CS 3510: Design & Analysis of Algorithms (Sections A & B)

Homework 2: Solving Recurrences, Divide & Conquer

Professors Dana Randall and Gerandy Brito **Due: 9/13/2022 11:59pm**

- **Your solutions must be in plain English and mathematical expressions. For every problem, you have to explain your design and why it is correct in your own words, and you have to analyze and justify the running time using the Master Theorem (where applicable).**
- **Unless otherwise stated, fastest (and correct) algorithms will receive more credit. If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.**
- **Do not use pseudocode. Your answer will receive zero credit even if the pseudocode is correct.**

Name: Anthony Wong

1.) (5 points each) Solve the following recurrences using the Master Theorem. You must identify the values of a, b, and d, and state which case you are using. All answers must be in big-O notation.

(a.) $T(n) = 3T(\frac{n}{2}) + n$

Solution:

a = 3
b = 2
d = 1
Because $a > b^d$ $(3 > 2^1)$, we use the case $O(n^{\log_b a})$. It follows that $T(n) = O(n^{\log_2 3})$.

(b.) $T(n) = 4T(\frac{n}{2}) + 7n^2$

Solution:

a = 4
b = 2
d = 2
Because $a = b^d$ $(4 = 2^2)$, we use the case $O(n^d \log n)$. It follows that $T(n) = O(n^2 \log n)$.

(c.) $T(n) = T(\frac{n}{3}) + 25$

Solution:

a = 1
b = 3
d = 0
Because $a = b^d$ $(1 = 3^0)$, we use the case $O(n^d \log n)$. It follows that $T(n) = O(\log n)$.

(d.) $T(n) = 5T(\frac{n}{5}) + \frac{n}{5}$

Solution:

a = 5
b = 5
d = 1
Because $a = b^d$ $(5 = 5^1)$, we use the case $O(n^d \log n)$. It follows that $T(n) = O(n \log n)$.

(e.) $T(n) = 5T(\frac{n}{2}) + 4n^2$

Solution:

a = 5
b = 2
d = 2
Because $a > b^d$ $(5 > 2^2)$, we use the case $O(n^{\log_b a})$. It follows that $T(n) = O(n^{\log_2 5})$.

(f.) $T(n) = 8T(\frac{n}{3}) + 2n^2$

Solution:

a = 8
b = 3
d = 2
Because $a < b^d$ $(8 < 3^2)$, we use the case $O(n^d)$. It follows that $T(n) = O(n^2)$.

2.) (15 points each) Suppose we have an array $A = [a, b, c, ..., y]$ of 25 integers. In the process of finding the median of medians, we form the following 5 subarrays, each of which is sorted:

$$[d, c, b, e, a]$$
$$[i, f, h, j, g]$$
$$[l, m, k, n, o]$$
$$[p, s, q, r, t]$$
$$[w, x, v, u, y]$$

We take the medians from the 5 subarrays (i.e. b, h, k, q, and v), and we collect them into a new sorted array $[k, h, q, b, v]$. Since the median of this new sorted array is q, we can say the median of medians is q.

(a.) Which elements of the original array A are guaranteed to be less than or equal to q? List them all and briefly justify your answer.

Solution:

We first need to rearrange the 5 sorted subarrays to follow the order of the sorted medians (The line with k is first, followed by h, etc.). The middle column of the new array matrix are the medians, and due to the order of the medians stated in the problem, we know the letters directly above q are less than or equal to q. Because the individual arrays are also sorted, any letters to the left of q and its lesser medians are less than or equal to q.
List = $[l, m, k, i, f, h, p, s]$

(b.) Which elements of the original array A are guaranteed to be greater than or equal to q? List them all and briefly justify your answer.

Solution:

We first need to rearrange the 5 sorted subarrays to follow the order of the sorted medians (The line with k is first, followed by h, etc.). The middle column of the new array matrix are the medians, and due to the order of the medians stated in the problem, we know the letters directly below q are greater than or equal to q. Because the individual arrays are also sorted, any letters to the right of q and its greater medians are greater than or equal to q.
List = $[r, t, b, e, a, v, u, y]$

(c.) Is q a good pivot? Justify your answer.

Solution:

We can conclude that q is a good pivot. By looking at both problems 2a and 2b, we can see that in the worst case when we pick q as a pivot, the largest possible subproblem is 17/25 in size. Pessimistically, we will say 3/4. This means that the time complexity of our subproblem is $T(n) = T(3n/4) + O(n)$. Using the Master Theorem, we can see that $T(n) = O(n)$. The linearity of the time complexity proves q is a good pivot.

3.) (15 points each) Suppose we have an array, A, containing n *distinct* integers. We are also given as input, a lower bound l and an upper bound u. Provide a Divide & Conquer algorithm to find the number of elements in A between l and u, inclusive, under each of the following conditions. Make sure to provide reasoning for your algorithm and give the time complexity with proof (using the Master Theorem).

(a.) Condition 1: A is an unsorted array

Solution:

One Divide & Conquer algorithm that would work in this case is similar to the dividing phase in Merge Sort where we continually split our array in half until we split into arrays of length one, which is our base case. Once we reach our base case we can check whether the number is within the defined lower and upper bounds. Tallying these cases will give us the number of elements in A between l and u. The time complexity can be defined as $T(n) = 2T(n/2) + O(1)$. Using the Master Theorem, this can be rewritten as $T(n) = O(n^{\log_2 2}) = O(n)$.

(b.) Condition 2: A is a sorted array

Solution:

Given that A is sorted and has n distinct integers we can instead run binary search twice, searching for both l and u. In the event of an index tie during the splitting process, we choose the left element. First, we search for l. There are two cases, one is if l is in A and one if it is not. If l is in A, we simply return the index of l. If it is not, we search until we reach an element less than l and return the incremented index of that element. Then we search for u which has the same two cases. If u is in A, we return the incremented index of u. If it is not, we search until we reach an element greater than u and return that index. Once we have the two indices we can subtract them and get the the number of elements in A between l and u. The time complexity can be written as $T(n) = 2(T(n/2) + O(1))$. Using the Master Theorem, this can be rewritten as $O(2n^0 \log n) = O(2 \log n) = O(\log n)$.