CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Homework #1

January 26, 2021

Name: Ryan Coslove *Extension: No*

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question. This however does not apply to the extra credit questions.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (video recordings, notes, textbook, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $\Theta(n \log n)$ time using merge sort".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question). See "Practice Homework" for an example.
- The "Challenge yourself" and "Fun with algorithms" are both extra credit. These problems are significantly more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.

Problem 1. This question reviews asymptotic notation. You may assume the following inequalities for this question (and throughout the course): For any constant $c \geq 1$,

$$(\log n)^c = o(n) \quad , \quad n^c = o(n^{c+1}) \quad , \quad c^n = o((c+1)^n) \quad , \quad (n/2)^{(n/2)} = o(n!) \quad , \quad n! = o(n^n).$$

(a) Rank the following functions based on their asymptotic value in the increasing order, i.e., list them as functions $f_1, f_2, f_3, \ldots, f_9$ such that $f_1 = O(f_2)$, $f_2 = O(f_3), \ldots, f_8 = O(f_9)$. Remember to write down your proof for each equation $f_i = O(f_{i+1})$ in the sequence above. **(15 points)**

$\sqrt{\log n}$	$\log \log n$	$2^{\log n}$
$100n$	10^n	$2^{2^{2^{2^2}}}$
2^n	$n!$	$\dfrac{n}{\log n}$

Hint: For some of the proofs, you can simply show that $f_i(n) \leq f_{i+1}(n)$ for all sufficiently large n which immediately implies $f_i = O(f_{i+1})$.

Solution.

$f_1 = 2^{2^{2^{2^2}}}, f_2 = \log \log n \quad \lim_{n \to \infty} \frac{2^{2^{2^{2^2}}}}{\log \log n} = 0$ so $f_1 = O(f_2)$

$f_2 = \log\log n, f_3 = \sqrt{\log n} \quad m = \log n \quad \log m = O(\sqrt{m}) \quad => (\log m)^2 = O(m)$ so $f_2 = O(f_3)$

$f_3 = \sqrt{\log n}, f_4 = 2^{\log n} \quad m = \log n \quad m = O(2^m)^2 \quad$ so $f_3 = O(f_4)$

$f_4 = 2^{\log n}, f_5 = \frac{n}{\log n} \quad m = \log n \quad \lim_{n\to\infty} \frac{2^m}{\frac{n}{m}} = \frac{m2^m}{n} = 0 \quad$ so $f_4 = O(f_5)$

$f_5 = \frac{n}{\log n}, f_6 = 100n \quad \lim_{n\to\infty} \frac{\frac{n}{\log n}}{100n} = \frac{100n^2}{\log n} = 0 \quad$ so $f_5 = O(f_6)$

$f_6 = 100n, f_7 = 2^n \quad \lim_{n\to\infty} \frac{100n}{2^n} = 0 \quad$ so $f_6 = O(f_7)$

$f_7 = 2^n, f_8 = 10^n \quad 2^n = O(10^n) \quad$ rule: $c^n = o((c+1)^n) \quad$ so $f_7 = O(f_8)$

$f_8 = 10^n, f_9 = n! \quad$ rule: $n! = o(n^n) \quad n! \geq O(10^n)$ so $f_8 = O(f_9)$

Ranked in increasing order: $f_1 = 2^{2^{2^{2^2}}}, f_2 = \log\log n, f_3 = \sqrt{\log n}, f_4 = 2^{\log n}, f_5 = \frac{n}{\log n}$

$f_6 = 100n, f_7 = 2^n, f_8 = 10^n, f_9 = n!$

(b) Consider the following four different functions $f(n)$:

$$1 \qquad \log n \qquad n^2 \qquad 4^{4^n}.$$

For each of these functions, determine which of the following statements is true and which one is false. Remember to write down your proof for each choice. **(10 points)**

- $f(n) = \Theta(f(n-1))$;
- $f(n) = \Theta(f(\frac{n}{2}))$;
- $f(n) = \Theta(f(\sqrt{n}))$;

Example: For the function $f(n) = 4^{4^n}$, we have $f(n-1) = 4^{4^{n-1}}$. Since $4^{4^{n-1}} = 4^{\frac{1}{4}\cdot 4^n} = (4^{4^n})^{1/4}$.

$$\lim_{n\to\infty} \frac{f(n)}{f(n-1)} = \lim_{n\to\infty} \frac{4^{4^n}}{4^{4^{n-1}}} = \lim_{n\to\infty} \frac{4^{4^n}}{(4^{4^n})^{1/4}} = \lim_{n\to\infty} (4^{4^n})^{3/4} = +\infty.$$

As such, $f(n) \neq O(f(n-1))$ and thus the first statement is false for 4^{4^n}.

Solution.

$f(n) = 1, f(n-1) = 1$

$\lim_{n\to\infty} \frac{f(n)}{f(n-1)} = \lim_{n\to\infty} \frac{1}{1} = 1$ therefore $0 < 1 < \infty$

So $f(n) = \Theta(f(n-1))$ and the first statement is true for 1

$f(n) = 1, f(\frac{n}{2}) = 1$

$\lim_{n\to\infty} \frac{f(n)}{f(\frac{n}{2})} = \lim_{n\to\infty} \frac{1}{1} = 1$ therefore $0 < 1 < \infty$

So $f(n) = \Theta(f(\frac{n}{2}))$ and the second statement is true for 1

$f(n) = 1, f(\sqrt{n}) = 1;$

$$\lim_{n\to\infty} \frac{f(n)}{f(\sqrt{n})} = \lim_{n\to\infty} \frac{1}{1} = 1 \text{ therefore } 0 < 1 < \infty$$

So $f(n) = \Theta(f(\sqrt{n}))$ and the third statement is true for 1

$f(n) = \log n, f(n-1) = \log(n-1)$

$$\lim_{n\to\infty} \frac{\log n}{\log(n-1)} = \lim_{n\to\infty} \frac{\log n}{\log(1-\frac{1}{n})+\log n} = 1 \text{ therefore } 0 < 1 < \infty$$

So $f(n) = \Theta(f(n-1))$ and the first statement is true.

$f(n) = \log n, f(\frac{n}{2}) = \log(\frac{n}{2})$

$$\lim_{n\to\infty} \frac{\log n}{\log(\frac{n}{2})} = \lim_{n\to\infty} \frac{\log n}{\log n - \log 2} = 1$$

So $f(n) = \Theta(f(\frac{n}{2}))$ and the second statement is true.

$f(n) = \log n, f(\sqrt{n}) = \log(\sqrt{n}) = \frac{1}{2}\log n$

$$\lim_{n\to\infty} \frac{\log n}{\frac{1}{2}\log n} = \lim_{n\to\infty} 2 * 1 = 2$$

So $f(n) = \Theta(f(\sqrt{n}))$ and the third statement is true.

$f(n) = n^2, f(n-1) = (n-1)^2 = n^2 - 2n + 1$

$$\lim_{n\to\infty} \frac{n^2}{n^2-2n+1} = \lim_{n\to\infty} \frac{2n}{2n*2} = \lim_{n\to\infty} \frac{2}{2} = 1$$

So $f(n) = \Theta(f(n-1))$ and the first statement is true.

$f(n) = n^2, f(\frac{n}{2}) = (\frac{n}{2})^2 = \frac{n^2}{4}$

$$\lim_{n\to\infty} \frac{n^2}{\frac{n^2}{4}} = 4$$

So $f(n) = \Theta(f(n^2))$ and the second statement is true.

$f(n) = n^2, f((\sqrt{n})^2) = n$

$$\lim_{n\to\infty} \frac{n^2}{n} = \lim_{n\to\infty} n = \infty$$

So $f(n) \neq \Theta(f(\sqrt{n}))$ and the third statement is false.

$f(n) = 4^{4^n}, f(n-1) = 4^{4^{(n-1)}}$

was done in the example

$f(n) = 4^{4^n}, f(\frac{n}{2}) = 4^{4^{(\frac{n}{2})}}$

$$\lim_{n\to\infty} \frac{4^{4^n}}{4^{4^{(\frac{n}{2})}}} = \infty$$

So $f(n) \neq \Theta(f(\frac{n}{2}))$ and the second statement is false.

$f(n) = 4^{4^n}, f(\sqrt{n}) = 4^{4^{\sqrt{n}}}$

$$\lim_{n\to\infty} \frac{4^{4^n}}{4^{4^{\sqrt{n}}}} = \infty$$

So $f(n) \neq \Theta(f(\sqrt{n}))$ and the third statement is false.

Problem 2. Your goal in this problem is to analyze the *runtime* of the following (imaginary) recursive algorithms for some (even more imaginary) problem:

(A) Algorithm A divides an instance of size n into 4 subproblems of size $n/4$ each, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

Solution.

$T(n) = 4T(\frac{n}{4}) + On)$

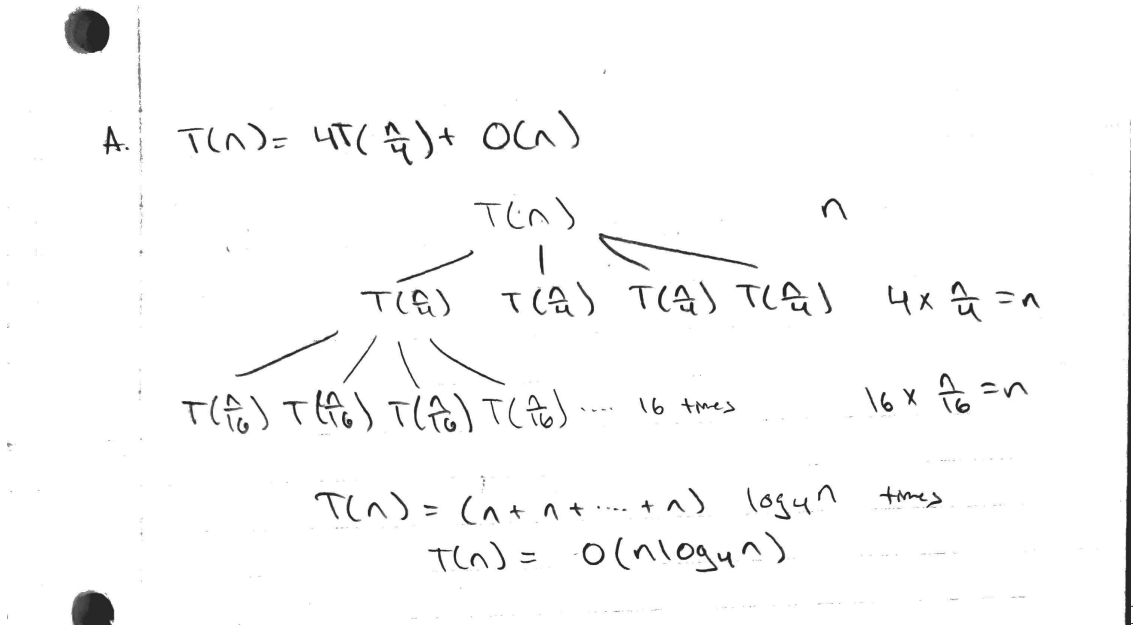
$T(n) = O(n \log_4 n)$

(B) Algorithm B divides an instance of size n into 2 subproblems, one with size $n/4$ and one with size $n/5$, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

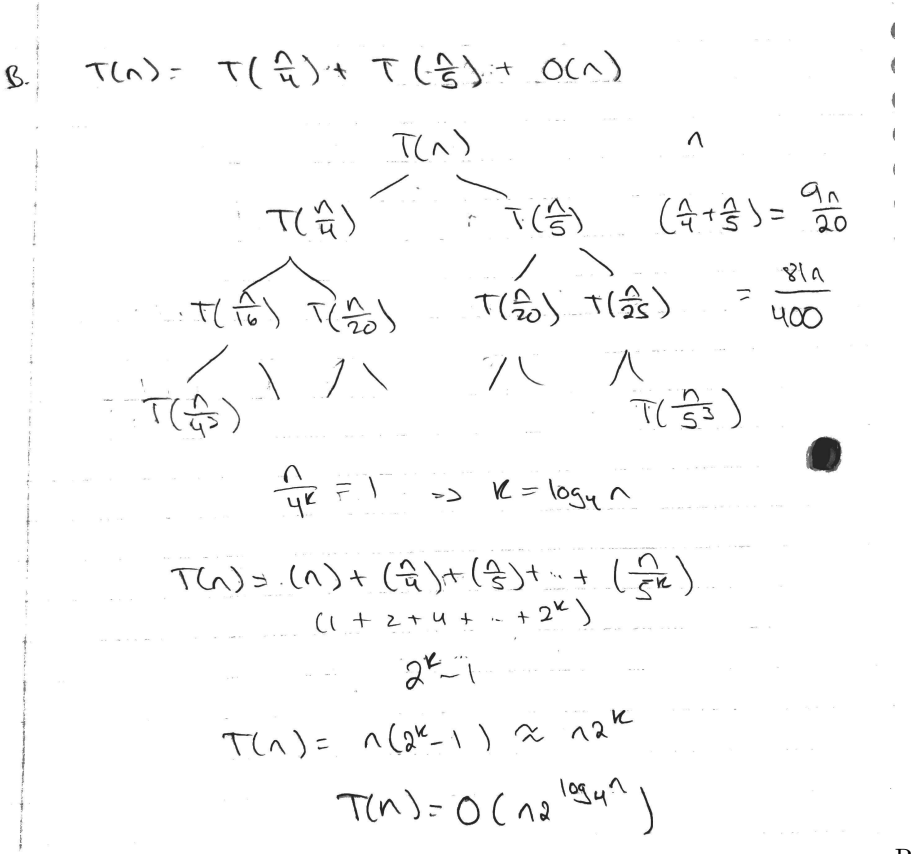
Solution.

$T(n) = T(\frac{n}{4}) + T(\frac{n}{5}) + O(n)$

$T(n) = O(n * 2^{\log_4 n})$



Recursion tree for algorithm A.



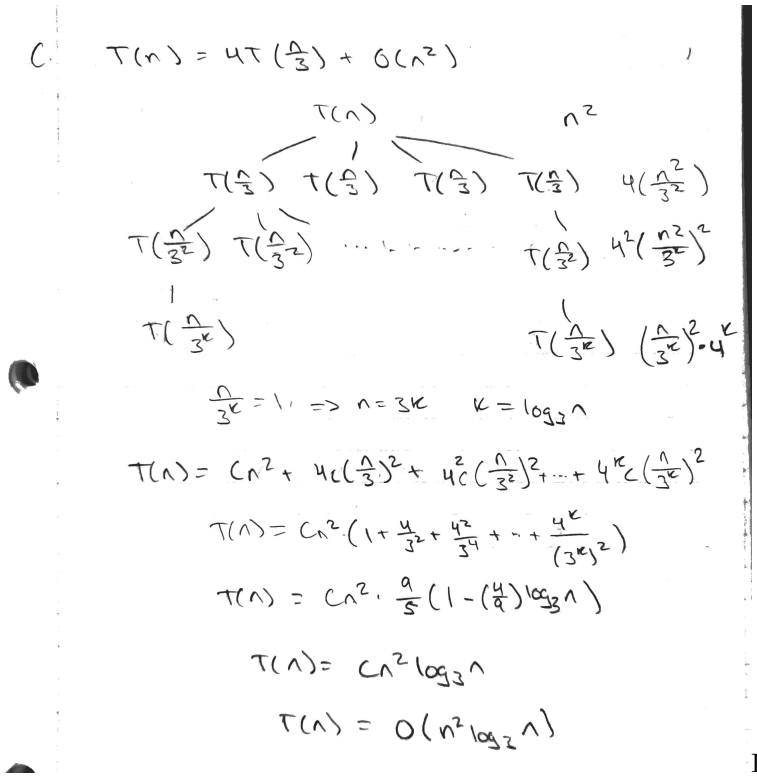
Recursion tree for algorithm B.

(C) Algorithm C divides an instance of size n into 4 subproblems of size $n/3$ each, recursively solves each one, and then takes $O(n^2)$ time to combine the solutions and output the answer.

Solution.

$T(n) = 4T(\frac{n}{3}) + O(n^2)$

$T(n) = O(n^2 \log_3 n)$



Recursion tree for algorithm C.

(D) Algorithm D divides an instance of size n in to 2 subproblems of size $n-1$ each, recursively solves each one, and then takes $O(1)$ time to combine the solutions and output the answer.

Solution.

$T(n) = 2T(n-1) + O(1)$

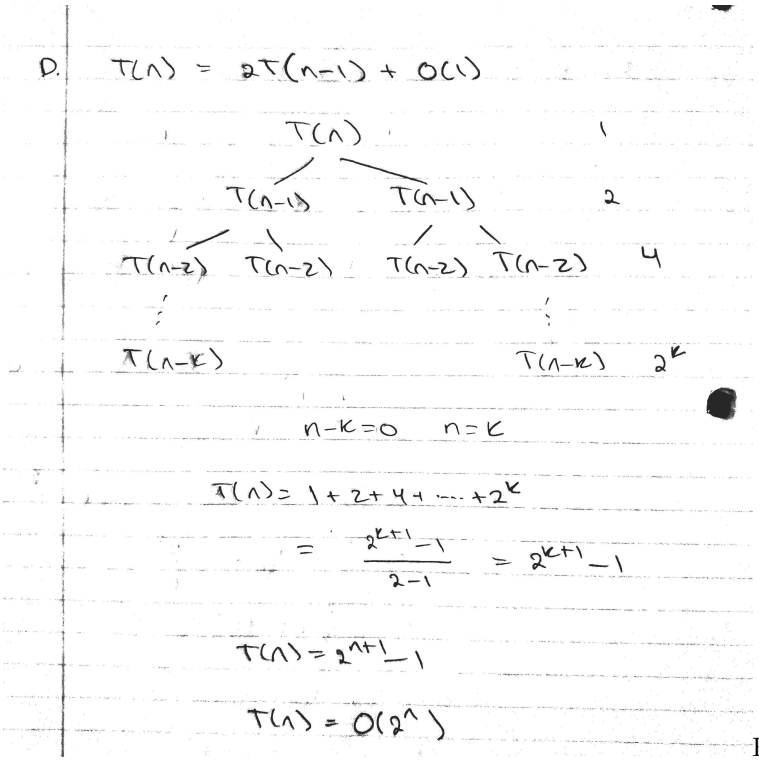
$T(n) = O(2^n)$

For each algorithm, write a recurrence for its runtime and *use the recursion tree method* of Lecture 3 to solve this recurrence and find the *tightest asymptotic* upper bound on runtime of the algorithm. **(25 points)**

Problem 3. In this problem, we consider a non-standard sorting algorithm called the *Silly Sort*. Given an array $A[1:n]$ of n integers, the algorithm is as follows:

• **Silly-Sort**$(A[1:n])$:

1. If $n < 5$, run merge sort (or selection sort or insertion sort) on A.
2. Otherwise, run **Silly-Sort**$(A[1:3n/4])$, **Silly-Sort**$(A[n/4:n])$, and **Silly-Sort**$(A[1:3n/4])$ again.



Recursion tree for algorithm D.

We now analyze this algorithm.

(a) Prove the correctness of **Silly-Sort**. **(10 points)**

Solution. Solution to part (a) goes here.

(b) Write a recurrence for **Silly-Sort** and use the recursion tree method of Lecture 3 to solve this recurrence and find the *tightest asymptotic* upper bound on the runtime of **Silly-Sort**. **(10 points)**

Solution. Solution to part (b) goes here.

(c) Suppose we like to change the second line of the algorithm to

$$\textbf{Silly-Sort}(A[1:m]),\ \textbf{Silly-Sort}(A[n-m:n]),\ \text{and}\ \textbf{Silly-Sort}(A[1:m])$$

for some other value of m instead (in the original algorithm, $m = 3n/4$).

What is the smallest number we could pick while still maintaining the correctness of the algorithm? What would be the runtime of the resulting algorithm? For this part of the question, you can simply write a few lines for proof of correctness and runtime analysis by pointing out how your proofs and calculations in parts (a) and (b) should be changed. **(5 points)**

Solution. Solution to part (c) goes here.

Problem 4. You are given an array $A[1:n]$ which includes the scores of n players in a game. They are ranked in the following way: Rank of a player is an integer r if there are exactly $r-1$ *distinct* scores strictly smaller than the score of this player (irrespective of the number of players).

(a) Design and analyze an algorithm that given the array A, can find the rank of all players in the array in $O(n \log n)$ time. **(15 points)**

Example. Suppose the input array is $A = [1, 7, 6, 5, 2, 4, 5, 2]$ for 8 players; then the rank of players is:

- Player $A[1]$ has rank 1 (as $A[1] = 1$ is the smallest number);
- Player $A[2]$ has rank 6 (as $A[2] = 7$ has 5 distinct smaller numbers: $\{1, 6, 5, 4, 2\}$);
- Player $A[3]$ has rank 5 (as $A[3] = 6$ has 4 distinct smaller numbers: $\{1, 5, 4, 2\}$);
- Player $A[4]$ has rank 4 (as $A[4] = 5$ has 3 distinct smaller numbers: $\{1, 4, 2\}$);
- Player $A[5]$ has rank 2 (as $A[5] = 2$ has 1 distinct smaller number: $\{1\}$);
- Player $A[6]$ has rank 3 (as $A[6] = 4$ has 2 distinct smaller numbers: $\{1, 2\}$);
- Player $A[7]$ has rank 4 (as $A[7] = 5$ has 3 distinct smaller numbers: $\{1, 4, 2\}$);
- Player $A[8]$ has rank 2 (as $A[8] = 2$ has 1 distinct smaller number: $\{1\}$);

Solution.

Using a quick sort algorithm

1. If n = 0 or n =1 return the current array

2. Let p = n

3. Use partition algorithm to partitiion A with pivot p. Let q be the correct position of A[p] in the sorted array computed by the partiton algorithm.

Recursively quick sort A[1: q-1] and A[q+1:n]

Make the nth value the pivot p.

pick (q - 1) as the pivot in the first recursive call for A[1: q-1] and n as pivot in the second recursive call for A[q+1:n]

repeat this process until we sort the entire array

Proof by induction

let A be array of size n =k+1. Because after the recursions of every element in A[1:q-1] which is at most large as A[q], and A[q+1:n] which is at least as large as A[q], both are smaller than k+1. By the guarantee of the partition algorithm and this sorting step, we have A[1 :q-1] is sorted array of elements $\leq$A[q], and A[q+ 1 :n] is the sorted array of elements $\geq$ A[q]; hence A is also now sorted.

Runtime

Partioning the array takes O(n) time. Picking the correct position q as pivot, we recursively sort arrays A[1: q-1] and A[q+1:n]. T(n) denotes worst-case runtime.

$$T(n) \leq T(q-1) + T(n-q) + O(n)$$

$$T(n) \leq T(n-1) + O(n) => T(n) = O(n^2)$$

$$T(n) \leq 2T(n/2) + O(n)$$

$$T(n) = O(n \log n)$$

(b) Suppose you are additionally given an array $B[1 : m]$ with the score of m new players. Design and analyze an algorithm that given both arrays A and B, can find the rank of each player B inside the array A, i.e., for each $B[i]$, determines what would be the rank of $B[i]$ in the array consisting of all elements of A plus $B[i]$. Your algorithm should run in $O((n + m) \cdot \log n)$ time. **(10 points)**

Example. Suppose the input array is $A = [1, 7, 6, 5, 2, 4, 5, 2]$ as before and $B = [3, 9, 4]$; then the correct answer for each player in B is:

- Player $B[1]$ will have rank 3 (as $B[1] = 3$ has 2 distinct smaller numbers in A: $\{1, 2\}$);
- Player $B[2]$ will have rank 7 (as $B[2] = 9$ has 6 distinct smaller numbers in A: $\{1, 7, 6, 5, 4, 2\}$);
- Player $B[3]$ will have rank 3 (as $B[3] = 4$ has 2 distinct smaller numbers in A: $\{1, 2\}$);

Solution.

We use the same algorithm as part A, except we change the steps following the recursions. Once the recursions have taken place, we can run a for loop to iterate through the B array against the sorted A array. We would have the following to compare the B array to the A array:

$for(i = 0; i \leq m, i++)$

$setB[i] = A[i]$

$B[i]++$

Proof by induction

the proof for Array A still applies. For Array B, let B be the size of m = k+1. After recursing through the B array like we did the A array, we see the subarrays are smaller than k+1 and allows B to be sorted. The for loop that is used is of size m, which is smaller than m+1. Hence the arrays are sorted and have been compared.

Runtime

Partioning the array takes O(n) time. Running the for loop takes O(m) time. Picking the correct position q as pivot, we recursively sort arrays A[1: q-1] and A[q+1:n]. T(n) denotes worst-case runtime.

$$T(n) \leq T(q-1) + T(n-q) + O(m) + O(n)$$

$$T(n) \leq 2T(n/2) + O(m) + O(n)$$

$$T(n) = O((n + m) * \log n)$$

Challenge Yourself. Let us revisit the community detection problem but with an interesting twist. Remember that we have a collection of n people for some odd integer n and we know that strictly more than half of them belong to a hidden community. As before, when we introduce two people together, the members of the hidden community would say they know the other person if they also belong to the community, and otherwise they say they do not know the other person. The twist is now as follows: the people that do not belong to the community *may lie*, meaning that they may decide to say they know the other person even though in reality only people inside the hidden community know each other.

Concretely, suppose we introduce two people A and B, then what they will say would be one of the following (first part of tuple is the answer of A and second part is the answer of B):

- if both belong: (*know* , *know*);
- if A belongs and B does not: (*does not know* , *know/does not know*);
- if B belongs and A does not: (*know/does not know* , *does not know*);
- if neither belongs: (*know/does not know* , *know/does not know*);

Design an algorithm that finds all members of the hidden community using $O(n)$ greetings. **(+10 points)**

Fun with Algorithms. We have an n-story building and a series of magical vases that work as follows: there is some unknown level L in the building that if we throw these vases down from any of the levels $L, L+1, \ldots, n$, they will definitely break; however, no matter how many times we throw the vases down from any level below L nothing will happen them. Our goal in this question is to determine this level L by throwing the vases from different levels of the building (!).

For each of the scenarios below, design an algorithm that uses asymptotically the smallest number of times we throw a vase (so the measure of efficiency for us is the number of vase throws).

(a) When we have only one vase. Once we break the vase, there is nothing else we can do. **(+2 points)**

Solution. Solution to part (a) goes here.

(b) When we have four vases. Once we break all four vases, there is nothing else we can do. **(+4 points)**

Solution. Solution to part (b) goes here.

(c) When we have an unlimited number of vases. **(+4 points)**

Solution. Solution to part (c) goes here.
