Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7205: Fundamentals of Computer Engineering

Spring 2020 - Homework 2

Instructions

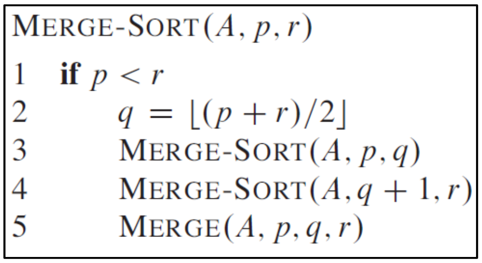
- For programming problems:
  - Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
  - You are not allowed to use any advanced C++ library unless it is clearly allowed by the problem. For example, you cannot use a library 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).
  - Test your code on the COE Linux server before submitting it:
    - If your program does not compile in the COE server due to incompatible text encoding format, then before uploading your source code file to the server make sure it 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
    - Compile using: `g++ -std=c++11 <filename>`
- Submit the following to the homework assignment page on Blackboard:
  - Your homework report submitted as one PDF file. The report includes the answers to the non-programming problems and the screen shots of your program's sample runs for the programming problems. Your report must be developed by a word processor (no handwritten or drawn contents are acceptable).
  - Your well-commented source code files (i.e., the .cc or .cpp files) for the programming problems.
  - Do NOT submit any files as a compressed (zipped) package. Rather, upload individually each of the source code files (cpp or cc files) and the pdf report file.

__Note__: Yon can submit multiple attempts for this homework, however, only your last submitted attempt will be graded.

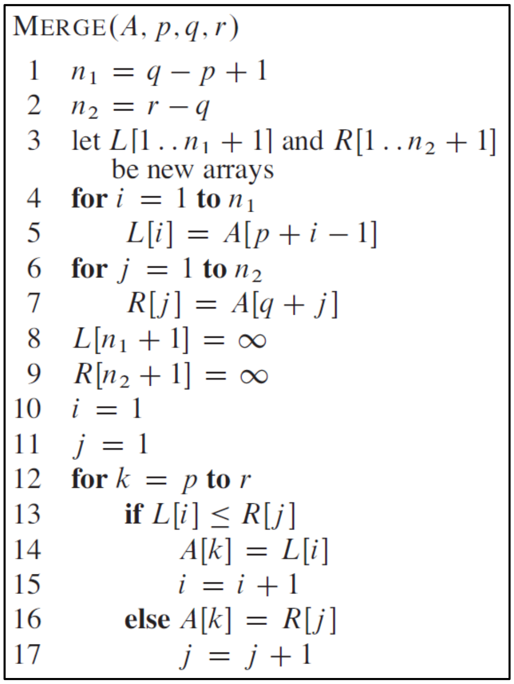
Problem 1 (40 Points)

Write a C++ program for sorting an array *A* of *n* integers. First you need to implement both the MERGE-SORT and MERGE algorithms (shown below). The `main()` function of the program carries 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 sorted array *A*.



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



```
MERGE($A, p, q, r$)
$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
    be new arrays
for $i = 1$ to $n_1$
    $L[i] = A[p + i - 1]$
for $j = 1$ to $n_2$
    $R[j] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ to $r$
    if $L[i] \le R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$
```

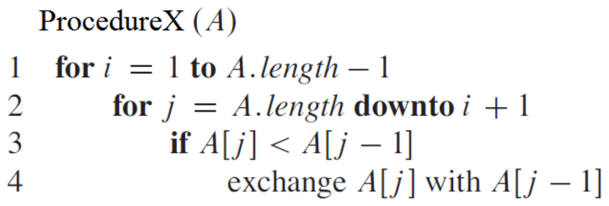
Problem 2 (20 Points)

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 copying the remainder of the other array back into *A*.

No need to submit any updated C++ program code for this problem (on the updated algorithm in pseudo code as presented in the previous problem).

Problem 3 (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 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 4 (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 $A[1..n\text{-}1]$. Follow the same level of abstraction used in writing our recursive merge sort algorithm in slide 28 of the 4th lecture slides.

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.