log_{\frac{3}{2}} 3}) = \Theta((log_a n)^{\log_{\frac{3}{2}} 3}) = \Theta((logn)^{\log_{\frac{3}{2}} 3})$

3 Problem 3 (13 points)

Let A and B be two sorted arrays of n elements each. We can easily find the median element in A – it is just the element in the middle – and similarly we can easily find the median element in B. (Let us define the median of $2k$ elements as the element that is greater than $k-1$ elements and less than k elements.) However, suppose we want to find the median element overall – i.e., the nth smallest in the union of A and B.

Give an $O(\log n)$ time algorithm to compute the median of $A \cup B$. You may assume there are no duplicate elements.

As usual, prove correctness and the runtime of your algorithm.

Solution:
My algorithm is described as below.
Step 1:
Calculate the medians m1 and m2 of the input arrays A and B respectively until the size of A and B becomes 2.
Step 2:
Case 1:
If $m1 = m2$, then we have found the median, which is m1(or m2). return m1(or m2).
Case 2:
If $m1 > m2$, then return to Step 1 with A replaced by A[0 ... len(A) / 2] (The first len(A) / 2 elements in A, which len(A) is the size of array A) and B replaced by B[len(B) / 2 ... len(B)](The last len(B) / 2 elements of array B, which len(B) is the size of array B).
Explanation: the median must be in these two sub-arrays. Because if the medium is in B, it must be bigger than m2 and if the medium is in A, it must be smaller than m1.
Case 3:
If $m1 < m2$, then return to Step 1 with A replaced by A[len(B) / 2 ... len(B)] (The first len(A) / 2 elements in A, which len(A) is the size of array A) and B replaced by B[0 ... len(A) / 2](The last len(B) / 2 elements of array B, which len(B) is the size of array B).
Explanation:the median must be in these two sub-arrays. Because if the medium is in B, it must be smaller than m2 and if the medium is in A, it must be bigger than m1.

Step 3: repeat the above procedure until the size of 2 searching subarrays becomes 2. After that, use the below formula the get the median.

$$\text{Median} = (\max(A[0], B[0]) + \min(A[1], B[1]))/2$$

Proof of Correctness

Proof. I gonna prove that the medium number is always in the searching area after every loop of steps below. And after doing all these loops, we can find the medium we want.

Initialization:

We start to show the the loop invariant holds before the first iteration. At The beginning, we can see that the medium is obviously in the searching area because it must in these 2 arrays.

Maintenance:

Induction hypothesis (IH):After k loops of step 1 and step2, the medium is still in the to-be-searched sub-array ares.

Induction step (IS): We gonna prove that after the k + 1 loop of step 1 and step 2, the medium is still in the to-be-searched sub-array ares. We know that medium is in our searching area before k+1 step. So if at k + 1 step, $m1 > m2$, then if the medium is in B, it must be bigger than m2 and if the medium is in A, it must be smaller than m1. So we can narrow the to-be-searched area to A[0 ... len(A) / 2] and B[len(B) / 2 ... len(B)]. On the other hand, if $m1 < m2$, then if the medium is in B, it must be smaller than m2 and if the medium is in A, it must be bigger than m1.So we can narrow the to-be-searched area to A[len(B) / 2 ... len(B)] and B[0 ... len(A) / 2]. If $m1 = m2$, it means that the medium of the two sub-arrays are the same, which can serve as the medium of the combined array. So after we narrow the to-be-searched area, the medium is either found or in the new searching area.

Termination:

So after many loops of step 1 and step 2, there are 2 possible cases. The first case is we have found the medium while these loops(step 2 case 1), then we found the medium successfully. The second case is that there are both only two numbers in to-be-searched sub-array in A and B. At this time, the medium is obviously the mean of the larger one in 2 first elements in arrays and the smaller one in 2 last elements in arrays. The answer can be presented as Median = (max(A[0], B[0]) + min(A[1], B[1]))/2. □

Proof of time complexity

Proof. Every time we go through Step1 and Step2, we halved the size of to-be-searched array. In the first step ,there are m = n + n elements in the to-be-searched array. And when we go through step1 and step2, we only need to find the 2 medians in sorted array and compare them, which both actions takes only O(1) time. So the time complexity can be represented by $T(m) = T(m/2) + O(1)$. From the master theorem, we can easily figure out that the time complexity $T(m) = \Theta(\log m) = \Theta(\log n)$. □