Divide-and-Conquer: The Maximum Partial Sum

(CLRS 4.1)

Laura Toma, Bowdoin College

D&C is a powerful technique for solving problems:

To Solve problem P:

1. *Divide* P into smaller problems P_1, P_2
2. *Conquer* by solving the (smaller) subproblems recursively.
3. *Combine* solutions to P_1, P_2 into solution for P.

The Maximum Partial Sum (Maximum Sub-Array) Problem

Definition. The *maximum partial sum* (MPS) problem is defined as follows. Given an array A of n integers, find values of i and j with $0 \le i \le j < n$ such that

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

is maximized. Sometimes this is called the *maximum sub-array* problem.

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

What if there are no negative values?

Applications. This problem might be encounter in financial analysis, and the textbook offers a nice explanation. Basically, think of the values in the array as relative gain/loss of a stock (with respect 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. There are also applications in genomics. More to the point, perhaps, this problem is also asked in interviews. We'll come up with an $O(n \lg n)$ solution via divide-and-conquer. A faster, linear solution is also possible.

In-Class Work

1. Consider the following array:

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

Find MPS (what is $i = ?, j = ?$).

2. Describe an algorithm to find the MPS of an array A and analyze its running time. We'll refer to this as the simple, or straightforward algorithm.

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. We'll come up with it in a few steps.

3. First, we'll consider an easier variant of the problem. Namely, consider that the left index l is given and you want to find the index j ($\ell \leq j < n$) such that

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

is maximized. Let's call this problem $LMPS(\ell)$, namely the *maximal partial sum starting at left position ℓ.*

Example: For the array [4,-5,6,7,8,-10,5] the solution to $LMPS(3)$ is $j = 4$, $(7 + 8 = 15)$.

4. Similarly, given the right index r, you want to find the value i $(0 \leq i \leq r)$ such that

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

is maximized.

Example: For $A = [4, -5, 6, 7, 8, -10, 5]$ the solution to $RMPS(6)$ is $i = 2$, $(5-10+8+7+6 = 16)$.

Describe an $O(n)$ time algorithm for each of these problems: $LMPS(\ell)$ and $RMPS(r)$.

5. If someone (an oracle) told you that a certain index k is part of the MPS, does that help? how would you use that to find the MPS?

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