# COMPSCI330 Design and Analysis of Algorithms
# Assignment 1

Due Date: Wednesday, September 13, 2017

## 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 each problem as an *individual* pdf file for the correct problem on Sakai. 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 submit within one week of the deadline you will get half credit. Any submission after that will not receive any credit.
- **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** (Recursions)**.** Please solve the following recursions (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.

(a) (10 points)

$$T(n) = T(\frac{n}{2}) + 2T(\frac{n}{4}) + n^2.$$

(b) (10 points)

$$T(n) = n^{1/3}T(n^{2/3}) + n.$$

(Note: You cannot use Master Theorem here because $n^{1/3}, n^{2/3}$ are not constants.)

**Problem 2** (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) (8 points) Analyze the running time of the binary search algorithm.

(b) (12 points) Using similar ideas, you are going to solve a related problem. In this problem you are given an array $A[1..n]$ that is first decreasing and then increasing. More precisely, there is a coordinate $1 \le p \le n$ such that for all $i < p$, $A[i] > A[i+1]$, and for all $i \ge p$, $A[i] < A[i+1]$. Your goal is to find the smallest element in this array. Please design an algorithm that has the same asymptotic running time as binary search.

**Problem 3** (Garden Decorations)**.** (20 points) Alice is going to decorate her garden using $L$ shaped tiles. Her garden has a square shape. The size of the garden is $n \times n$, where $n$ is a power of 2 ($n = 2^k$ for some integer $k \ge 1$). The tiles she is going to use are $L$ shaped with 3 $1 \times 1$ squares (see Figure 1). She insists that all the tiles needs to be aligned with the four sides of the garden, although they can be rotated by multiples of 90 degrees. There is a tree in her garden at location $(x, y)$. The tiles cannot overlap with each other or the tree. Now you are asked to design an algorithm that outputs a way to put the tiles (see Figure for an example where $n = 4, x = 2, y = 2$). (Of course, your algorithm does not need to draw the picture, it just needs to describe where to place the tiles.)

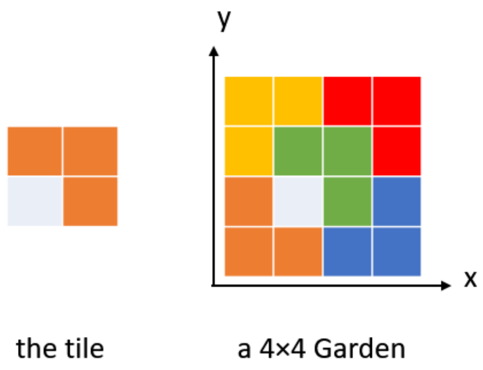

Figure 1: The tile and a sample garden