Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7205: Fundamentals of Computer Engineering

Homework 2

Instructions

- For programming problems:
  - Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
  - You are **not** allowed to use any advanced C++ library unless it is clearly allowed by the problem. For example, you cannot use a library function to sort a list of data if the problem is asking you to implement an algorithm to sort the list.
  - At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
  - Your code must compile and tested with a standard GCC compiler (available in the CoE Linux server). before submitting the source code file(s) (do not submit any compiled/executable files):
    a. If your program does not compile with a GCC compiler due to incompatible text encoding format, then make sure the program is saved with Encoding Unicode (UTF-8). In visual studio, Save As -> Click on the arrow next to Save -> Save with Encoding -> Yes -> Unicode (UTF-8) -> Ok
    b. Compile using g++ -std=c++11 <filename>

- Submit the following to the homework assignment page on Canvas:
  - Your homework report developed by a word processor and submitted as <u>one</u> PDF file. For answers that require drawing and if it is difficult on you to use a drawing application, which is preferred, you can neatly hand draw the answer, scan it, and include it into your report. The report includes the following (depending on the assignment contents):
    a. Answers to the non-programming problems that show all the details of the steps you follow to reach these answers.
    b. A summary of your approach to solve the programming problems.
    c. The screen shots of the sample run(s) of your program(s).
  - Your well-commented programs source code files (i.e., the .cc or .cpp files).

  Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

<u>Note</u>: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded. This means all required files must be included in this last submission attempt.

Problem 1 (30 Points)

Write a C++ program for sorting an array *A* of *n* integers using the Merge Sort algorithm. First you need to implement both the MERGE-SORT and MERGE algorithms (shown below). The `main()` function of your program must carry out the following tasks:

1. Ask the user to input the value of *n*, where 1< *n* ≤ 50
2. Fill *A* with random integers in the range 0 to 100. To generate such random numbers, you need to use the <random> header. Check the following link for an example: http://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
3. Call the MERGE-SORT function to sort the contents of *A* (where MERGE-SORT needs to call the MERGE function).
4. Display on the screen the contents of the array *A* before and after calling MERGE-SORT on it.
5. Modify the MERGE function code so that if all elements in both sub-arrays are already sorted (best case scenario where all elements in one sub-array are less than all the elements in the other sub-array), then the function skip the merge process. Test your modified MERGE function by calling MERGE-SORT with an array with already sorted content.

```
MERGE-SORT($A, p, r$)
1  if $p < r$
2      $q = \lfloor (p + r)/2 \rfloor$
3      MERGE-SORT($A, p, q$)
4      MERGE-SORT($A, q + 1, r$)
5      MERGE($A, p, q, r$)
```

```
MERGE($A, p, q, r$)
 1  $n_1 = q - p + 1$
 2  $n_2 = r - q$
 3  let $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
        be new arrays
 4  for $i = 1$ to $n_1$
 5      $L[i] = A[p + i - 1]$
 6  for $j = 1$ to $n_2$
 7      $R[j] = A[q + j]$
 8  $L[n_1 + 1] = \infty$
 9  $R[n_2 + 1] = \infty$
10  $i = 1$
11  $j = 1$
12  for $k = p$ to $r$
13      if $L[i] \le R[j]$
14          $A[k] = L[i]$
15          $i = i + 1$
16      else $A[k] = R[j]$
17          $j = j + 1$
```

Problem 2 (10 Points)

In the above MERGE algorithm and to avoid having to check whether either list is empty in each basic step, a sentinel value of ∞ is placed at the end of each list.

Rewrite the algorithm so that it does not use sentinels, instead it stops once either array *L* or *R* has had all its elements copied back to *A* and then copies the remainder of the other array back into *A*.

No need to submit any C++ program code for this problem (only the updated algorithm in pseudo code similar to what is presented in the previous problem).

Problem 3 (20 Points)

Write a C++ program to test the following two functions. Call the functions with values for ***n*** between 1 and 10.

```
int F1(int n)
{
     if (n == 0) return 1;
     return F1(n - 1) + F1(n - 1);
}

int F2(int n)
{
     if (n == 0) return 1;
     if (n % 2 == 0) {
          int result = F2(n / 2);
          return result * result;
     }
     else
          return 2 * F2(n - 1);
}
```

Submit your test program and in your report answer the following questions:

a. What does each function do?
b. Which function is faster (*hint:* test them with ***n*** = 30)?
c. Guess the running time growth of each function and hence explain why one function is faster than the other.

Problem 4 (20 Points)

```
ProcedureX ($A$)
for $i = 1$ to $A.length - 1$
    for $j = A.length$ downto $i + 1$
        if $A[j] < A[j - 1]$
            exchange $A[j]$ with $A[j - 1]$
```

The above code is for **ProcedureX** that takes list *A* of integers, with starting index 1, as an input parameter.

a) In 70 words or less, explain the purpose of ProcedureX and how it achieves that purpose.
b) If *n* = *A.length*, determine the worst-case running time formula of ProcedureX. Write the formula as a function of *n* (show your steps).

Problem 5 (20 Points)

In the lecture we implemented the insertion sort algorithm using iterative approach. Write a recursive version of that algorithm (only the algorithm not its C++ implementation). In order to sort the contents of array $A[1..n]$ using a recursive version of the insertion sort algorithm, you can recursively sort $A[1..n\text{-}1]$ and then insert $A[n]$ into the sorted array $A[1..n\text{-}1]$. Only provide the insertion sort algorithm and no need to write the algorithm to perform the "insert" task. Following the same level of abstraction used in writing our recursive merge sort algorithm, your recursive insertion sort algorithm should be about 3 or 4 lines of pseudo code.

Following the same approach that we used with analyzing the merge sort algorithm, analyze your recursive insertion sort algorithm to find its running time recurrence equation. Solve the recurrence equation to find the asymptotic notation of the running time.