School of Electrical and Computer Engineering

CptS223: Advanced Data Structures C/C++

Fall 2018

Programming Assignment 2 (PA2)

Maximum Subsequence Sum Problem

Due: Wednesday 9/12/2018 @ 11:59pm

0. Instructions

All source code must be in C++. You are required to use the Unix environment. Your submission should accompany a report that explains your work. Your code will be tested on EECS servers. Thus, you are strongly encouraged to test (or event write) your code on EECS servers before submitting them. If you do not have an EECS account, you should get one immediately. The EECS IT team will assist you with setting up your EECS account. Look for IT on this page for contact information of the IT personnel:

https://school.eecs.wsu.edu/people/staff/

If you have not logged into EECS Gitlab, you should do so immediately. Information about how to install Git on your computer and how to upload your package (code + report) to the EECS Git can be found under **"Programming Assignments"** section of the course web page. All submission files should be submitted through Git. Note that submissions through any other means for example by emailing your package to the instructor/TA/TAs will be discarded and will NOT be graded. So please follow the above instructions carefully.

You can find the course web page following this link:

http://epsl.eecs.wsu.edu/teaching/

This is an individual programming assignment. No team work is allowed.

The final material for submission should be entirely written by you. If you decide to consult with others or refer materials online, you MUST give due credits to these sources (people, books, webpages, etc.) by listing them in the report that accompanies your submission. Note that no points will be deducted for referencing these sources. However, your discussion/consultation should be limited to the initial design level (if any). Sharing or even showing your source code/assignment solution verbiage to anyone else in the class, or direct reproduction of source code/verbiage from online resources, will all be considered plagiarism, and will therefore be awarded ZERO points and subject to the WSU Academic Dishonesty policy. (Reproducing from the Weiss textbook is an exception to this rule, and such reproduction is encouraged wherever possible.)

Late submission policy: A late penalty of 10% will be assessed for late submissions within the next 24 hours (i.e., until midnight of the next day after the submission deadline). After this one-time late submission, no submissions will be accepted.

1. Description of the Assignment

For this assignment you will be comparing the performance of the four different algorithms we discussed in class for the maximum subsequence sum problem. Here are the details:

- Implement the four algorithms (maxSubSum1, maxSubSum2, maxSubSum3, maxSubSum4) from the Weiss textbook (pages 52-58) [You can find these algorithms in "Appendix" section (section 4 of this document)]. You will need to implement your own max3 function, which is needed to complete the maxSubSum3 code. For the maxSubSum4 code alone, you can either use the Weiss version or the dynamic programming version discussed in class (no need to do both).
- Write a test driver code called *test_driver.cpp* which has a main() function that will allow you to
    i. load an arbitrary input array from a text file (to be specified as an argument to the program);
    ii. run each algorithm on the input array; and
    iii. report their respective running times (the timing should not include the initial input load time). Report time in microseconds.
- You should test all the four algorithms on varying input sizes: 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 (increase more as needed to show difference in times). For each size, test on 10 different inputs and use the average of these running times to generate the plot for that input size. If the runtime for some of your codes take a very long time (>1 hour) then there is no need to document or plot the runtime other than note it down as ">1 hour" in your runtime table.

How to generate inputs?

A set of inputs are generated and have been placed in a zip archive (called inputs.zip). The zip file is included as part of this assignment. The zip file can be also downloaded from the following address:

http://epsl.eecs.wsu.edu/wp-content/uploads/2018/02/inputs.zip

Alternatively, if you want to generate the inputs yourself, then please follow these instructions: use the program *gen_random_arr.c* (or similar) to generate a specified number ("#samples") of random arrays, each with positive or negative integers in the range of -9 to +9 for a user-specified input size ("*n*"). You need to compile the code on your machine. (Please read the code to understand what it is doing.) Let us assume *gen_random_arr* be the name of the executable you created. Then the program can be used as follows:

gen_random_arr {specify array size} {#samples}

For example, "*gen_random_arr 8 10*" will generate 10 different random arrays each with 8 elements. The arrays will be stored and available for use in 10 different files which will be named as: input_8_0.txt, input_8,1.txt, input_8_2.txt, ..., input_8_9.txt.

The program *gen_random_arr.c* is provided as part of this assignment and is also available at the following address.

http://epsl.eecs.wsu.edu/wp-content/uploads/2018/02/gen_random_arr.c

You can either use the inputs you generated or the inputs that are provided, in your tests.

What outputs should my program generate?

- In each run of your test, you should output:
- the name of the algorithm (maxSubSum1,maxSubSum2, maxSubSum3, maxSubSum4);
- the size of the input sequence (n);
- the final result (maximum subsequence sum); and
- the total time taken by the algorithm (not including the input loading time or any other print routines).

What plots should I create and how?

Use the above test results and plot the total running times for all 4 algorithms (on Y axis) against each of the input sizes from 8 to 8192 (on X axis). Create a single plot for all the four algorithms so that you are able to cross-compare their behavior. Use legends to show which curve is for which algorithm. You must use electronic tool such as matlab, gnuplot, Excel, or similar tools to create the plot. Hand-drawn plots will not be accepted.

2. Report

In a separate document (Word or PDF), compile the following sections:

A: Problem Statement

In 1-2 sentences state the goal(s) of this exercise.

B: Experimental Setup

- Specify the machine architecture (CPU, clock speed, RAM) where all the testing was conducted.
- Mention whether you used Windows or Unix or Mac OS X for your testing.
- How many experiments were performed and averaged, to determine each point in your plot?

C: Experimental Results

In this section, include the following:

- The plots from the above test results
- Are the observations made in the above plots as per your theoretical expectations? If so, why, and if not, why not? Explain.

3. Grading

This assignment is mainly about empirical testing and analysis. There is not really a design component (except may be for your test code that calls the different function versions, times them and generates a timing report). So for grading this PA, we will primarily look at how well you have designed experiments, what the plots look like, and have you offered satisfactory/convincing rationale to explain your observations. Therefore the points during grading will be distributed as follows (The whole assignment is worth a total of 100 points):

CODING (45 pts):

- (10 pts): Are all versions of the algorithms and test driver implemented correctly?
- (10 pts): Is the code implemented in an efficient way? i.e., are there parts in the code that appear redundant, or implemented in ways that can be easily improved? Does the code conform to good coding practices of Objected Oriented programming?
- (15 pts): Does the code compile and run successfully on a couple of test cases?
- (10 pts): Is the code documented well and generally easy to read (with helpful comments and pointers)?

REPORT (55 pts):

- (15 pts): Is the experimental plan technically sound? Is the experimental setup specified clearly?
- (10 pts): Are results shown as plots in a well annotated manner and are general trends in the results visible?
- (30 pts): Are the justifications/reasons provided to explain the observations analytically sound? Is there a reasonable attempt to explain anomalies (i.e., results that go against analytical expectations), if any?

Obviously to come up with the above evaluation, the TAs are going to both read and run your code.

4.Appendix

```
1  /**
2   * Cubic maximum contiguous subsequence sum algorithm.
3   */
4  int maxSubSum1( const vector<int> & a )
5  {
6      int maxSum = 0;
7
8      for( int i = 0; i < a.size( ); ++i )
9          for( int j = i; j < a.size( ); ++j )
10         {
11             int thisSum = 0;
12
13             for( int k = i; k <= j; ++k )
14                 thisSum += a[ k ];
15
16             if( thisSum > maxSum )
17                 maxSum = thisSum;
18         }
19
20     return maxSum;
21 }
```

Algorithm1: maxSubSum1

```
1  /**
2   * Quadratic maximum contiguous subsequence sum algorithm.
3   */
4  int maxSubSum2( const vector<int> & a )
5  {
6      int maxSum = 0;
7
8      for( int i = 0; i < a.size( ); ++i )
9      {
10         int thisSum = 0;
11         for( int j = i; j < a.size( ); ++j )
12         {
13             thisSum += a[ j ];
14
15             if( thisSum > maxSum )
16                 maxSum = thisSum;
17         }
18     }
19
20     return maxSum;
21 }
```

Algorithm2: maxSubSum2

```
1  /**
2   * Recursive maximum contiguous subsequence sum algorithm.
3   * Finds maximum sum in subarray spanning a[left..right].
4   * Does not attempt to maintain actual best sequence.
5   */
6  int maxSumRec( const vector<int> & a, int left, int right )
7  {
8      if( left == right )  // Base case
9          if( a[ left ] > 0 )
10             return a[ left ];
11         else
12             return 0;
13
14     int center = ( left + right ) / 2;
15     int maxLeftSum  = maxSumRec( a, left, center );
16     int maxRightSum = maxSumRec( a, center + 1, right );
17
18     int maxLeftBorderSum = 0, leftBorderSum = 0;
19     for( int i = center; i >= left; --i )
20     {
21         leftBorderSum += a[ i ];
22         if( leftBorderSum > maxLeftBorderSum )
23             maxLeftBorderSum = leftBorderSum;
24     }
25
26     int maxRightBorderSum = 0, rightBorderSum = 0;
27     for( int j = center + 1; j <= right; ++j )
28     {
29         rightBorderSum += a[ j ];
30         if( rightBorderSum > maxRightBorderSum )
31             maxRightBorderSum = rightBorderSum;
32     }
33
34     return max3( maxLeftSum, maxRightSum,
35                  maxLeftBorderSum + maxRightBorderSum );
36 }
37
38 /**
39  * Driver for divide-and-conquer maximum contiguous
40  * subsequence sum algorithm.
41  */
42 int maxSubSum3( const vector<int> & a )
43 {
44     return maxSumRec( a, 0, a.size( ) - 1 );
45 }
```

Algorithm3: maxSubSum3

```
1  /**
2   * Linear-time maximum contiguous subsequence sum algorithm.
3   */
4  int maxSubSum4( const vector<int> & a )
5  {
6      int maxSum = 0, thisSum = 0;
7
8      for( int j = 0; j < a.size( ); ++j )
9      {
10         thisSum += a[ j ];
11
12         if( thisSum > maxSum )
13             maxSum = thisSum;
14         else if( thisSum < 0 )
15             thisSum = 0;
16     }
17
18     return maxSum;
19 }
```

Algorithm4: maxSubSum4