Week 7: Lab

Collaboration level 0 (no restrictions). Open notes.

Overview: This week we saw two examples of divide-and-conquer algorithms that are widely used: large integer multiplication, and matrix multiplication.

1 The maximum sub-array problem

The maximum sub-array (MPS) problem: Given an array A of n integers, find values of i and j with $0 \leq i \leq j < n$ such that the sum

$$A[i] + A[i+1] + ... + A[j] = \sum_{k=i}^{j} A[k]$$

is maximized. Sometimes this is called the *maximum partial sum problem.*

Example: Consider the array $A = [4, -5, 6, 7, 8, -10, 5]$. What is the MPS?
Answer: the solution to MPS is $i = 2$ and $j = 4$ $(6 + 7 + 8 = 21)$.

Note that if all values are positive, then the problem is trivial ($i = 0$ and $j = n - 1$). The problem is non-trivial only if there are negative numbers.

Applications. This problem might be encountered in financial analysis, and one textbook offers a possible scenario. Basically, think of the values in the array as relative gain/loss of a stock (wrt to some fixed initial value). Finding the maximum sub-array would (retro-actively) tell you when you should have purchased and sold a stock in order to maximize gain. Refer to the textbook for full explanation, or just use your imagination. This problem is also asked in interviews. Today we'll come up with an $O(n \lg n)$ solution via divide-and-conquer. [1].

[1] In a couple of weeks we'll come back and give an $O(n)$ solution for this problem using dynamic programming

1. Consider the following array:

$$A = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]$$

   Find the MPS and the corresponding $i =?, j =?$

2. One thought that will come to mind is whether the MPS can include negative numbers. A negative number will afterall decrese the MPS, so it feels that they should be skipped. Or do they?

   Examine this issue and give an argument one way or the other.

3. Describe a straightfoward algorithm to find the MPS of an array A and analyze its running time. We'll refer to this as the simple algorithm, the straightforward algorithm, or the brute-force.

4. As always, the question is: Can we do better? For e.g., can we solve MPS in $O(n \lg n)$ time? As it turns out, a neat $O(n \lg n)$ algorithm for MPS is possible via divide-and-conquer. The subsequent questions guide you towards the solution.

   Assume that you know what the MPS of A is. Now consider the index in the middle of the array. List all the posibilities that can happen based on how the MPS i and j indices are located compared to the middle index $k = \lfloor n/2 \rfloor$.

   Case 1:

   Case 2:

   Case 3:

5. You can use this insight to set up the correctness grounds for recursion. Fill in the blanks:

   Claim: The MPS of array A[0.., n-1] is either (a) the MPS of .....; or (b) the MPS of ..... or (c) .....

6. To find the MPS we need to find the two indices i and j that mark its start and end.

   Now consider the one-dimensional version of this problem, Namely, consider that the left index of the MPS l is given and you want to find the index j ($l \leq j < n$) such that $A[l] + A[l+1] + ... + A[j]$ is maximized. (This would be the "Find the MPS that starts at a given index" problem).

   How fast can you find index j in this case? Remember that l is given.

7. Similarly, assume that the right index r of the MPS is given, and you want to find the index i such that $A[i] + A[i+1] + ... + A[r]$ is maximized. (This would be the "Find the MPS that ends at a given index" problem).

   How would you find index j and how fast? Brief answer ok.

8. So if *you knew* the start or end of the MPS, you could find the other side in $O(n)$ time. Right? What if: an oracle told you that a certain index k is part of the MPS; would that help? how would you use that to find the MPS in linear time?

9. Describe an $O(n \log n)$ divide-and-conquer algorithm for solving MPS problem.

2 Another class of divide-and-conquer algorithms

The algorithms we saw so far use divide-and-conquer in a particular way: namely they divide the problem into half-problems and solve *both* half-problems recursively: We start from a problem of size n, then solve two subproblems of size $n/2$, then four subproblems of size $n/4$, and so on. Note that at every recursion level we still look at *all* n elements.

Now let's consider Binary search: To search an array of size n, you search either the left half or the right half, recursively. So you start with an array and recurse on one of its halves. You are still *dividing* the problem, but you only solve one of the subproblems, not both of them. This can also be considered a divide-and-conquer-type algorithm.

The next problem is an example of this type of divide-and-conquer (we will see more examples later).

1. Suppose we are given an array $A[1..n]$ with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a *local minimum* if it is less or equal to both its neighbors, or more formally, if $A[x-1] \geq A[x]$ and $A[x \ \leq A[x+1]$. For example. there are six local minima in the following array:

$$A = [9, 7, 7, 2, 1, 3, 7, 5, 4, 7, 3, 3, 4, 8, 6, 9]$$

.

We can obviously find a a local minimum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in $O(\lg n)$ time. (*Hint: with the given boundary conditions, the array must have at least one local minimum. Why?*)

We expect: pseudocode and a brief English description of your algorithm; why is it correct, i.e. why i can't miss the minimum value; (3) analysis of its running time.