311 Home Work 3

Ben purdy

March 12, 2020

(1)

Give an $O(logm + logn)$ -time algorithm that takes two sorted lists of sizes m and n, respectively, as input and returns the ith smallest element in the union of the two lists. Justify your algorithm and analyze its running time

Algorithm 1: ith smallest element

```
initialization(Array a, Array b, k);
a_index ← k/2 - 1;
b_index ← k - a_index-2;
count ← k/4;
while count < k do
    if i = k || j = k then
        break loop
    if a[i+1] ≤ b[j] then
        i ← i + 1;
        j ← j - 1;
    else
        i ← i - 1;
        j ← j + 1;
    count ← count/2;
if a[i] > b[j] then
    return a[i];
else
    return b[j];
```

- *Proof.* NTS log(mn)
  suppose you have an arrays A,B where A is size n and B is size m. Then in every induration the indices of A,B are touched and the max number of loops is k/4 means that only a subsection of the nodes in A,B are touched. This reduction in interactions makes it logarithmic. so the summation of this algorithm is

$$\sum_{i=1}^{k/4} \frac{mn}{2i}$$
$$\sum_{i=1}^{k/4} \frac{mn}{2i} = \frac{1}{2}\sum_{i=1}^{k/4} \frac{mn}{i}$$
$$\Rightarrow \frac{1}{2}O(log(mn) \Rightarrow O(log(mn)$$

□

2)

We can define the distance between two points in ways other than euclidean. The L_∞- distance between points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the plane is given by $max(|x_1 - x_2|, |y_1 - y_2|)$ Modify the closest-pair algorithm seen in class to use the L_∞ distance. Justify your algorithm and analyze its running time. Also, write the recurrence for the running time $T(n)$ of your algorithm.

step 1: replace $min\{(x_1 - x_2), (y_1, y_2)\}$ with $max(|x_1 - x_2|, |y_1 - y_2|)$

step 2: Split the points into sets of points P_L and P_R

step 3: we recursively find the closest pairs in P_L and P_R.

ρ_R equal the minimum P_L and P_R.

step 4: we use distance δ = max(ρ_L and ρ_R) around the strip. Now we know the closest points are P_L and P_R or its two points divided by the dividing line. Taking all the points within δ distance from the strip ordered in arrays baced on x and y. Then by spliting up the area into 4 quadrants we can use are distance method to find the closest points in this range. This also fits into the original method without changing the run time

step 5: we compare ρ_L and ρ_R and return the closest.

3)

Professor Caesar wishes to develop an integer-multiplication algorithm that is asymptotically faster than Karatsuba's $O(n^{log_2 3})$ algorithm. His algorithm will use the divide and conquer method, dividing each integer into pieces of size n/4, and the divide and combine steps together will take O(n) time. He needs to determine how many subproblems his algorithm has to create in order to beat Karatsuba's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time T(n) becomes $T(n) = aT(n/4) + n$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Karatsuba's algorithm? Justify your answer

0.1 Justification

The running time for Karatsuba's algorithm is $O(n^{log_2 3})$

The number of sub-problems determines the running time of the problem and case 1 of master theorem applies. So, in worst case, running time of the algortihm will be T(n) = $\Theta(log_b a)$ where b = 4

$$\begin{aligned} n^{log_4 a} &< n^{log_2 3} \\ \Rightarrow n^{log_2 \sqrt{a}} &< n^{log_2 3} \\ \Rightarrow log_2 \sqrt{a} &< log_2 3 \\ \Rightarrow \sqrt{a} &< 3 \\ hence\ a &< 9 \end{aligned}$$

4)

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answers.

- $T(n) = 4T(n/2) + n^2\sqrt{n}$

  because of case 3 in the masters Theorem $f(n/b) \leq cf(n)$ thus the run time is $\Theta(n^2\sqrt{n}) = \Theta(n^{2.5})$

- $T(n) = T(n/2) + T(n/4) + T(n/8) + n$
  Substitution method. guess T(n) = $\Theta(n)$
  $c\frac{7}{8}n + n \leq n$ if $c \leq 8$
  $c\frac{7}{8}n + n \geq n$ if $c \geq 8$
  T(n) = $\Theta(n)$

- $T(n) = T(n-1) + logn$

  $T(n) = T(n-1) + logn = T(n-2) + log(n) + log(n-1)$
  $= T(n-2) + log[n * (n-1)] = T(1) + log(n!)$
  $log(n!) < logn^n \rightarrow \Theta(nlogn)$

  This can also be shows with Recursion Tree approch. Each layer of the tree takes where log(n)+log(n-1)+log(n-2)+log(n-3) + ... +log(1) this gives us logn! $\Rightarrow \Theta$ (nlogn)