CS 325 Project 1: Maximum Sum Subarray

Your report must be typed and submitted online. Each team member's name must be listed as well as any resources used to finish the project.

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

Given an array of small integers A[1,…,*n*], compute $\max_{i \le j} \sum_{k=i}^{j} A[k]$, and determine the subarray

> For example, MAXSUBARRAY([31, -41, 59, 26, -53, 58, 97, -93, -23, 84])=187 with subarray = [**59, 26,-53, 58, 97].** Note: If there are multiple subarrays with the same sum you only need to identify one.

Instructions

You may use any language you choose to implement your algorithms. All algorithms will take as input an array and output the subarray with a maximum sum along with the sum. Your 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 the subarray and a prefix of the second half

Algorithm 4: Linear-time. Use the following ideas to develop a nonrecursive linear time algorithm. Start at the left end of the array and progress towards the right, keeping track of the maximum subarray sum seen so far. Knowing a maximum subarray of A[1 . . j], extend the answer to find a maximum subarray ending at index *j*+1 by using the following observation: a maximum subarray of A[1 . . j+1] is either a maximum subarray of A[1 . . j] or a subarray A[i . . j+1], for some 1 ≤ $i \le j$+1. Determine a maximum subarray of the form A[i . . j+1] in constant time based on knowing a maximum subarray ending at index *j*.

CS 325 Project 1: Maximum Sum Subarray

You can implement the four algorithms in one program or have four separate programs (include instructions on compiling and running the code in the README file). If applicable include a make file.

Input and Output. Your program should read the input from the file MSS_Problems.txt. Output is to be written to MSS_Results.txt. The output will include the max sum value and the corresponding array. The output from all four algorithms on all arrays should be in one file (there will be redundancy). We will run your program using MSS_Problems.txt. All elements in the input arrays will be integers with at least one positive element in each array.

Testing for correctness. Above all else your algorithm should be correct. A file containing test sets, MSS_TestProblems.txt and solutions MSS_TestResults.txt will be posted. You may use these files to check if your code is correct. You should also test your code on self-generated instances. You will not submit results from these test cases.

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 various input sizes for example n = 100, 200,…,900 or 1000,2000,…,10000. If your algorithm runs very fast and has a time of 0 you will need to increase the sizes of n. Likewise if the algorithm runs for over an hour you will need to decrease your input size. Each algorithm may require different sizes of n and you will need at least 10 different input sizes for each algorithm.

For each input size n and algorithm collect the running times for at least 10 different input arrays. To do this, generate random instances using a random number generator as provided by you programming language. Note that you should not include the time to generate the instance or write the results to a file in your timing analysis.

Experimental Analysis for each Algorithm:

1. Calculate the average running time for each n. Remember you have at least 10 values for each n.
2. Plot the average running times as a function of input size n. Label your graphs (axes, title, etc).
3. Find a function that models the relationship between input size n and time. This function will produce a curve that "fits" the data you plotted in part 2. To determine the equation of the function for the curve use regression techniques. The shape of the curve will determine the type of regression you use. Is the data linear/quadratic/logarithmic/exponential?

4. Discuss any discrepancies between the experimental and theoretical running times.

5. Use the regression model to determine the largest input for the algorithm that can be solved in 5 seconds, 10 seconds and 1minute.

6. For each algorithm create a log-log plot of the running times. http://en.wikipedia.org/wiki/Log-log_plot . If the data appears linear with slope m in the loglog plot then the function is of the form $O(x^m)$. You may find this video helpful: http://www.khanacademy.org/math/algebra/logarithms/v/logarithmic-scale

7. Also present the results of all four Algorithms together on a single graph and/or log-log plot depending on the scale

Project Report

Your report must include:

- **Names and Group Number:** List the names of all group members and the Group number at the top of the report.
- **Theoretical Run-time Analysis:** Give pseudo-code for each of the four algorithms and an analysis of the asymptotic running-times of the algorithms.
- **Testing:** In order to get credit for this project, your code must produce correct responses. Test all algorithms against a test set. Describe how you tested your algorithm. You do not need to include the test results.
- **Experimental Analysis**: Perform experimental analysis and include plots as described above part 1) – 7).

Submit one copy per group to the following locations:

- **Canvas** - Project Report (PDF) only. In the comment section post the onid username of the student who submits the code to TEACH.
- **TEACH**
  - ✓ **Project Report** : PDF
  - ✓ **Code:** Submit a ZIP file containing - your fully functioning program that runs on our engr server FLIP for verification, a make file and a README with directions describing exactly how to use it.
  - ✓ **Results:** Submit results for the MSS_Problems.txt in the form of a .txt file MSS_Results.txt