CS240E S26	Tutorial 3	May 29

- Sorting
- Average-case Runtime
- Expected Runtime
- Indicator random variables
- Probabilistic Counting
- Decision Trees

Q1. Almost 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.

Q2. Collinear points. We are given n distinct ordered pairs of integers $(x_1, y_1), \ldots, (x_n, y_n)$ in general position.[1]

We say that a set of points S is *collinear* if there exist m, b such that for all $(x_i, y_i) \in S$ we have $y_1 = mx_i + b$.

We would like to find a maximum cardinality subset of the given points such that all of them are collinear. You may assume that given two points, we can compute the corresponding m and b for the line passing through them in constant time.

(a) Give an algorithm to find a maximum cardinality set of collinear points in $O(n^2 \log n)$ time.

(b) It is unknown whether there is an algorithm with time better than $O(n^2)$; however, if we assume that a maximum cardinality subset of collinear points has exactly n/k points, for some fixed constant k, then we can find such a subset in $O(n)$ expected time. Give a randomized algorithm to report the points in such a subset in expected $O(n)$ time.

Note: you may treat k as a constant in asymptotic notation.

Q3. Bogosort.

Give the best-case, worst-case, and expected running time for the infamous Bogosort algorithm in terms of the size n of the array. You can assume that the shuffle function takes $O(n)$ time and produces each permutation equally likely. You can also assume that the array contains no duplicates.

```
bogosort(A):
        shuffle(A);
        if (A is sorted) {
                return A;
```

[1] Recall that *general position* means that no two x-coordinates are the same and no two y-coordinates are the same.

```
} else {
        bogosort(A);
}
```

Q4. String compare. 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 $O(1)$.

Q5. 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 are a uniform random permutation of $\langle 1, \ldots, n \rangle$. Find the expected number of inversions of A.

Q6. The 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 that have already been interviewed 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 $\langle \text{rank}[1], \ldots, \text{rank}[n] \rangle$ 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.

Q7. 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.

Q8. 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.