Assignment 3

Vivek Bansal (Id : 112044493)
CSE 548 : Analysis of Algorithms

Due Date : October 22, 2018
Discussed with : Ayushi Srivastava

1 Bank Equivalence Tester

We have to find out if there are more than n/2 cards which are equivalent in the whole set. We will be using divide and conquer strategy to solve this problem.

Algorithm :

1) If length of set is equal to 1, return that card.
2) If length of set is equal to 2, check their equivalency. If they are equivalent, return any one of them. If they are not equivalent discard both of them.
3) Else, divide the whole set into 2 equal piles of size n/2 each. Let the sets be S1 and S2. This is the divide step.
4) During conquer phase, merge the 2 sets : S1 and S2. If S1 and S2 both returned the same majority element, then it is the majority equivalent for the whole set as well. If only one of the set from S1 and S2 returned the majority element, then check its equivalency with all cards in both parts in O(n) time. If S1 and S2 both returned the majority elements but they are different, still we can check their equivalency in 2n operations.

Correctness:
If a majority equivalent card with value v is there , then the majority equivalent property also exists for at least one side. If more than n/2 cards are equivalent in the set, then at least n/4 cards with value val are on at least one side. So at least one side will return a card which is equivalent on calling it recursively and we will be able to find our equivalent card at the final merge phase of the algorithm.

Running time:
Let T(n) be the maximum comparisons done for n cards. It has two recursive calls with each side of length n/2 and it does at most 2n tests outside the recursive calls.

$$T(n) = 2T(n/2) + 2n$$

We have a = 2, b = 2, k = 1. So with master's theorem we have,

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

2 Hidden Surface Removal Problem

Approach : Let us take the case of 3 lines. We arranged these lines in order of their increasing slopes. Let these lines be L1, L2 and L3 in order of their slopes. L1 and L3 lines are always visible. There are 2 cases possible:
1) Only 2 lines (L1 and L3) are visible.
2) All 3 lines (L1, L2, L3) are visible.
To check middle line L2 is visible or not, we will check the intersection point of other 2 lines. If that intersection point lies above this middle line, then middle line L2 is hidden. If intersection point lies below this line, then middle line L2 is visible. So we can check for 3 lines visibility in constant time.

Algorithm:
1) Base case : If there are only 3 lines, then we can check using the above method.
2) Let's say we are having n lines.
3) In divide step, we divide these n lines into 2 parts of size n/2 each.
4) Call the algorithm recursively on the first n/2 part to find the visible lines and also the adjacent lines intersection points. These intersection points will be in the order of increasing x-coordinates.
5) Now we got the visible lines in both the 2 parts. Let us call the 2 parts as S1 and S2. During the merge step, we will find the line L1 in S1 which is uppermost in S1 and line L2 in S2 which is uppermost in S2 at each x-coordinate. This step will take O(n) time. Let m be the smallest index such that line L2 is uppermost to line L1. We will compute their intersection points (x,y), which implies that L1 is uppermost to the immediate left of x and L2 is uppermost to the immediate right of x. So in this manner we will compute the sequence of visible lines and intersection points while merging and algorithm continues to the next level of recursion which justifies the correctness of the algorithm as well.

Running time:

$$T(n) = 2T(n/2) + n$$
$$T(n) = O(n \log n)$$

3 Local Minima in a complete binary tree

Algorithm :

1) First check the value of root with its 2 childs. If value at root is less than its 2 childs, then return the root value as it is a local minima.
2) Else if value at root is more than any one of its childs, then move to the child which has smaller value.
3) Return that node value which has smaller value than both of its childs.
4) If we will reach a leaf node, then return the value of leaf node as the local minima.

Correctness:
There are 3 possible cases :
1) If the root node is returned as the local minimum, then it is smaller than its both children.

2) If any other child node is returned other than the leaf node, then it is correct since it is smaller than its parent and both of its children.
3) If leaf node is returned , then it is correct because it is smaller than its parent and it is the smallest value along a path.

Time complexity = $O(\log n)$ since we are traversing one path in a complete binary tree whose length is $\log n$.

4 Solving Recurrence Equations

Ans a.

$$A(n) = 4A(\lfloor (n/2) \rfloor + 5) + n^2$$

Since the value of $\lfloor (n/2) \rfloor$ is always less than or equal to n/2, we have:

$$A(n) \leq 4A((n/2) + 5) + n^2$$

Let us suppose we have:

$$S(n) = A(n + \alpha)$$

So $S(n)$ will follow the recurrence relation :

$$S(n) \leq 4S(n/2) + O(n^2)$$

Therefore using $S(n) = A(n + \alpha)$ in the above equation we get,

$$S(n) \leq 4S(n/2) + O(n^2) \implies A(n + \alpha) \leq 4A((n/2) + \alpha) + O(n^2)$$
$$\text{A(n)} \leq 4A((n/2) + 5) + O(n^2) \implies A(n + \alpha) \leq 4A((n + \alpha)/2) + 5) + (n + \alpha)^2$$

For the above two recurrences to be equal, we should have :

$$n/2 + \alpha = (n + \alpha)/2 + 5$$

$$n + 2\alpha = n + \alpha + 10$$

Hence, $\alpha = 10$

$$S(n) = A(n + \alpha)$$
$$S(n) = A(n + 10)$$
$$S(n - 10) = A(n)$$

Using Master's Theorem,we have:

$$f(n) = n^2, a = 4, b = 2$$
$$af(n/b) = (4/4) * n^2$$

$$af(n/b) = n^2$$

So we have k = 1

Now according to Master's theorem,

$$S(n) = f(n)logn$$
$$S(n) = O(n^2logn)$$

Substituting value $S(n) = A(n + \alpha)$

$$A(n) = O((n-10)^2log(n-10))$$
$$A(n) = O(n^2log(n))$$

Ans b.

$$B(n) = B(n-4) + \frac{1}{n} + \frac{5}{n^2+6} + \frac{7n^2}{3n^3+8}$$

It is clear that both terms

$$\frac{7n^2}{3n^3+8} \quad , \quad \frac{5}{n^2+6}$$

are bounded by $O(1/n)$.

Therefore we have,

$$B(n) = B(n-4) + O(\frac{1}{n})$$

Solve the above recurrence equation by substitution in the recursion tree as below,

$$B(n) = B(n-4) + \frac{1}{n}$$
$$B(n-4) = B(n-8) + \frac{1}{n-4}$$
$$.....$$
$$.....$$
$$.....$$
$$B(n-4(k-1)) = B(n-4k) + \frac{1}{n-4k}$$

On adding all the above equations together and cancelling the common terms we get,

$$B(n) = B(1) + \sum_{k=0}^{n/4} \frac{1}{n-4k}$$

$$\implies B(n) = B(1) + \frac{1}{4} + \frac{1}{8} + \frac{1}{12} .... \frac{1}{4n/4}$$

Take 1/4 common from the above terms, we get

$$\implies B(n) = O(1) + \frac{1}{4}(1 + \frac{1}{2} + \frac{1}{3}....\frac{1}{n/4})$$

Since,

$$\sum_{k=1}^{n/4} \frac{1}{k} = \log(n/4)$$

We get,

$$B(n) = O(1) + O(\log n)$$
$$B(n) = O(\log n)$$

Ans c.

$$C(n) = n + 2\sqrt{n}C(\sqrt{n}) \qquad -(1)$$

Dividing above equation (1) by n, we get:

$$\frac{C(n)}{n} = 1 + 2(\frac{C(\sqrt{n})}{\sqrt{n}}) \qquad -(2)$$

Let n = 2^m

Equation (2) becomes :

$$\frac{C(2^m)}{2^m} = 1 + 2(\frac{C(2^{m/2})}{2^{m/2}}) \qquad -(3)$$

Let $\frac{C(2^m)}{2^m} = S(m)$, so we have :

$$S(m) = 2S(m/2) + 1 \qquad -(4)$$

Applying master's theorem with a = 2, b = 2, k = 2, we get :

$$S(m) = O(m) \qquad -(5)$$

Now, we have:

$$\frac{C(n)}{n} = O(m) \qquad -(6)$$

But we have m = $\log n$, so we got :

$$\frac{C(n)}{n} = O(\log n)$$
$$C(n) = O(n \log n)$$

5 Matrices Problems

a) We are given,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

After squaring this we have,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a^2 + bc & ab + bd \\ ca + dc & cb + d^2 \end{bmatrix}$$

which is equal to:

$$\begin{bmatrix} a^2 + bc & b(a + d) \\ c(a + d) & cb + d^2 \end{bmatrix}$$

So we have to do 5 multiplications : a^2, bc, d^2, c(a + d), b(a + d) to compute the square of matrix A of size 2 x 2.

b) We are given,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

After squaring this we have,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a^2 + bc & ab + bd \\ ca + dc & cb + d^2 \end{bmatrix}$$

which is not equal to:

$$\begin{bmatrix} a^2 + bc & b(a + d) \\ c(a + d) & cb + d^2 \end{bmatrix}$$

Matrix multiplication is not commutative, i.e ab $\neq$ ba. Also ab + bd $\neq$ b(a + d).
So we still have to solve 7 problems from Strassen Matrix multiplication to get the square of matrix A.
Hence the recurrence, T(n) = 5T(n/2) + O(n^2) is not correct .
Hence , the analysis that the running time is O($n^{(\log_2 5)}$) is not correct.
We will get the running time equation as T(n) = 7T(n/2) + O(n^2) to compute the square of matrix A of size n x n when we will applying Strassen matrix multiplication.

c)
i) We can calculate AB + BA by using the square formula:
AB + BA = $(A + B)^2 - A.A - B.B$
As we can see squaring of matrices is done in total 3 times . Therefore 3S(n) time and the addition operations will take $O(n^2)$.
So, this will take total of $3S(n) + O(n^2)$ time.

ii) The given matrices are :

$$A = \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}$$

$$AB = \begin{bmatrix} 0 & XY \\ 0 & 0 \end{bmatrix}$$

$$BA = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$AB + BA = \begin{bmatrix} 0 & XY \\ 0 & 0 \end{bmatrix}$$

iii)

$$AB + BA = \begin{bmatrix} 0 & XY \\ 0 & 0 \end{bmatrix}$$

as deduced from the part (i).

So, XY is basically calculating AB + BA. XY is an n*n matrix and AB + BA is 2n * 2n matrix. We have already proved in i) part, that calculating AB + BA will take $3S(n) + O(n^2)$ time.
Thus, XY is calculating AB + BA only, so it will take $3S(2n) + O(n^2)$ time.