Illinois Institute of Technology
Department of Computer Science

Solutions to First Examination

CS 430 Introduction to Algorithms
Fall, 2014

Monday, September 29, 2014
10am–11:15pm, 121 Life Sciences
11:25am–12:40pm, 113 Stuart Building

Exam Statistics

95 students took the exam. 3 students were absent; their zeros are not counted in the statistics that follow. The range of scores was 6–90, with a mean of 38.1, a median of 34, and a standard deviation of 19.3. Very roughly speaking, if I had to assign final grades on the basis of this exam only, above 60 would be an A (17), 34–59 a B (33), 20–33 a C (29), 10–19 a D (14), below 10 an E (2).

Problem Solutions

1. (a) $T(n) = T(n) + 100$ means that $T(n) = \Theta(n)$, so halving the size of the input cuts the time to half of the original time.
   (b) $T(n) = T(n-1) + n$ means that $T(n) = \Theta(n^2)$, so halving the size of the input cuts the time to one quarter of the original time.
   (c) $T(n) = T(n-1) + T(n-2)$ means that $T(n) = \Theta(\phi^n)$ where $\phi = (1+\sqrt{5})/2 \approx 1.618034$ is the golden ratio, so halving the size of the input cuts the time to $\Theta(\phi^{n/2}) = \sqrt{\phi^n}$; thus the algorithm runs in the *square root* of the original time.
   (d) $T(n) = T(n-1) + 1/n$ means that $T(n) = \Theta(H_n) = \Theta(\log n)$, so halving the size of the input cuts the time by only a small constant amount because $\Theta(\log \frac{n}{2}) = \Theta(\log n) - O(1)$.

2. Following the hint, begin by comparing x to A_{1m}. If $x = A_{1m}$, we have found x. Otherwise, if $x < A_{1m}$, x cannot be in column m, so we step left in row 1, continuing recursively at $A_{1,m-1}$; if $x > A_{1m}$, x cannot be in row 1, so we step down in column m, continuing recursively at A_{2m}. In either of the unequal cases, we have eliminated a column or a row, respectively, from consideration: in the worst case we zig-zag down to the lower left entry, A_{n1} making a worst-case total of $n + m - 1$ comparisons (steps).

3. In Lecture 4 (September 8), the first lecture on Quicksort, Professor Reingold specifically mentioned that Section 7.4.2 in CLRS3 was "a gold mine for exam problems." You were warned!

   (a) If z_1 is the first pivot element, it is compared to the remaining $n - 1$ elements; this is clearly the largest possible.

(b) If z_2 is the first pivot element, it is compared to z_1, after which z_1 is alone in its partition, so z_1 is never again compared to any other element; this is clearly the smallest possible.

(c) Using equation (7.3) on page 183 of CLRS3, we have $i = 1$ and $j = 2$, so the probability is 1. Or, informally, we could argue that they *must* be compared for z_1 to end up to the left of z_2.

(d) Again, using equation (7.3) on page 183 of CLRS3, we have $i = 1$ and $j = 3$, so the probability is 2/3.

(e) As in the discussion of section 7.4.2 of CLRS3, the expectation of the sum is the sum of the expectation, so the expected number of comparisons involving z_1 is the sum of equation (7.3) for $i = 1$ and $j = 2 \ldots n$, that is, $2H_n - 1 \approx 2 \ln n$.

4. Below and on the next page is code that implements the insert operation. We insert the key at the next available spot in the heap and we "bubble" it upward. We call one of two different bubble up routines depending on the level in the heap at which we perform the initial insert, and depending on the inserted key's value relative to the keys of its parent and grandparent (the only difference between the two routines is the way we compare $H[i]$ in line 1).

Algorithm 1 INSERTMINMAX(k, H)

```
 1: SIZE(H) = SIZE(H) + 1
 2: i = SIZE(H)
 3: H[i] = k
 4: if i is on a max level then
 5:    if PARENT(i) > 0 and H[PARENT(i)] > H[i] then
 6:       exchange H[i] with H[PARENT(i)]
 7:       BUBBLEUPMIN(PARENT(i), H)
 8:    else
 9:       BUBBLEUPMAX(i, H)
10:    end if
11: else
12:    {i is on a min level}
13:    if PARENT(i) > 0 and H[PARENT(i)] < H[i] then
14:       exchange H[i] with H[PARENT(i)]
15:       BUBBLEUPMAX(PARENT(i), H)
16:    else
17:       BUBBLEUPMIN(i, H)
18:    end if
19: end if
```

Here is the supporting BubbleUp code; the value moves up by two levels in the heap with each exchange.

Algorithm 2 BubbleUpMin(i, H)

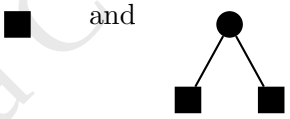
```
1: if Parent(Parent(i)) > 0 and H[Parent(Parent(i))] > H[i] then
2:     exchange H[i] with H[Parent(Parent(i))]
3:     BubbleUpMin(Parent(Parent(i)), H)
4: end if
```

Algorithm 3 BubbleUpMax(i, H)

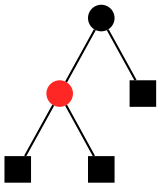
```
1: if Parent(Parent(i)) > 0 and H[Parent(Parent(i))] < H[i] then
2:     exchange H[i] with H[Parent(Parent(i))]
3:     BubbleUpMax(Parent(Parent(i)), H)
4: end if
```

5. (a) In Lemma 1 of Lecture 6 (September 15) we proved that the $(n+1)$-leaf binary tree with the least possible external path length has all of its leaves on levels l and $l+1$ for some value of l. So we simply color everything black *except* the internal nodes at level l; those we color red. There is clearly no parent/child pair of red nodes because all red nodes are at level l. All leaves are at black depth l, so the result is a proper red-black tree.

   (b) If there are $n+1$ leaves, there are n internal nodes. We proved in section 1.2.2 of Lecture 7 (September 17) that the greatest possible EPL happens when the tree is (essentially) a linear list, and for n internal nodes the EPL is $n(n+3)/2$ because there is one leaf at depth 1 from the root, 1 at depth 2, ..., 1 at depth $n-1$, and 2 at n. The statement is clearly true for $n = 0$ and $n = 1$, because the trees are just



respectively, which are red-black trees, and for $n = 2$, because we can color the one non-leaf child of the root red:



However, it is false for $n > 2$: the shallowest leaf is at depth 1 from the root and the deepest leaves (a pair) is at level n. For $n > 2$ consider the deepest leaves at depth n. Because there is a leaf at depth 1, to get uniform black depth we would need all $n - 1 \geq 2$ non-root nodes above those two deepest leaves to be red, implying at least one parent-child pair of red nodes, an unacceptable situation.