CS240E W24	**Tutorial 3**	Jan. 29

Overview

- average-case vs expected runtime
  - BST height
  - probabilistic analysis (hiring problem)
  - Morris' counter (Monte Carlo algorithm)
  - string comparison
- amortized analysis
- partially sorted array (sorting lower bound review)
- searching lower bound
- more practice with: indicator random variables, recurrence relations, binary search

---

Problems

Q1. Average-case vs expected (hiring problem). Suppose we must hire a new employee. There are n candidates sequentially, one each day.

It takes I time to interview a candidate, and it takes H units to hire them.

We want to have at all times the best possible person for the job. After interviewing each applicant, if they are better than our current employee, we hire them immediately (and fire our current employee).

We can compare two candidates in constant time.

```
hire(cand[1..n]):
    curr = dummy candidate // compares worse than anyone
    for i = 1..n:
        interview cand[i]
        if cand[i] is better than curr:
            hire cand[i]
            curr = cand[i]
```

Suppose m candidates are hired. Then the worst-case runtime is in $\Theta(In + Hm)$.

We can rank each candidate with a unique number between 1 and n, and use $rank[i]$ to denote the rank of candidate i. We adopt the convention that a higher ranked applicant corresponds to a better qualified applicant. Note that the ordered list

$$\langle rank[1], \ldots, rank[n] \rangle$$

is a permutation of the list $\langle 1, \ldots, n \rangle$.

(a) Describe an instance that achieves the runtime $\Omega(Hn)$.

(b) Show that in the average-case we hire a new candidate $O(\log n)$ times.

Q2. Amortized analysis. We are given a binary search tree on n nodes, storing n distinct keys. We can list all keys in increasing order using in-order traversal in time linear in n.

The operation *successor*(x) returns the in-order *successor* of x in the tree, which is the node z such that $z.key > x.key$ and no ther keys are stored in between (or *null* if such z does not exist), in Θ(height of the tree) time.

Consider the following algorithm to print all keys in the tree T in increasing order:

```
x = T.get_min(); // left-most node, Theta(height)
do {
    print(x.key)
    x = T.successor(x)
} while (x is not null);
```

(a) Give an asymptotic bound on the worst-case runtime of this algorithm in terms of the height h of T. When is the bound minimized?

(b) Show that the amortized runtime of *successor* is $O(1)$. Conclude that the runtime of the algorithm is in $\Theta(n)$.

Q3. Morris' probabilistic counting. With a deterministic b-bit counter, we can only count up to $2^b - 1$. With *probabilistic counting* we can count to larger values at the expence of loss of precision.

Let a counter reading of i represent a count of v_i, for $0 \leq i \leq 2^b - 1$. Initially the counter reads 0, indicating the count of $v_0 = 0$.

The operation *increment* works on a counter with reading i in a probabilistic manner:

if $i < 2^b - 1$, increase counter reading with probability $1/(v_{i+1} - v_i)$, and leave the counter unchanged otherwise;

if $i = 2^b - 1$, report overflow.

Note that if we take $v_i = i$, then the counter is an ordinary deterministic counter. More interesing situations arise if $v_i = 100i$, if $v_i = 2^i$, or if $v_i = i$-th Fibonacci number.

Assuming the probability of an overflow is negligible, show that the value represented by the counter after n *increment* opreations is n.

Q4. String comparison. Let A, B be two binary strings of length n. A string comparison of A with B determines whether A is smaller, larger, or the same as B by the first index where they differ (if it exists):

```
str_cmp(A, B, n):
    for(i = 0; i < n; ++i):
        if A[i] < B[i]: return "A is smaller"
        if A[i] > B[i]: return "A is bigger"
    return "they are equal"
```

Show that the average-case runtime of *str-cmp* is $O(1)$. You may use that $\sum_{i \geq 0} i/2^i \in O(1)$ without proof.

Q5. Partially sorted array. Let $0 < \epsilon < 1$. Suppose we have an array A of n items such that the first $n - n^\epsilon$ items are sorted. Give an $O(n)$ time algorithm to sort A.

Q6. Searching lower bound. Show that any comparison-based searching algorithm uses $\Omega(\log n)$ comparisons:

(a) in the worst case; and

(b) in the average case.

Hint: this is true even if the input is sorted.

Additional problems

Q7. Indicator random variables. Let $A[1..n]$ be an array of n distinct integers. We say that a pair (i, j) is an *inversion* of A if $i < j$ but $A[i] > A[j]$. Suppose that the elements of A form a uniform random permutation of $\langle 1, \ldots, n \rangle$. Find the expected number of inversions of A.

Q8. Recursion tree. Use a recursion tree to give an asymptotically tight solution to the recurrence

$$T(n) = T(n - a) + T(a) + cn$$

for constants $a \geq 1, c > 0$.

Q9. Perfect square.

(a) Give an algorithm to test whether a given $n \geq 1$ is a perfect square in $O(\log n)$ time.

(b) If n is a perfect square, compute $\sqrt{n}$ in $O(\log n)$ time.

Q10. Shifted array. We are given an array a of n numbers that was initially in sorted order, and then was shifted by some unknown amount. For example,

$$[1, 2, 3, 4, 5, 6] \mapsto [3, 4, 5, 6, 1, 2].$$

Find the minimum element of a in $O(\log a)$ time.