Programming Project

Posted on September 25, due on November 29

1. Introduction

The first goal of this programming project is to observe empirically the complexities of different algorithms solving the same problem. The second goal is to discover how accurate the theoretical estimates of complexity are when compared to real execution times.

In this project students will implement three algorithms (**ALG1**, **ALG2**, and **ALG3**) solving the same problem. The algorithm implementation should follow *exactly* the pseudocode learned in class, please see the notes/textbook. You can use C, C++, Java, or Python. For other programming languages, please consult with the instructor.

The project submission consists of:

- Project Report
- Source code

2. The Problem & Algorithms to be Implemented and Analyzed

In this project students will study the **Selection Problem**. The Selection Problem is explained in the Textbook chapter 9 and in the notes Module 4.1.

Selection Problem: given an array A[1 … n] of n elements, find the i^{th} order statistic (e.g. i^{th} smallest element) where $1 \leq i \leq n$.

First Algorithm (ALG1)

- strategy: sort the array A using Insertion sort and return A[i].
- Insertion sort is discussed in the notes Module1.1
- pseudocode using A[1..n]:

```
ALG1(A, n, i)
INSERTION-SORT(A,n)
Print A[i]
```

RT = $O(n^2)$

Second Algorithm (ALG2)

- strategy: sort the array A using MergeSort and return A[i].
- MergeSort is discussed in the notes Module 1.2
- pseudocode using A[1..n]:

```
ALG2(A, n, i)
MERGE-SORT(A,1,n)
Print A[i]
```

$RT = O(n \log_2 n)$

Third Algorithm (ALG3)

- strategy: use the RANDOMIZED-SELECT algorithm from textbook/notes (Textbook chapter 9 and notes Module 4.1)
- pseudocode using A[1..n]

```
ALG3(A, n, i)
x = RANDOMIZED-SELECT(A,1,n,i)
Print x
```

Expected RT = O(n)

Notes:

- You need to use the algorithms learned in class (e.g. from the textbook/notes). For example, if the algorithm has a while loop traversing the array, then you need to use a while loop as well. Implementing or using other versions from Internet will not receive credit.
- You need to address the issue with array indexes. In class/textbooks we assume that the arrays start from index 1, e.g. A[1..n], while most programming languages have arrays starting from the index 0, e.g. A[0..n-1]. There are two ways to address this issue, choose one:
  (1) Method1: Adapt the code accordingly, and use A[0..n-1]. Here is for example the pseudocode for INSERTION-SORT using an array A[0..n-1].

```
INSERTION-SORT(A,n)   // assumes A[0..n-1]
for j = 1 to n-1
    key = A[j]
    i = j – 1
    while i ≥ 0 and A[i] > key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key
```

  As you change from A[1..n] to A[0..n-1], notice that for ALG1 and ALG2 you will need to print A[i-1] as the i^{th} order statistics element
  (2) Method2: Allocate one more element, e.g. A[0… n], and ignore A[0]. Then you do not need too many changes to the pseudocode.
- It is acceptable to add "n" as an additional argument in the functions. For example, in the textbook we have INSERTION-SORT(A). You can keep as it is or you can add n such as INSERTION-SORT(A,n).

- You need to implement the supporting functions. For example, for the MERGE-SORT algorithm, you will have to implement the MERGE function.

3. Planning of the Experiments

- Decide which programming language you want to use C, C++, Java, or Python
- We will measure the RT for the following input sizes
  n = 10000, 20000, 30000, 40000, …., 100000. More specifically, n takes 10 values from 10k to 100k with increment of 10k.
  Note: if it takes too long to run your programs, then you can work with the following smaller input sizes n = 1000, 2000, 3000, 4000, …., 10000. There is no penalty in grades if you work with these smaller values.
- The number of iterations (or runs) for each value of n is m = 5 iterations. For example, for n = 10000 elements, you will measure the RT of each algorithm on m = 5 different arrays of length 10000 and report the average RT.
- In this project, let us take i = $\lfloor 2n/3 \rfloor$ for all the experiments
- You need to run all three algorithms on exactly the same input arrays
- To generate the arrays A, use a random number generator. Then for each array instance run the algorithms. One option is to use rand() which generates a random number between 0 and RAND_MAX = 32767. Include #include<stdlib.h>
- To measure the running time, you need a function to return the current time, such as "gettimeofday" which gives the time in sec and microseconds. Use "man gettimeofday" to see the manual description. Then you call this function before and after you run the algorithm and compute the running time as the difference of the two values.
- If you write your code in C/C++ do not allocate the arrays or vectors on the stack, since you will get a **stack overflow** error.

  Do **NOT** allocate the arrays like this:

```
int solution_alg(....) {
   int very_large_array[1000000];
            // THIS GOES ON THE STACK AND IT GETS ERROR
   ....
}
```

- Instead, if you use C/C++ you must allocate the vectors/arrays on the heap (with malloc/calloc/new). Don't forget to deallocate them when they are not needed anymore. Another option for C++ is to declare them as a global variable.

4. Tables and Graphs for the Project Report

- The report must include a total of 4 graphs and 3 tables, as described next
- For each algorithm use a Table to compute the hidden constant c. The table has the following columns: n, TheoreticalRT, EmpiricalRT, Ratio, PredictedRT. See the example below.

Table ALG1

n	TheoreticalRT n^2	EmpiricalRT (msec)	Ratio = (EmpiricalRT)/(TheoreticalRT)	Predicted RT
10^4	10^8	51	$r_1 = 51*10^{-8}$	
$2*10^4$	$4*10^8$	75	$r_2 = 18.75*10^{-8}$	
$3*10^4$	$9*10^8$	97	$r_3 = 10.77*10^{-8}$	
$4*10^4$	$16*10^8$	…	So on ….	
$5*10^4$	$25*10^8$	…		
$6*10^4$	$36*10^8$	….		
….				
$10*10^4$	$100*10^8$	…		

Compute c_1 as follows, $c_1 = \max(r_1, r_2, …., r_{20})$
Note: If r_1 (or another r value) is an outlier, then omit r_1.

PredictedRT = c_1∗TheoreticalRT
PredictedRT = c_1∗n^2

Use similar tables to compute c_2 for ALG2 and constant c_3 for ALG3.

- Draw a graph where x axis has the values n = 10^4, $2*10^4$, $3*10^4$, ….$10*10^4$. Plot the values for ALG1:EmpiricalRT, ALG2:EmpiricalRT, and ALG3:EmpiricalRT.
- Draw a graph where x axis has the values n = 10^4, $2*10^4$, $3*10^4$, ….$10*10^4$. Plot the values for ALG1:EmpiricalRT and ALG1:PredictedRT. Repeat for ALG2 and ALG 3.

5. Structure of the Project Report

Project Title
Student Name

1. **Problem definition**. Clearly state the problem. Define the input size. Identify real-world applications.
2. **Algorithms and RT analysis**. Write the pseudocode for each algorithm and RT analysis.
3. **Experimental Results**. Include the 3 tables and 4 graphs. The graphs/tables cannot be drawn by hand. Use Microsoft Excel, Matlab, or another tool.

4. **Conclusions**. What conclusions can you draw from your experiments? Are the theoretical and empirical results consistent?
5. **References**. List the textbook and any other references if applicable.

6. Grading

The maximum total is 100 points, distributed as follows:

- Project Report: 60 points
  - Problem definition (5 pts)
  - Algorithms & RT (15 pts)
  - Experimental Results (35 pts)
  - Conclusions & References (5 pts)
- Source Code: 40 points
  - Three algorithms implemented correctly following the pseudocode from textbook/notes (30 pts)
  - Correct number of m iterations and compute the average RT(5 pts)
  - All algorithms run on the same input (5 pts)