BME695 Numerical Methods in BME

Warm-up Coding Assignment I

January 2021

Please review these notes before you start!

- Each person will get 30 minutes to discuss the warm-up coding assignment in a one-on-one meeting on either Tuesday (10 a – noon) or Thursday (10 a – noon).
- Please sign up for a 30-minute time slot on the google Excel spreadsheet "Warm-up Coding Assignment Discussion" I have created.
- I have placed a number of documents in the supplemental materials folder under Warm-up Coding Assignment on Google drive. I will cross reference the materials throughout this assignment document. When I do so, I will refer the folder to be the SM folder for convenience.

1 Divide-and-Conquer for Integer Multiplication

The divide-and-conquer strategy solves a problem by

1. Breaking it into subproblems that are themselves smaller instances of the same type of problem
2. Recursively solving these subproblems
3. Appropriately combining their answers

The real work is done piecemeal,in three different places: in the partitioning of problems into subproblems; at the very tail end of the recursion, when the subproblems are so small that they are solved outright; and in the gluing together of partial answers. These are held together and coordinated by the algorithm's core recursive structure.

We will use integer multiplication as an example to see how this technique yields a quite interesting algorithm. For those of you who have known about the divide-and-conquer strategy, this may be to your surprise. At least, it is the case to ME. I have used the strategy for various computing tasks and algorithm designs, e.g., *Merge Sort*, but not here. Later we will visit *Merge Sort* and compare it with other classic sorting algorithms.

I will only present an algorithm statement in the following so that you can basically follow it to work out this assignment. If you need more information about this example, please review the document divide-and-conquer in the SM folder.

Algorithm 1: A divide-and-conquer algorithm for integer multiplication.

```
function multiply($x$,$y$)
Input: Positive integers $x$ and $y$, in binary
Output: Their product

$n$ = max(size of $x$, size of $y$)
if $n = 1$ then
    return $xy$
else
    $x_L, x_R$ = leftmost $\lceil n/2 \rceil$; rightmost $\lfloor n/2 \rfloor$ bits of $x$
    $y_L, y_R$ = leftmost $\lceil n/2 \rceil$; rightmost $\lfloor n/2 \rfloor$ bits of $y$

    $P_1$ = multiply($x_L, y_L$)
    $P_2$ = multiply($x_R, y_R$)
    $P_3$ = multiply($x_L + x_R, y_L + y_R$)
    return $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$
end
```

1.1 Your Tasks

- Implement Algorithm 1 and show me your code line by line and section by section in the one-on-one meeting
- Test the correctness of your algorithm with three integer multiplication problems: 178; 2850×1587; and 45746×25443. For each of the above three computations, output the product and the level of recursion.
- Wrap your code with a series of squared numbers, namely n^2, from $n = 1,000$ to $n = 50,000$ with an increment of 1,000. Record the time for each computation, plot computation time vs. n. Can you find a generic pattern as covered in the Master theorem in the document divide-and-conquer in the SM folder.

2 Sorting Algorithms

Now we come to sorting. What sorting algorithms have you heard of, learnt, used, or implemented? What about *Bubble Sort*, *Selection Sort*? In fact, the problem of sorting a list of numbers lends itself immediately to a divide-and-conquer strategy: split the list into two havles, recursively sort each half, and then merge the two sorted sublists. This is the so-called *Merge Sort* algorithm.

I present an algorithm statement for *mergesort* in the following so that you can basically follow it to work out this assignment. If you need more information about this algorithm, please review the document in place in the SM folder.

Algorithm 2: Merge Sort – A divide-and-conquer strategy for sorting.

```
function mergesort($a[1,\dots,n]$)
Input: An array of numbers $a[1,\dots,n]$
Output: A sorted version of this array

if $n > 1$ then
    return merge (mergesort($a[1,\dots,\lfloor n/2 \rfloor]$), mergesort ($a[\lfloor n/2 \rfloor + 1,$
      $\dots, n]$))
else
    return $a$
end
```

2.1 Your Tasks

Let us consider the problem of sorting arrays of randomly generated integer numbers astoundingly.

- Implement Algorithm 2 and show me your code line by line and section by section in the one-on-one meeting.
- Present algorithm statements for *Bubble Sort*, *Insertion Sort*, *Selection Sort* in a format similar to Algorithms 1 and 2 above. Please typeset them!
- Implement the three algorithms *Bubble Sort*, *Insertion Sort*, *Selection Sort* and show me your code. I know there are source codes online. Preferably you do not just do copy paste. I will check your understanding in the one-on-one meeting.
- Find the JPG file that shows the comparison of classic sorting algorithms in the SM folder. Justify the computational complexity both in terms of time and Space complexity. Try to think about different behaviors on time complexity for different instances (i.e., best, average, worst)
- Run the four algorithms with five sets of experiments (dim = 10, 20, 30, 50, 100). In each set of experiments, you are given 10 randomly generated arrays (*by me*; find the arrays in the excel sheet in the SM folder). Please record the running time. I will first check the correctness and then check the running time.
- Use the computational results of these concrete experiments to further verify the comparison presented in the JPG file.