CSCI104 Homework 5

Yang Li (yli546@usc.edu)

March 21, 2018

1 Problem 1c (Break QuickSort)

One example of such array would be `5.5, 6.2, 9.3, 11.6, 13.4, 7.1, 8.2, 10.9, 12.7, 14.1`. The calltree is shown in the `calltree_qsort.jpg`.

2 Problem 2 (Break Interpolation Search)

The worst case of interpolation search would happen when the later element is at least twice or more times than the element ahead, and the element we are searching is penultimate one. For example, a given double array of `2.0, 4.0, 8.0, 16.0, 3200.0` and we are searching for `16.0` in this case. The calltree is shown in the attached `calltree_interpolation.jpg` file and interpolation search code can be found in `interpolation.cpp` file. Throughout the interpolation search, the first and last elements have been visited when condition `a[r] == a[l]` is called, and each recursive call would move `int l` to the next entry till it finds the element `16.0`.

3 Problem 3 (Median Finding in Linear Time)

Part a (Set up a Recurrence Relation)

1. For the `if` condition part, which is `if (right <= left + 10)`, insertion sort would take $\Theta(n^2)$. However, since the size of array for `InsertionSort()` is always smaller than or equal to 10, this part could be considered as running in $\Theta(1)$ and return statement would also take constant running time so that if `if` statement is called, it would take $\Theta(1)$ running time. For the `else` condition part, everything except recursive calls, dynamic allocation, `linearSearch()`, and `partition()` would only take constant running time, while dynamic allocation would take $\Theta(n/5) = \Theta(n)$ because the size of `smallsize` is $n/5$, `linearSearch()` and `partition()` would take $\Theta(n)$ running time. Therefore, if the `else` statement is called, it would take $\Theta(n)$ running time.

2. The input size for `b[i]` is 5 because the size of array, which is `[left+5*i] - [left+5*(i+1)-1]+1`, is `5`. Therefore, in the loop that assigns values to each `b[i]`, `quantile()` function is supposed to run in $\Theta(n^2) * (n/5) = \Theta(n^3)$ time, because when the input size is 5, which is smaller than 10, `InsertionSort()`, which has a running time of $\Theta(n^2)$, would be called. However, `InsertionSort()` can be considered as running in constant time in this case because the size of array for `InsertionSort()` would always be smaller than or equal to 5. In this way, the recursive calls to `quantile()` take about constant running time, which is $\Theta(1)$. As `smallsize = (right-left)/5 = n/5`, there would be $n/5$ recursive calls. In order to calculate the total running time, $T(n) = \Theta(1) * (n/5) = \Theta(n/5) = \Theta(n)$.

3. The size of array would be `smallsize-1-0+1 = (right - left)-1-0+1= (right - left) = n/5`. Therefore, this recursive call would take $T(n/5)$.

4. Of the `b` array, which is of size $n/5$, half the number of groups of $n/10$ have their median less than the pivot. Also, another half the number of groups, which is also of size $n/10$, have their median greater than the pivot. In each of the $n/10$ groups with median less than the pivot, there are two elements that are smaller than their respective medians, which are smaller than the pivot. Thus, each of the $n/10$ groups have at least 3 elements that are smaller than the pivot. Similarly, in each of the $n/10$ groups with median greater than pivot, there are two elements that are greater than their respective medians, which are greater than the pivot. Thus, each of the $n/10$ groups have at least 3 elements that are greater than the pivot. Hence, the pivot is less than $3 * (n/10) = 3n/10$ elements and greater than another $3 * (n/10) = 3n/10$ elements.

5. As the chosen pivot splits the elements somewhere between $3n/10$ and $7n/10$, so that in the worst case, the size of array for the recursive call would be of $7n/10$.

6. The base case for the `quantile()` function takes place when `(right-left)` is smaller than or equal to 10. In this way, the total running time can be expressed as: $T(n) = \Theta(1)$. Otherwise, the total running time, which is expressed as $T(n) = T(n/5) + T(7n/10) + \Theta(n)$ can be divided into three parts.

    - The first part, which is $T(n/5) = T(n * 2/10)$ is for finding the median of $n/5$ medians, which is spcifically expressed as `T pivot = quantile(b, 0, smallsize-1, smallsize/2)` in the code.
    - The second part, which is $\Theta(n)$, is for the partitioning work to create the two sides, one of which our `quantile()` function will recurse (we visited each element a constant number of times, in order to form them into $n/5$ groups and take each median in $O(1)$ time).

- The third part, which is $T(7n/10)$, is the part for actual `quantile()` recursion (for the worst case, in which the k-th element is in the bigger partition that can be of size $7n/10$ maximally).

Part b (Solve the Recurrence)

- The calltree for this part is shown in the attached `calltree_medianfind.jpg`.
- 4. Each item in the calltree would generate two terms in the next layer, one with coefficient 1/5 and the other with coefficient 7/10. Therefore, the sum of work done in the 2nd layer could be represented as:

$$\left(\frac{1}{5}\frac{1}{5}+\frac{7}{10}\frac{1}{5}+\frac{7}{10}\frac{1}{5}+\frac{7}{10}\frac{7}{10}\right)*\Theta(n)=\left(\frac{1}{5}+\frac{7}{10}\right)^2\Theta(n)$$

and the sum of work done in the 3rd later could be represented as:

$$\left(\frac{1}{5}\frac{1}{5}\frac{1}{5}+\frac{7}{10}\frac{1}{5}\frac{1}{5}+\frac{1}{5}\frac{7}{10}\frac{1}{5}+\frac{7}{10}\frac{7}{10}\frac{1}{5}+\frac{1}{5}\frac{1}{5}\frac{7}{10}+\frac{7}{10}\frac{1}{5}\frac{7}{10}+\frac{1}{5}\frac{7}{10}\frac{7}{10}+\frac{7}{10}\frac{7}{10}\frac{7}{10}\right)*\Theta(n)=\left(\frac{1}{5}+\frac{7}{10}\right)^3\Theta(n)$$

Therefore, the sum of work done in the ith layer could be represented as $(\frac{1}{5}+\frac{7}{10})^i\Theta(n)$.

- 5. From the `calltree_medianfind.jpg`, it could be supposed that the number of layers is x. Therefore, as the calltree for `quantile()` function is shown as a binary tree, which is similar to the calltee of `mergesort()`, the formula can be represented as:

$$\left(\frac{7}{10}\right)^x*n=10$$

$$\left(\frac{7}{10}\right)^x*n=10$$

$$\left(\frac{7}{10}\right)^x=\frac{10}{n}$$

$$x=\log_{\frac{7}{10}}\left(\frac{10}{n}\right)=\log_{\frac{10}{7}}\left(\frac{n}{10}\right)\approx\log n$$

Therefore, there would be approximately $\log n$ layers.

- 6. As n can be considered as an imaginary number that approaches to infinity, the total running time could be represented as:

$$\sum_{i=0}^{\log n}\left(\frac{1}{5}+\frac{7}{10}\right)^i\Theta(n)=\Theta(n)*\sum_{i=0}^{\log n}\left(\frac{9}{10}\right)^i$$

It can be observed that the series on the right is a geometric series with a common ratio $r = \frac{9}{10} < 1$, so that the sum of this geometric series is

$$\sum_{i=0}^{\log n} \left(\frac{1}{5}+\frac{7}{10}\right)^i \Theta(n) = \frac{(\frac{1}{5}+\frac{7}{10})^0 * (1-\frac{9}{10})^{\log n}}{1-\frac{9}{10}} * \Theta(n) = \frac{1 * (\frac{1}{10})^{\log n}}{\frac{1}{10}} * \Theta(n)$$

As n approaches to infinity, the coefficient of $\Theta(n)$ would approach to 10. Therefore, the total running time would be $\Theta(n)$.