COMPSCI330 Design and Analysis of Algorithms
Assignment 1

Due Date: Wednesday, January 22, 2020

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit the problems to GradeScope. You will be asked to **label your solution for individual problems**. Failing to label your solution can cost you 5% of the total points (3 points out of 60 for this homework).
- LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you need more time for your homework please use this form and submit a STINF.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Problem 1 (Recursion-I)**.** Read the descriptions of following algorithms, write a recursion for its running time, and solve it (you can use Master's Theorem). Your answer should just contain two lines, the first line is the recursion and the second line is its solution.

(a) (5 points) Strassen's Matrix Multiplication: The naïve way of multiplying two $n \times n$ matrices takes $O(n^3)$ time. Strassen developed a divide-and-conquer algorithm for matrix multiplication. In this algorithm, to multiply two $n \times n$ matrices, one needs to recursively multiply $n/2 \times n/2$ matrices 7 times, and merging results takes $O(n^2)$ time. Let $T(n)$ be the running time of Strassen's algorithm, write a recursion for $T(n)$ and solve it.

(b) (5 points) Inefficient Inversion: For the Counting Inversions problem, suppose we did not realize we can do inversion counting and sorting at the same time. One can still do the sorting within the merging step. The final algorithm will look like:

```
merge_count(b[], c[])
   Sort b[]
   Sort c[]
   i = 1, j = 1
   count = 0
   WHILE i <= Length(b[]) OR j <= Length(c[])
     IF b[i] < c[j] THEN
        i = i + 1
        count = count + (j - 1)
     ELSE
        j = j+1
   RETURN count

count_inversion(a[])
   IF length(a) < 2 THEN RETURN 0
   Break a[] into two halves b[], c[]
   RETURN count_inversion(b) + count_inversion(c) + merge_count(b,c)
```

Suppose $Sort$ in the code takes $\Theta(n \log n)$ time. Let $T(n)$ be the running time of this algorithm, write a recursion for $T(n)$ and solve it.

Problem 2 (Recursion-II)**.** (10 points) Please solve the following recursion (write the answer in asymptotic notations $T(n) = \Theta(f(n))$).

If you decide to use the recursion tree method, you **do not** need to draw the tree. Just describe what the tree looks like in the second layer (e.g. there are a subproblems each with size n/b), bound the amount of work in each layer, and take the sum over all layers. You do not need to write the induction proof if you are using the recursion tree method.

$$T(n) = T(n/2) + 2T(n/4) + n, T(1) = 0.$$

Problem 3 (Binary Search)**.** Binary search is a classical algorithm. Given an array $A[1..n]$ sorted in ascending order, binary search can find whether an element b is in the array A. The algorithm works as follows:

```
binary_search(A[1..n], b)
   If n <= 2 then check whether b is in A by looking through all elements.
   Let k = n/2
   Partition A into B, C where B contains A[1..k-1], and C contains A[k+1..n]
   If A[k] == b then b is in array A
   If A[k] > b then call binary_search(B, b)
   If A[k] < b then call binary_search(C, b)
```

(Note: you can also describe this algorithm as: When length of A is at least 3, compare b against the middle element in A, if b is larger then search in the right half of A, if b is smaller then search in the left half of A.)

(a) (5 points) Analyze the running time of the binary search algorithm (same requirement as Problem 1).

(b) (15 points) Suppose the array is very long (n very large), but we have the additional information that the number we are trying to find appears early in the array (that is, we are trying to find a number $A[p]$ where p is a number much smaller than n). Design an algorithm whose running time is $O(\log p)$. (Hint: First you want to find a index q such that $q/2 < p \leq q$.)

Problem 4 (Magic Square)**.** (20 points) A magic-square is a $n \times n$ array where every number from 1 to n^2 appears once in the array. The sum of numbers in every row, column and diagonal of the array is the same. See below for a 3×3 magic-square:

8	1	6
3	5	7
4	9	2

Suppose $n = 3^k$ for some integer $k \geq 1$, design an algorithm that outputs a $n \times n$ magic square. Analyze the running time of your algorithm.

(There are many ways to solve this problem, but please try to use divide-and-conquer.)