# Test your knowledge: Solve the max subarray problem 4 ways

Prof. Glencora Borradaile
Oregon State University

For this project, you will design, implement and analyze (both experimentally and mathematically) *four* algorithms for the maximum subarray problem:

Given array of small integers $a[1,\ldots,n]$, compute

$$\max_{i\le j}\sum_{k=i}^{j} a[k]$$

For example, $\textsc{MaxSubarray}([31,-41,\mathbf{59},\mathbf{26},\mathbf{-53},\mathbf{58},\mathbf{97},-93,-23,84]) = 187$

You may use any language you choose to implement your algorithms. Be sure that your algorithm correctly finds the maximum subarray of an input array with all negative numbers.

## Instructions

Your three algorithms are to be based on these ideas:

**Algorithm 1: Enumeration** Loop over each pair of indices $i, j$ and compute the sum, $\sum_{k=i}^{j} a[k]$. Keep the best sum you have found so far.

**Algorithm 2: Better Enumeration** Notice that in the previous algorithm, the same sum is computed many times. In particular, notice that $\sum_{k=i}^{j} a[k]$ can be computed from $\sum_{k=i}^{j-1} a[k]$ in $O(1)$ time, rather than starting from scratch. Write a new version of the first algorithm that takes advantage of this observation.

**Algorithm 3: Divide and Conquer** If we split the array into two halves, we know that the maximum subarray will either be

- contained entirely in the first half,
- contained entirely in the second half, or
- made of a suffix of the first half of maximum sum and a prefix of the second half of maximum sum

The first two cases can be found recursively. The last case can be found in linear time.

**Algorithm 4: Dynamic Programming** Your dynamic programming algorithm should be based on the following idea:

- The maximum subarray either uses the last element in the input array, or it doesn't.

Describe the solution to the maximum subarray problem recursively and mathematically based on the above idea.

**Testing for correctness** Above all else, your algorithms should be correct. A file containing test sets can be found here: `http://www.eecs.orst.edu/~glencora/cs325/mstest.txt` The file has one test case per line (10 cases each with 100 entries). A line corresponding to the example above would be:

```
[31, -41, 59, 26, -53, 58, 97, -93, -23, 84], 187
```

with the input array followed by the sum of the maximum subarray. You may use this test file to check that your code is correct. You should also test your code on small hand-generated instances.

**Experimental analysis** For the experimental analysis you will plot running times as a function of input size. Every programming language should provide access to a clock (not necessarily in seconds). Run each of your algorithms on input arrays of size $100, 200, 300, \ldots, 900$ and $1000, 2000, 3000, \ldots, 9000$ (that is, you should have 18 data points for each algorithm). The first enumerative algorithm may be frustratingly slow, so you may compute running times for sizes $100, 200, 300, \ldots, 900$.

To do this, generate random instances using a random number generator as provided by your programming language. Remember to include both positive and negative numbers! For each data point, you should take the average of a small number (say, 10) runs to smooth out any noise. For example, for the first data point, you will do something like:

```
for i = 1:10
    A = random array with 100 entries
    start clock
    maxsubarray(A)
    pause clock
return elapsed time
```

Note that you should not include the time to generate the instance.

Plot the running times as a function of input size for each algorithm in a single plot. Label your plot (axes, title, etc.). Include an additional plot of the running times on a log-log axis. See here for an explanation: `http://en.wikipedia.org/wiki/Log-log_graph` Note that if the slope of a line in a log-log plot is $m$, then the line is of the form $O(x^m)$ on a linear plot. You may also find this video helpful: `http://www.khanacademy.org/math/algebra/logarithms/v/logarithmic-scale`

For an example of an experimental analysis in java, see `http://algs4.cs.princeton.edu/14analysis/`

## Project report

For each of the above algorithms, your report **must** include:

**Mathematical analysis** Give pseudocode for each algorithm and an analysis of the asymptotic running-times of the algorithms.

**Theoretical correctness** For the third, recursive algorithm, write a formal proof of correctness using induction.

**Experimental analysis** Perform an experimental analysis and include plots as described above.

**Extrapolation and interpretation** Use the data from the experimental analysis to answer the following questions:

1. For each algorithm, what is the size of the biggest instance that you could solve with your algorithm in one hour?
2. Determine the slope of the lines in your log-log plot and from these slopes infer the experimental running time for each algorithm. Discuss any discrepancies between the experimental and theoretical running times.