# In-class work: The divide-and-conquer technique (CLRS 4.1: The maximum sub-array problem)

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

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

is maximized.

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

(1) Consider the following array:

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

Try to find MPS by hand. This will give an example of how MPS can include negative numbers.

(2) Describe an algorithm to find the MPS 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 very 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 consider the *left position $\ell$ maximal partial sum* problem ($LMPS_\ell$). Here the left index is given and the problem is 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.

**Example**: For the array [4,-5,6,7,8,-10,5] the solution to $LMPS_3$ is $j = 4$ $(7 + 8 = 15)$.

Describe an $O(n)$ time algorithms for solving $LMPS_\ell$ (given $A$ and $\ell$).

(4) Similarly, the *right position $r$ maximal partial sum* problem ($RMPS_r$), consists of finding 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 algorithms for solving $RMPS_r$ (given $A$ and $r$).

(5) Using the algorithm for $LMPS_\ell$, describe a simple $O(n^2)$ algorithm for $MPS$.

(6) Using $O(n)$ time algorithms for $LMPS_\ell$ and $RMPS_r$, describe an $O(n \log n)$ divide-and-conquer algorithm for solving $MPS$.