Illinois Institute of Technology
Department of Computer Science

First Examination

CS 430 Introduction to Algorithms
Spring, 2018

Wednesday, February 7, 2018
10am–11:15am & 11:25am–12:40pm
111 Robert A. Pritzker Science Center

Print your name and student ID, *neatly* in the space provided below; print your name at the upper right corner of *every* page. Please print legibly.

Name:	
Student ID:	

This is an *open book* exam. You are permitted to use the textbook, any class handouts, anything posted on the web page, any of your own assignments, and anything in your own handwriting. Foreign students may use a dictionary. *Nothing else is permitted*: No calculators, laptops, cell phones, Ipods, Ipads, etc.!

Do all five problems in this booklet. *All problems are equally weighted, so do not spend too much time on any one question.*

Show your work! You will not get partial credit if the grader cannot figure out how you arrived at your answer.

Question	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Time Bounds**

   Suppose you are given five algorithms with the following exact running times in terms of the input size n:

   (a) n^2
   (b) n^3
   (c) $100n$
   (d) $n\lceil \lg n \rceil$
   (e) 2^n

   How much *faster* do these algorithms run when you

   (a) Halve the size of the input?
   (b) Decrease the size of the input by 1?

2. **Randomized Max**

   Consider the following algorithm to find the maximum element in an unordered array of n elements, $A[1..n]$:

```
1: RandomMax($A[1..n]$)
2: $m \leftarrow -\infty$
3: for $i \leftarrow 1$ to $n$ in random order do
4:    if $A[i] > m$ then
5:       $m \leftarrow A[i]$
6:    end if
7: end for
8: return $m$
```

   Notice that the **for** loop (line 3) takes the numbers $1, 2, \ldots, n$ in random order.

   (a) In the worst case, how many times does RandomMax execute line 5?

   (b) What is the probability that nth (final) iteration executes line 5?

   (c) Analyze the expected number of executions of line 5.

3. **Unlucky Quicksort**

   Suppose your implementation of quicksort (as in CLRS3, page 171) always gets a partitioning element at the extremes of the sorted values. In particular, the partitioning element is always either the largest or smallest element. Analyze the time required of quicksort in these circumstances, assuming (as we did in class) that all elements are distinct.

4. **A Strange Sorting Algorithm**

   The following algorithm is supposed to sort part of an array of distinct items, $A[i..j]$, that is, positions $i \ldots j$ of array A, in which no two of which are equal:

```
StrangeSort($A[i..j]$)
if $j > i$ then
    if $A[i] > A[j]$ then
        swap $A[i]$ and $A[j]$
    end if
    StrangeSort($A[i+1..j-1]$)
    if $A[i] > A[i+1]$ then
        swap $A[i]$ and $A[i+1]$
    end if
    StrangeSort($A[i+1..j]$)
end if
```

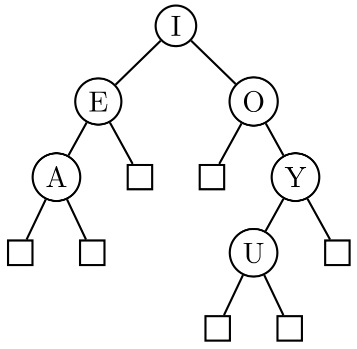
   (a) Verify that StrangeSort properly sorts for the cases $j = i + 1$ and $j = i + 2$.

   (b) Using the previous part as the basis of induction, prove by induction on $j - i$ that, indeed, StrangeSort correctly sorts $A[i..j]$ for all $i < j$.

   (c) Write and solve a recurrence for the time required for the execution of StrangeSort($A[1..n]$).

5. **Binary Search Trees**

   Given the following binary search tree on English vowels, in which internal nodes are shown as circles and external nodes (leaves; nil pointers) are shown as small boxes:



   (a) Calculate the average successful search time (number of probes), assuming that the letters are equally probable.

   (b) Calculate the worst-case unsuccessful search time (number of probes).

   (c) Is there a lexicographic binary search tree on the same set of letters, {A, E, I, O, U, Y}, that has better worst-case unsuccessful search time? Either give such a tree or prove it does not exist.