CSE 330 LABORATORY -- Week 3, Spring 2018

Instructor: Kerstin Voigt

This lab will have you experience three algorithms that solve the same problem, but do so with markedly different computational complexities.

The problem to solve is the "maximal subsequence sum" problem from our textbook (Weiss, DSAAC++) in chapter 2. A vector of arbitrary positive and negative integers is given. In this sequence of numbers the sum of values of any subsequence of contiguous values can be computed. The problem is to determine the maximal sum of this sort that can be computed for the given vector.

You may use the following C++ code in order to produce a vector of random positive and negative integers.

In .h file, you need …

```
#include <cstdlib>
#include <vector>
#include <ctime>
```

… and the function prototypes for the below.

In .cpp file, implement functions …

```
void rand_seed()
{
	int seed = static_cast<int>(time(0));
	srand(seed);
}

// random integer between a and b;
int rand_int(int a, int b)
{
	return a + rand() % (b - a + 1);
}

// 33% chance of negative number; repeats allowed;
// absolute value between from and upto;

void random_vector(int k, int from, int upto, vector<int>& v)
{
	int rnum;
	int r33;
	for (int i = 1; i <= k; i++)
	{
		rnum = rand_int(from,upto);
		r33 = rand_int(1,4);
```

```
            if (r33 == 3)
                  v.push_back(-rnum);
            else
                  v.push_back(rnum);
      }
      return;
}
```

The three different functions to solve the maximal subsequence sum problem are

```
// adopted from Weiss, DSAAC++, p. 61
int max_subseq_sum_cube(const vector<int>& vec, int& ops)
{
      int maxSum = 0;
      ops = 0;
      for (int i = 0; i < vec.size(); i++)
            for(int j = i; j < vec.size(); j++)
            {
                  int thisSum = 0;
                  for (int k = i; k <= j; k++)
                  {
                        thisSum += vec[k];
                        ops += 1;          // count number of +;
                  }
                  if (thisSum > maxSum)
                        maxSum = thisSum;
            }
      return maxSum;
}

// adopted from Weiss, DSAAC++, p. 62
int max_subseq_sum_quad(const vector<int>& vec, int& ops)
{
      int maxSum = 0;
      ops = 0;
      for (int i = 0; i < vec.size(); i++)
      {
            int thisSum = 0;
            for(int j = i; j < vec.size(); j++)
            {
                  thisSum += vec[j];
                  ops += 1;          // count number of +;
                  if (thisSum > maxSum)
                        maxSum = thisSum;
            }
      }
      return maxSum;
}
```

```
// adopted from Weiss, DSAAC++, p. 66
int max_subseq_sum_lin(const vector<int>& vec, int& ops)
{
      int maxSum = 0;
      int thisSum = 0;
      ops = 0;

      for (int i = 0; i < vec.size(); i++)
      {
            thisSum += vec[i];
            ops += 1;
            if (thisSum > maxSum)
                  maxSum = thisSum;
            else if (thisSum < 0)
                  thisSum = 0;
      }
      return maxSum;
}
```

Notice that the function implementations are identical to the ones featured in the textbook, except for the **reference argument 'int& ops' which is increased by one each time an addition is executed.** This will be our mechanism by which we measure the **computational complexity** of the function. We care less about the actual time taken up by the bodies of the functions; much more interesting is how the cost of computation changes with the size of the problem. We use the number of additions as a basic measure of computational cost; we increase the size of the problem by increasing the size of the vector.

Complete your .h and .cpp files (you may call them MaxSubSum.h and MaxSumSum.cpp) by copying the given three functions.

Next launch a small study of computational complexity by setting up an 'int main()' function (e.g., in a File Lab3Main.cpp) which allows you to request the generation of random integer vectors of a given size. The program is then to call each of the three max_subseq_sum functions to find the maximal subsequence sum and to report the computational cost through the 'int& ops' reference argument.

To do so, in your main, you will need to declare an integer with is passed to the functions. A bare-bones variant of code to do this could look like this:

```
int no_ops = 0 ;
…
int maxsum = max_subseq_sum_...(…., no_ops);
…
cout ≪ maxsum ≪ " " ≪ no_ops;
```

Run your completed program for random vectors of size 5, 10, 15, 20, 25, and 30. For each vector, run each of the three functions; all functions should produce the same maximal subsequence sum. However, the number of reported operations will vary among them. Obtain sums and number of operations for each max_sum… function, and plot the data (use Excel or equivalent).

	vecsize = 5	10	15	20	25	30
max_subseq_...cube						
max_subseq_...quad						
Max_subseq_...lin						

Credit for this lab: (1) Make sure to sign the **signup sheet**. (2) **Combine all your .h, and .cpp files into one large file,** (adding a line '-----' between the contents of each file; this file will no longer qualify as compilable C++ file, but no worries), add the filled in **data table and line plots** at the end of the file and **email it to Sarthak** (006118706@coyote.csusb.edu) by midnight of the next day (Wednesday for the Tuesday group, and Friday for the Thursday group).