Illinois Institute of Technology
Department of Computer Science

# Solutions to 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

## Exam Statistics

112 students took the exam; there were 4 no-shows recorded as 0. The range of scores was 17–91, with a mean of 59.01 (this excludes the no-shows), a median of 61, and a standard deviation of 15.91. Very roughly speaking, if I had to assign final grades on the basis of this exam only, above 74 would be an A (22), 60–74 a B (38), 45–59 a C (31), 30–44 a D (17), below 30 an E (4). Every student should have been able to get almost full credit on the first, second, third, and fifth problems, plus a few points on problem four; no score should have been below 60.

## Problem Solutions

1. For a decrease by half:

   (a) $n^2$ becomes $(n/2)^2 = n^2/4$ so the algorithm runs 4 times as fast.

   (b) $n^3$ becomes $(n/2)^3 = n^3/8$ so the algorithm runs 8 times as fast.

   (c) $100n$ becomes $100 \cdot (n/2) = 50n$ so the algorithm runs twice as fast.

   (d) $n\lceil \lg n \rceil$ becomes $(n/2)\lceil \lg(n/2) \rceil = (n\lceil \lg n \rceil)/2 - n/2$ so the algorithm runs a bit more than twice as fast as $n$ gets large.

   (e) $2^n$ becomes $2^{n/2} = \sqrt{2^n}$ so the algorithm runs in the *square root* of the original time.

   For a decrease by 1:

   (a) $n^2$ becomes $(n-1)^2 = n^2 - 2n + 1$ so the algorithm saves $(2n-1)$ units of time.

   (b) $n^3$ becomes $(n-1)^3 = n^3 - 3n^2 + 3n - 1$ so the algorithm saves $(3n^2 - 3n + 1)$ units of time.

   (c) $100n$ becomes $100(n-1) = 100n - 100$ so the algorithm saves 100 units of time.

   (d) $n\lceil \lg n \rceil$ becomes $(n-1)\lceil \lg(n-1) \rceil \approx n\lceil \lg n \rceil - \lceil \lg n \rceil$ so the algorithm saves about $\lg n$ units of time.

   (e) $2^n$ becomes $2^{n-1} = 2^n/2$ so the algorithm runs twice as fast.

2. (a) Every time; that is, $n$ times.

(b) $1/n$ because each of the $n$ indices is equally likely to be the last in the random order, including the index at which the maximum element occurs.

(c) From the previous part, the expected number of executions is given by $e_n = e_{n-1} + 1/n$, $e_1 = 1$. Thus $e_n$ is the $n$th harmonic number $H_n = 1 + 1/2 + \cdots + 1/n = \ln n + o(1)$.

3. This is *exactly* the analysis of section 7.4 in CLRS3! The answer is $T(n) = \Theta(n^2)$.

4. (a) When $j = i+1$, and $A[i]$ and $A[i+1]$ are out of order (that is, $A[i] > A[i+1]$), they are swapped at line 4; line 5 is a call with $i > j$ which just returns. Line 6 finds $A[i]$ and $A[i+1]$ in order so line 7 is not executed. Line 9 is a call with $i = j$ which just returns. That is, when called with adjacent elements, they are left in order.

   When $j = i+2$, and $A[i]$ and $A[i+2]$ are out of order (that is, $A[i] > A[i+2]$), they are swapped at line 4 and hence in order when we get to line 5. Line 5 is a call with $i = j$ which just returns. If $A[i]$ and $A[i+1]$ are out of order, they are swapped at line 7. So when we get to line 9 we have $A[i]$ is the smallest of the three elements $A[i..i+2]$. Line 9 then puts $A[i+1]$ and $A[i+2]$ in order, so the three elements $A[i..i+2]$ are now sorted.

(b) For the inductive proof when $j - i > 2$, suppose StrangeSort correctly sorts intervals shorter than $j - i$. The call with $i$, $j$, $j - i > 2$, puts $A[i]$ and $A[j]$ in order; by induction, the recursive call at line 5 puts $A[i+1..j-1]$ in order so that $A[i+1]$ is the smallest in $A[i+1..j-1]$; lines 6–7 put the smallest element in $A[i..j-1]$ in $A[i]$ and we already had it smaller than $A[j]$; hence $A[i]$ is the smallest among $A[i..j]$. Again by induction, the recursive call at line 9 then sorts $A[i+1..j]$. This leaves the entire array sorted.

(c) The time to sort $n$ items by this bizarre method, $T(n)$, is thus given by the recurrence $T(n) = O(1) + T(n-2) + T(n-1)$, where $T(0)$ $T(1)$, and $T(2)$ are all $O(1)$. This is a variant of the Fibonacci recurrence that we solved in class; it has annihilator $(E-1)(E^2 - E - 1)$, so $T(n) = \Theta(\phi^n)$, where $\phi = (1+\sqrt{5})/2$ is the Golden Ratio. Imagine, an exponential time sorting algorithm.

5. (a) The average successful search time is the one plus average depth of an internal node: $1 + (2+1+0+1+3+2)/6 = 2.5$ probes. That is, searching for A takes 3 probes, for E takes 2 probes, etc.

(b) The worst case unsuccessful search is at either of the two leaves below U, that is, 4 probes when we search for any of the letters P, Q, R, S, T, V, W, or X.

(c) Yes, there are better (lexicographic) trees that have worst-case unsuccessful search time 3. For example,