EECE 7205 Fundamentals of Computer Engineering
Fall 2016

Homework 2: Due on Blackboard by Monday October 10 2016, 11:55pm

- This test contains three problems. They allow you to earn 100 points.
- Homework are due by the due date indicated on Blackboard. After that precise date and time the link for submitting your solutions to the homework problems will disappear. The instructor or the TA will not accept e-mails with attached the PDF file of your solutions. **No late submissions will be accepted.**
- You should upload a .zip file containing a .cpp file (solution to Problem # 1) for the TA to check and compile. Be sure to use basic C++, so that the TA can compile the program independently on the platform you used. In any case, write a C++ comment at the beginning of your file with your first and last names, and with the platform you used for developing the program. The .zip file should also contain the .pdf with the solutions to the other problems.
- Name your homework file with your name_last-name_number-of-hmw.zip, so that it can be traced back to you once it has been downloaded from Blackboard. Use similar naming for the .cpp file.
- Show your work, as partial credit can be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.

- **Problem # 1 [50 points]**. Write a C++ program for sorting an array of n integers. More specifically, the program should ask in input an integer number $n > 1$ and a real number $m > 1.0$. Three functions should be defined.

  One that implements the sorting algorithm INSERTION-SORT(A, n) as seen in class for sorting an array of integers.

  A second function should populate the vector with integer numbers selected randomly and uniformly in the range $[-m, m]$. For generating random numbers you should use the `<random>` header of C++11 and *not* the function `rand()` from `cstdlib`.

  Finally, a third function should be provided that (neatly) prints a vector out. The C++ STL `vector` container should be used, i.e., the prototypes of the functions should be as follows:

```
void insertionSort ( vector< int > &A );
void populateVector ( vector< int > &A, double m );
void printVector ( vector< int > A );
```

  (Notice that the parameter n indicating the size of the array should *not* be passed as a parameter to any of the functions.)

  The function `main ()` of the program should provide a demonstration (testing) of the INSERTION-SORT implementation.

- **Problem # 2 [20 points]**. Consider the following function, where the input is a non-negative integer n.

```
int misterya( int n ) {

  int S = 0;

  for( int i = 1; i < n+1; i++ )
    S += i * i;

  return S;

}
```

1. What does the algorithm compute?
2. What is its basic operation?
3. How many time is the basic operation executed?
4. What is the efficiency class (i.e., the time complexity) of this algorithm?
5. Suggest a better algorithm (write its pseudo-code or its C++ code) for computing the same function and indicate its complexity. If you cannot do it, prove that, in fact, this cannot be done.

- **Problem # 3 [30 points]**. Although MERGESORT as seen in class runs in $\Theta(n \log n)$ worst-case time, and INSERTIONSORT runs in $O(n^2)$ worst-case time, the constant factors (hidden by the asymptotic notation) of INSERTIONSORT can make it faster in practice for small problem sizes on many machines. Therefore, it makes sense to use INSERTIONSORT within MERGESORT when sub-problems becomes sufficiently small. Consider a modification to MERGESORT for sorting an array of n elements (say, integer numbers) in which n/k sub-sequences each of length k are sorted using INSERTIONSORT and then merged by using the standard MERGE procedure used by MERGESORT, where k is a value to be determined.

  1. Show that INSERTIONSORT can sort the n/k sub-sequences, each of length k, in $\Theta(nk)$ worst-case time.
  2. Show how to merge the sub-sequences in $\Theta(n \log(n/k))$ worst-case time.
  3. Given that the modified algorithm runs in $\Theta(nk + n \log(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as the MERGESORT seen in class?
  4. How should we choose k in practice?