CS 141, Fall 2017 Homework 3

Posted: October 16th, 2017 Due: October 23th, 2017

Name: Zhenxiao Qi

Student ID #: 500654348

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted

Problem 1. (25 points)

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Describe a divide and conquer algorithm that takes $O(kn \log k)$ time. Explain carefully why your algorithm takes $O(kn \log k)$ time.

Answer:

Suppose that every sorted array is an element of another sequence A, then we divide the sequence A into 2 subsequence of $n/2$ elements. So we divide $\log k$ times until the size of the subsequence become 1. Then we sort the subsequence recursively and merge the subsequences.

in the visual analysis, the height of the tree is $\log k$, and the time per level is kn. So the total time is $O(kn \log k)$.

we can also analyze step by step:

$$
\begin{aligned}
T(1) &= 1 \\
T(k) &= 2T(k/2) + nk \\
&= 2(2T(k/4) + nk/2) + nk \\
&= 4T(k/4) + 2nk \\
&= 4(2T(k/8) + nk/4) + 2nk \\
&= 8T(k/8) + 3nk \\
&\cdots \\
&= 2^i T(k/2^i) + i * nk \\
&\quad let \quad 2^i = k \\
T(k) &= k + nk \log k \\
&= O(kn \log k)
\end{aligned}
$$

Problem 2. (25 points)

A CS 141 student has been trying to speed-up Strassen's divide-and-conquer for matrix multiplication algorithm. Recall that Strassen's algorithm computes the product of two $n \times n$ matrices in $O(n^{2.81})$, and uses the fact that one can multiply two $n/2 \times n/2$ matrices with only 7 multiplications instead of 8 with the naive matrix-multiplication algorithm. Suppose the student came up with a variant of Strassen's algorithm based on the fact that the product of two $n/3 \times n/3$ can be found with only m multiplication instead of the normal 27. How small would m have to be for this algorithm to be asymptotically faster than Strassen's algorithm covered in class? You can assume $m > 9$. Justify your answer.

Answer:
Justify by master theorem method
$T(n) = mT(n/3) + dn^2$
In this case, a=m, b=3, $f(n) = dn^2$
$f(n) = O(n^{\log_3^m - \varepsilon})$
Thus we are in case 1, $T(n) = O(n^{\log_3^m})$
$\log_3^m < 2.81$
Thus $m \leq 21$

Problem 3. (25 points)

Give a divide-and-conquer algorithm for multiplying two polynomials of degree n in time $O(n^{\log_2 3})$. This algorithm is similar to Karatsuba's integer multiplication algorithm that we covered in class.

Answer:

$$
\begin{aligned}
assume \quad that A(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \\
B(x) &= b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n \\
we \quad divide \quad both \quad A(x) \quad and \quad B(x) \quad into \quad two \quad parts & \\
A(x) &= A_0(x) + x^{n/2} A_1(x) \\
A_0(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n/2} x^{n/2} \\
A_1(x) &= a_{n/2} + a_{n/2+1} x + a_{n/2+2} x^2 + \cdots + a_n x^{n/2} \\
B(x) &= B_0(x) + x^{n/2} B_1(x) \\
B_0(x) &= b_0 + b_1 x + b_2 x^2 + \cdots + b_{n/2} x^{n/2} \\
B_1(x) &= b_{n/2} + b_{n/2+1} x + b_{n/2+2} x^2 + \cdots + b_n x^{n/2} \\
A(x) \cdot B(x) &= (a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n) \cdot (b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n) \\
&= (A_0(x) + x^{n/2} A_1(x)) \cdot (B_0(x) + x^{n/2} B_1(x)) \\
&= A_0(x) B_0(x) + (A_0(x) B_1(x) + A_1(x) B_0(x) x^{n/2} + A_1(x) B_1(X) X^n \\
Let \quad a &= A_0(x), b = A_1(x), c = B_0(x), d = B_1(x) \\
A(x) \cdot B(x) &= ac + [(a-b)(d-c) + (ac+bd)] x^{n/2} + bd
\end{aligned}
$$

therefore, $T(n) = 3T(n/2) + cn$
$T(n) = O(n^{\log_2^3})$by master method

Problem 4. (25 points)

An array A is said to have a *majority element* if more than half of the entries in A are exactly the same. Describe an $O(n \log n)$ divide-and-conquer algorithm that determines whether an array A of n items has a majority element, and if so, returns that item. *The only comparison operation allowed on the items is equality.* That is, your algorithm can determine whether "A[i] == A[j]" or not in $O(1)$ time, but it cannot, for example, compare the items to sort them, or hash the items into buckets. Explain why your algorithm takes $O(n \log n)$ time.

Answer:

First we divide the array into 2 sub-array of $n/2$ elements. until the size of the sub-array is 1. then we call $FindM()$ and return the majority element of the sub-array.
By recursively call $func()$ and $FindM()$, we finally get the majority of the whole array.
Every time we divide the array into 2 sub-arrays, and it takes $O(n)$ time to combine them.

$T(1) = 1$
$T(n) = 2T(n/2) + dn$
$a = 2, b = 2, f(n) = dn$

so $T(n) = O(n \log n)$ by master method
the pseudo-code is as follow:

```
def func(A):
  if (len(A)==1):
    return A
  else m=len(A)/2
  MajorityL=func(A[:m])
  MajorityR=func(A[m+1:])
  return FindM(MajorityL,MajorityR)

def FindM(MajorityL,MajorityR):
  if (MajorityL==MajorityR):
    reutrn MajorityL
  if((MajorityL==Null)&&(MajorityR!=Null)):
    return MajorityR
  if ((MajorityR==Null)&&(MajorityL!=Null)):
    RETURN  MajorityL
  else return Null
```