Homework #3
Introduction to Algorithms/Algorithms 1
600.463
Spring 2016

Submitted By: Sindhuula Selvaraju
JHED Id: sselvar4

February 19, 2016

1 Problem 1 (13 points)

Let's say that a pivot provides $x|n-x$ separation if x elements in an array are smaller than the pivot, and $n-x$ elements are larger than the pivot.

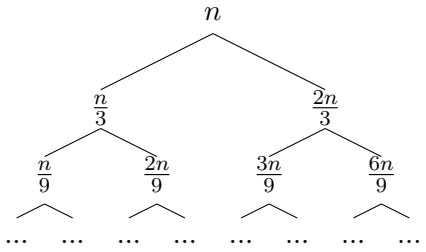
Suppose Bob knows a secret way to find a good pivot with $\frac{n}{3}|\frac{2n}{3}$ separation in constant time. But at the same time, Alice knows her own secret technique, which provides $\frac{n}{4}|\frac{3n}{4}$ separation and works in constant time.

Alice and Bob applied their secret techniques as a subroutine in QuickSort algorithm. Whose algorithm works **asymptotically** faster? Prove your statement.
ANSWER:
NOTE:
We know(given in CLRS page: 175) that the worst case partitioning of Quicksort is when the partitioning is most unbalanced for example in a case where the array is already sorted resulting in one subproblem with $n-1$ elements and the other with 0 elements. In this case the running time of the algorithm is $\theta(n^2)$. We also know that the best case partitioning for Quicksort is when the partitioning results in subproblems each of size no more then $n/2$. In this case the running time of the algorithm is $\theta(n \lg n)$
MAIN ANSWER:
Since Bob finds a pivot with $\frac{n}{3}|\frac{2n}{3}$ separation in constant time and Alice finds a

pivot with $\frac{n}{4}|\frac{3n}{4}$ separation in constant time, both these cases result in a case between the Best and Worst case scenarios as noted above. Hence, both belong to the average case scenario.

Let us first analyse Bob's algorithm:

In Bob's algorithm everytime we would result in subproblems with one side having $n/3$ elements and the other side having $2n/3$ elements.



Each level of the problem takes atmost cn partitioning time. The left child of each node is $\frac{1}{3}$ times the size of the parent and the right child is $\frac{2}{3}$ times the size of the parent. Since the smaller subproblems are on the left, by following a path of left children, we get from the root down to a subproblem size of 1 faster than along any other path. This happens after $\log_3 n$ levels. Similarly to reach a subproblem of size 1 down the right sub-tree it will take $\log_{\frac{3}{2}} n$ levels.

Since till the left sub-tree reaches a size of 1 there are n nodes the partitioning time for each level is cn. But once the left sub-tree has finished partitioning(i.e. all the elements there are now sorted) the right sub-tree still has some levels to complete but not the time will be atmost cn since the number of elements has reduced and will be O($n \log_{\frac{3}{2}} n$). $\log_{\frac{3}{2}} n$ differs for $\lg n$ only by a factor of $\lg \frac{3}{2}$ which is a constant time. So the running time of this algorithm is O($n \lg n$)

Similarly if we analyse Alice's algorithm:

The subproblems will have $\frac{n}{4}$ elements on one side and $\frac{3n}{4}$ elements on the other side. If we do a similar analysis as with Bob's algorithms we will see that to reach the size of 1 in the left subtree it take $\log_4 n$ levels and along the right subtree it will take $\log_{\frac{4}{3}} n$ levels.

Till the left subtree reaches a size of 1 the partitioning time will be cn and after that it will be O($n \log_{\frac{4}{3}} n$). $\log_{\frac{4}{3}} n$ differs for $\lg n$ only by a factor of $\lg \frac{4}{3}$ which is a constant time. So the running time of this algorithm is O($n \lg n$)

Bob's algorithm is aymptotically faster than Alice's algorithm if the constants in partitioning time are considered. But since the constants are ignored the overall running time for both algorithms is the same.

2 Problem 2 (13 points)

Alice and Bob are solving a problem. They are given a matrix A_n of size $n \times n$, where elements are sorted along every column and every row. For example, consider the case when $n = 4$ and matrix A_4 is as following:

$$A_4 = \begin{bmatrix} 3 & 5 & 9 & 12 \\ 4 & 8 & 10 & 32 \\ 12 & 13 & 29 & 43 \\ 16 & 19 & 30 & 60 \end{bmatrix}$$

They need to provide an algorithm which takes integer x as an input, and checks if x is an element of matrix A_n or not. Both Alice and Bob decided to apply divide and conquer method.
Alice's algorithm is quite simple, she uses a binary search routine to check each row for the input integer x.
Bob's algorithm is more advanced. Bob found out that if matrix A_n is represented in block-view:

$$A_n = \left[\begin{array}{c|c} A^1 & A^2 \\ \hline A^3 & A^4 \end{array}\right],$$

where all matrices A^1, A^2, A^3, A^4 are of the size $\frac{n}{2} \times \frac{n}{2}$, then he can compare element x with $A^4[1,1]$ and consider only three options:

1. $x = A^4[1,1]$ then his algorithm can output "yes".
2. $x < A^4[1,1]$ then he knows that $\forall i, j < n/2 : \quad x < A^4[i,j]$, thus he knows for sure that $x \notin A^4$, and only need to recursively apply his algorithm to check if $x \in A^1$, $x \in A^2$ or $x \in A^3$.
3. $x > A^4[1,1]$ then he knows that $\forall i, j < n/2 : \quad x > A^1[i,j]$, thus he knows for sure that $x \notin A^1$, and only need to recursively apply his algorithm to check if $x \in A^2$, $x \in A^3$ or $x \in A^4$.

Whose algorithm is faster on the worst case input? Prove your statement.

ANSWER: Since Alice performs binary search on each of the rows:
Each row is of size n and binary search will take O($\log$ n) time.
There are n rows so to search through the entire matrix it will take O(n$\log$n) time.
$\Rightarrow Alice's algorithm takes O(n\log$ n) time.
Bob's Algorithm tries to find x by using divide and conquer.
For a n = 1 matrix T(n) = 1
For each T(n) the time taken will be:

1. O(c) time step to compare x = $A^4[1,1]$

2. 3T(n/2) time to find if any of the other quadrants have x

So, T(n) = 3T($\frac{n}{2}$) + 1
Applying Master Theorem:
This equation is of the form aT($\frac{n}{b}$)+f(n)
Here a = 3;b=2 and f(n) = 1
n$^{(}\log_2 3) = n^{1.58}$
Clearly f(n) = $O(n^{1.58-\epsilon})$ where epsilon = 1.58
This corresponds to case 1 of master theorem and so:
T(n) = $\theta(n^{1.58})$
Comparing Alice and Bob's algorithms:
Since O(n $\log$n)<($\theta(n^{1.58})$)
Alice's algorithm performs better than Bob's

3 Problem 3 (24 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. $T(n) = 2T(n/8) + n^{\frac{1}{5}} \log n \log\log n$

   ANSWER: Using Master Theorem the given equation is of the form:
   $T(n) = aT(n/b) + f(n)$
   Where $a = 2, b = 8$ and $f(n) = n^{\frac{1}{5}} \log n \log\log n$
   Calculating : $n^{log_b a} = n^{log_8 2} = n^{1/3} = n^{0.3333}$
   Since $f(n) = n^{\frac{1}{5}} \log n \log\log n$
   The value of $f(n) = O(n^{1/5+\epsilon})O(n^{0.3333-\epsilon})$
   $n^{log_a b} > f(n)$
   This corresponds to case 1 of Master Theorem
   $\implies T(n) = \Theta(n^{log_a b}) = \Theta(n^{1/3})$

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

   ANSWER: Using Master Theorem the given equation is of the form:
   $T(n) = aT(n/b) + f(n)$
   Where $a = 16, b = 2$ and $f(n) = \sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i})$
   Calculating: $n^{log_b a} = n^{log_2 16} = n^{log_2 2^4} = n^4$
   Since $f(n) = \sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i})$ is an example of Taylor Series, we can write it as(This is similar to Homework 1 question 1.2):
   Note that : $1 < \ln(e + \frac{1}{i}) < 2$

   $$\sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i}) > \sum_{i=\frac{n}{2}}^{n} i^3 > \sum_{i=\frac{n}{2}}^{n} (\frac{n}{2})^3 = (\frac{n}{3})^4$$

   Thus f(n) = $\Omega(n^4)$

   $$\sum_{i=1}^{n} i^3 \ln(e + \frac{1}{i}) < 2 \sum_{i=\frac{n}{2}}^{n} n^3 < 2n^4$$

   Thus f(n) = O(n^4)
   Thus f(n) = $\Theta(n^4)$
   Thus $n^{\log_a b} = f(n)$ which corresponds to case 2 of Master Theorem

T(n) = $\Theta(n^{\log_b a} \lg n)$=$\Theta(n^4 \lg n)$

3. $T(n) = 4T(n-2) + 2^{2n}$
   ANSWER:
   $T(n) = 4T(n-2) + 2^{2n}$
   $T(n) = 4T(n-2) + 4^n$
   $T(n) = 4(4T(n-4) + 4^{(n-2)}) + 4^n$
   $T(n) = 4^2T(n-4) + 4^{(n-1)} + 4^n$
   $T(n) = 4^2(4T(n-6) + 4^{(n-4)}) + 4^{(n-1)} + 4^n$
   $T(n) = 4^3T(n-6) + 4^{(n-2)} + 4^{(n-1)} + 4^n$

$$T(n) = 4^{(k-1)}T(n-2-k) + \sum_{m=0}^{k-2} 4^{n-m}$$

   Let $k = n-3$

$$T(n) = 4^{(n-4)}T(1) + \sum_{m=0}^{n-5} 4^{n-m}$$

   Putting T(1) = 1

$$T(n) = 4^{(n-4)} + \sum_{m=0}^{n-5} 4^{n-m}$$

   $T(n) = 4^{(n-4)} + \frac{4}{3}(4^n - 4^4)$
   $T(n) = 4^{(n-4)} + \frac{4^{(n+1)} - 4^5}{3}$
   $T(n) = 2^{2(n-4)} + \frac{2^{2(n+1)} - 2^{10}}{3}$
   $T(n) = O(2^{2n})$

4. $T(n) = T(n/3) + n^3 \log n + 2n^2 - \sqrt{\log(n+1)}$
   ANSWER:
   Using Master Theorem the given equation is of the form:
   $T(n) = aT(n/b) + f(n)$
   Where a = 1, b = 3, f(n) = $n^3 \log n + 2n^2 - \sqrt{\log(n+1)}$
   $n^{\log_b a} = n^{\log_3 1 = n^0 = 1}$
   Clearly f(n) > 1 and is of the form f(n) = $n^{\log_b a + \epsilon}$
   The corresponds to case 3 of Master Theorem
   Thus, T(n) = $\Theta f(n) = \Theta(n^3 \log n + 2n^2 - \sqrt{\log(n+1)}) = \Theta(n^3 \log n)$

5. $T(n) = 8T(n/2) + n^3 - 8n \log n$
   ANSWER:
   Using Master Theorem the given equation is of the form:
   $T(n) = aT(n/b) + f(n)$
   Where a = 8, b = 2, f(n) = $n^3 - 8n \log n$
   $n^{\log_b a} = n^{\log_2 8} = n^3$
   f(n) = $n^3 - 8n \log n = \Theta(n^3)$
   This is case 2 of master theorem
   Hence, T(n) = $\Theta(n^3 \lg n)$

6. $T(n) = T(n/2) + \log n$
   ANSWER:
   $T(n) = T(\frac{n}{2}) + \log n$
   $T(n) = T(\frac{n}{4}) + \log(\frac{n}{2}) + \log n$
   $T(n) = T(\frac{n}{2^2}) + \log(\frac{n}{2}) + \log n$
   $T(n) = T(\frac{n}{2^3}) + \log(\frac{n}{2^2}) + \log(\frac{n}{2}) + \log n$

$$T(n) = T(\frac{n}{2^{i+1}}) + \sum_{m=0}^{i} \log(\frac{n}{2^m})$$

   Let $2^{i+1} = n$ and $i = \log_2 n$
   We know T(1) = 1

$$T(n) = T(1) + \sum_{m=0}^{\log_2 n - 1} \log(\frac{n}{2^m})$$

$$T(n) = 1 + \sum_{m=0}^{\log_2 n - 1} \log(n) - \log(2^m)$$

$$T(n) = 1 + \sum_{m=0}^{\log_2 n - 1} \log(n) - m.\log(2)$$

$$T(n) = 1 + \sum_{m=0}^{\log_2 n - 1} \log(n) - m$$

$$T(n) = 1 + (\log_2 n - 1).(\log(n)) - \frac{(\log_2 n - 1)(\log_2 n)}{2}$$

Since $\log n = \log_2 n$

$$T(n) = 1 + (\log n - 1).(\log(n)) - \frac{(\log n - 1)(\log n)}{2}$$

$$T(n) = 1 + \frac{(\log n - 1)(\log n)}{2}$$

$$T(n) = 1 + \frac{((\log n)^2 - \log n)}{2}$$

$$T(n) = O((\log n)^2)$$

$$T(n) = O(\log^2 n)$$

7. $T(n) = T(n-1) + T(n-2)$
   ANSWER:
   Given: T(1) = 1
   Assuming: T(1) = 1
   $T(n) = T(n-1) + T(n-2)$
   $T(3) = T(3-1) + T(3-2)$
   $T(3) = T(2) + T(1)$
   Since both T(n-1) and T(n-2) have approximately the same value. We can have:
   $T(n) = 2T(n-1)$
   $T(n) = 2(2T(n-2))$
   $T(n) = 2^2T(n-2)$
   $T(n) = 2^3T(n-3)$
   $T(n) = 2^(k)T(n-k)$
   Let k = n-1
   $T(n) = 2^(n-1)T(1)$
   $T(n) = 2^(n-1)$
   $T(n) = O(2^n)$

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

$T(n) = 3T(n^{\frac{2}{3}}) + \log n$
Assume $n = a^k \Rightarrow k = \log_a n$
$T(a^k) = 3T(a^{\frac{2k}{3}}) + \log(a^k)$
Let $T(a^k) = A(k)$
$A(k) = 3A(\frac{2k}{3}) + \log(k)$
$A(k) = 3A(\frac{k}{\frac{3}{2}}) + \log(k)$
Applying Master Theorem:
This equation is of the form aT($\frac{n}{b}$)+f(n)
Here a = 3;$b = \frac{3}{2}$ and $f(k) = \log(k)$
$k^{\log_b a} = k^{\log_{\frac{3}{2}} 3} = k^{2.71}$
$f(k) < k^2.71$
$f(k)$ is of the form $k^{\log_b a - \epsilon}$
$A(k) = \Theta(k^{\log_{\frac{3}{2}} 3})$
$T(n) = \Theta((\log_a n)^{\log_{\frac{3}{2}} 3})$