Practice Problems for Quiz 1

- The following is a compilation of relevant problems from previous semesters.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to *give an algorithm*, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- **Good luck!**

Problem-1: True or False

Please circle **T** or **F** for the following. *No justification is needed (nor will be considered).*

(a) **T** **F**

Suppose that you modify the median algorithm so that instead of splitting into groups of 5 and using the median of the medians of these groups as a pivoting element, you split into groups of size 11. Then the running time is still $O(n)$.

(b) **T** **F**

The running time of a divide-and-conquer algorithm on inputs of size n, $T(n)$, satisfies the recurrence $T(n) = 2^n \cdot T(n/2)^2$ with boundary condition $T(1) = 2$. This recurrence satisfies $T(n) = 2^{\Omega(n)}$.

(c) **T** **F**

In a disjoint-set data structure where we do union by size and use path compression, when we merge disjoint sets A and B, all of the elements of the smaller of the two sets will end up with a direct pointer to the representative of the new joined set.

(d) **T** **F**

Weighted Interval Scheduling* can be solved with a greedy algorithm that: (i) sorts the intervals with respect to weight; and then (ii) considers the intervals from highest to lowest weight, adding an interval to the schedule as long as it does not overlap with previously selected intervals.

**Weighted Interval Scheduling was discussed in Lecture 1: as input we are given n 3-tuples (t_i^s, t_i^e, w_i) where t_i^s is the leftmost point of interval i, t_i^e is the rightmost point of interval i, and w_i is the weight of the interval. The goal is to select a subset of non-overlapping intervals whose sum of weights is as large as possible.*

(e) **T** **F**

We consider an implementation of the ACCESS operation for self-organizing lists which, whenever it accesses an element x of the list, transposes x with its predecessor in the list (if it exists). This heuristic is 2-competitive.

(f) **T** **F**

Suppose algorithm $\mathcal{A}$ has two steps, and $\mathcal{A}$ succeeds if both the steps succeed. If the two steps succeed with probability p_1 and p_2 respectively, then $\mathcal{A}$ succeeds with probability $p_1 p_2$.

(g) **T** **F**

We are given an array $A[1 \ldots n]$. We can delete its $n/2$ largest elements in time $O(n)$.

(h) **T** **F**

We define n random variables $X_i = \sum_{j=1}^{n} c_{(i,j)}$, where for every (i, j), $c_{(i,j)}$ is determined by an independent unbiased coin flip, so $c_{(i,j)}$ is one with probability 1/2 and zero

otherwise. Then, for every $\beta \in (0,1)$

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i > (1+\beta)\frac{n^2}{2}\right] < e^{-\beta^2 n^2/6}$$

(i) **T** **F**

Consider a k-bit vector $v = x_1 \dots x_k$ chosen uniformly at random from $\{0,1\}^k$. For any $S \subseteq \{1,\dots,k\}$ such that $S \neq \emptyset$, define $m_S(v) = \sum_{i \in S} x_i \pmod 2$. Then,

$$\mathrm{Var}\left[\sum_{S \subseteq \{1,\dots,k\}, S \neq \emptyset} m_S(v)\right] = \frac{2^k - 1}{4}$$

(j) **T** **F**

If an algorithm runs in time $\Theta(n)$ with probability 0.9999 and in time $\Theta(n^2)$ with the remaining probability, then its expected run-time is $\Theta(n)$.

Problem-2: Democratic Feudal System

A kingdom has a single monarch. The monarch has three loyal dukes who reign over three parts of his kingdom. Each of these dukes in turn has three loyal advisors, who each have three loyal subjects, and so on, down to the peasants. This feudal society can be modeled by a tree with branching factor 3 and height h. Voting day has now arrived, and each peasant votes either for or against some public policy. This is modeled by assigning a value of 0 or 1 to each of the leaves of the tree. Any intermediate officer votes according to the majority of his immediate constituents. That is, the value assigned to any non-leaf node of the tree is the majority of the values of its immediate children, whose own value is the majority of the values of their own immediate children, and so on so forth down to the leaves of the tree. If the value thus assigned to the root of the tree (the king's node) is 1, the policy passes; otherwise, the policy fails. Since collecting votes is expensive, the king would like to determine an efficient algorithm to evaluate the final vote.

(a) Show that in the worst case, a deterministic algorithm will have to query the vote of all of the $n = 3^h$ leaves in order to determine the value assigned to the root. That is, argue that it is impossible for a deterministic algorithm to correctly determine the value at the root of the tree without knowing every single leaf's value.

(b) Design a randomized algorithm for computing the majority of three bits b_1, b_2, b_3, such that, no matter what the values of b_1, b_2, b_3 are, with probability at least $\frac{1}{3}$, the algorithm does not inspect one of the values.

(c) Design a randomized algorithm that evaluates the final vote correctly with probability 1, but only inspects the value of n^α leaves in expectation, for some constant $\alpha < 1$ (independent of n).

(d) Show that there exists some constant $c \geq 1$ (independent of n) such that, with probability at least 0.99, your algorithm from (c) inspects at most cn^α leaves, where α is the constant you derived in (c).

Problem-3: Fast Fourier Transform (FFT)

Ben Bitdiddle is trying to multiply two polynomials using the FFT. In his trivial example, Ben sets $a = (0, 1)$ and $b = (0, 1)$, both representing $0 + x$, and calculates:

$$A = \mathcal{F}(a) = B = \mathcal{F}(b) = (1, -1),$$

$$C = A * B = (1, 1),$$

$$c = \mathcal{F}^{-1}(C) = (1, 0).$$

So c represents $1 + 0 \cdot x$, which is clearly wrong. Point out Ben's mistake in one sentence; no calculation needed. (Ben swears he has calculated FFT $\mathcal{F}$ and inverse FFT $\mathcal{F}^{-1}$ correctly.)

Problem-4: Amortized Analysis

Design a data structure to maintain a set S of n distinct integers that supports the following two operations:

(a) INSERT(x, S): insert integer x into S.

(b) REMOVE-BOTTOM-HALF(S): remove the smallest $\lceil \frac{n}{2} \rceil$ integers from S.

Describe your algorithm and give the worse-case time complexity of the two operations. Then carry out an amortized analysis to make INSERT(x, S) run in amortized $O(1)$ time, and REMOVE-BOTTOM-HALF(S) run in amortized 0 time.

Problem-5: Verifying Polynomial Multiplication

This problem will explore how to check the product of two polynomials. Specifically, we are given three polynomials:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,$$

$$q(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_0,$$

$$r(x) = c_{2n} x^{2n} + c_{2n-1} x^{2n-1} + \cdots + c_0.$$

We want to check whether $p(x) \cdot q(x) = r(x)$ (for all values x). Via FFT, we could simply compute $p(x) \cdot q(x)$ and check in $O(n \log n)$ time. Instead, we aim to achieve $O(n)$ time via randomization.

(a) Describe an $O(n)$-time randomized algorithm for testing whether $p(x) \cdot q(x) = r(x)$ that satisfies the following properties:

   i. If the two sides are equal, the algorithm outputs YES.

   ii. If the two sides are unequal, the algorithm outputs NO with probability at least $\frac{1}{2}$.

(b) Prove that your algorithm satisfies Property **Problem-5:**(a)i.

(c) Prove that your algorithm satisfies Property **Problem-5:**(a)ii.

Hint: Recall the Fundamental Theorem of Algebra: A degree-d polynomial has (at most) d roots.

(d) Design a randomized algorithm to check whether $p(x) \cdot q(x) = r(x)$ that is correct with probability at least $1 - \epsilon$. Analyze your algorithm in terms of n and $1/\epsilon$.

Problem-6: Median of two sorted arrays

Finding the median of a sorted array is easy: return the middle element. But what if you are given two sorted arrays A and B, of size m and n respectively, and you want to find the median of all the numbers in A and B? You may assume that A and B are disjoint.

(a) Give a naïve algorithm running in $\Theta(m+n)$ time.

(b) If $m = n$, give an algorithm that runs in $\Theta(\lg n)$ time.

(c) Give an algorithm that runs in $O(\lg(\min\{m, n\}))$ time, for any m and n.

Problem-7: Forgetful Forrest

Prof. Forrest Gump is very forgetful, so he uses automatic calendar reminders for his appointments. For each reminder he receives for an event, he has a 50% chance of actually remembering the event (decided by an independent coin flip).

(a) Suppose we send Forrest k reminders for each of n events. What is the expected number of appointments Forrest will remember? Give your answer in terms of k and n.

(b) Suppose we send Forrest k reminders for a *single* event. How should we set k with respect to n so that Forrest will remember the event with high probability, i.e., $1-1/n^{\alpha}$?

(c) Suppose we send Forrest k reminders for each of n events. How should we set k with respect to n so that Forrest will remember *all* the events with high probability, i.e., $1-1/n^{\alpha}$?

Problem-8: k**-Sum Subset**

Suppose we have an array A with n elements, each of which is an integer in the range of $[0, \ldots, 50n]$. Give an $O(n \log n)$ algorithm that when given an integer $0 \leq t \leq 5 \cdot 50n$, determines whether there exist at most 5 (not necessarily distinct) elements of A which sum to t. That is, output "YES" if there exist $i_1, \ldots, i_k$ with $k \leq 5$ such that $A[i_1] + \cdots + A[i_k] = t$ and output "NO" otherwise.

Problem-9: Ordered Stack

An ordered stack is a data structure that stores a sequence of items and supports the following operations.

- ORDEREDPUSH(x) removes all items smaller than x from the beginning of the sequence and then adds x to the beginning of the sequence.
- POP deletes and returns the first item in the sequence (or returns Null if the sequence is empty).

(a) Show how to implement an ordered stack with a simple linked list. Prove that if we start with an empty data structure and perform a sequence of n arbitrary ORDEREDPUSH(x) and POP operations, then the amortized cost of each such operation is $O(1)$.

(b) Suppose we are given an array $A[1 \ldots n]$ that stores the height of n buildings on a city street, indexed from west to east. The view of building i is defined as the number of buildings between building i and the first building west of i that is taller than i. For example if $A = [1, 5, 2, 3, 1, 2, 1, 8]$, then the view of building 4 (which has height 3) is 1. You want to compute the view of the n buildings of A. Show how you can modify the ordered stack in order to do that in $O(n)$ time.

Problem-10: Find the Pairs

Consider a set U of $2n$ distinct balls numbered from $1 \ldots 2n$ distributed evenly across n containers (each container has exactly 2 balls). You are not allowed to look inside the containers. But, we will answer your queries of the following form:

"*What is the smallest number of containers that contain all the balls in* S?" where $S \subseteq U$ is any subset of the $2n$ balls.

For example if $n = 4$ and the balls were arranged in the following way

$$\{1, 4\}, \{2, 8\}, \{3, 6\}, \{5, 7\}$$

in 4 containers, an example query would be "*What is the smallest number of containers that contain the balls* $\{1, 4, 2, 6\}$?" to which we would reply 3.

Two balls are said to be paired if they lie in the same container. Your task is to figure out for each ball the other ball it is paired with. Give an algorithm for figuring out all the pairings using $O(n \log n)$ queries. Note that we only care about the total number of queries your algorithm uses and not the running time.

Problem-11: Bin Packing

Let $X_i, 1 \leq i \leq n$ be independent and identically distributed random variables following the distribution

$$\mathbb{P}\left(X_i = \frac{1}{2^k}\right) = \frac{1}{2^k} \quad \textit{for} \quad 1 \leq k \leq \infty$$

Each X_i represents the size of the item i. Our task is to pack the n items in the least possible number of bins of size 1. Let Z be the random variable that represents the total number of bins that we have to use if we pack the n items optimally. Consider the following algorithm for performing the packing:

BIN-PACKING$(x_1, ..., x_n)$:

Sort $(x_1, ..., x_n)$ by size in decreasing order. Let $(z_1, ..., z_n)$ be the sorted array of items
Create a bin b_1
For each $i \in \{1, ..., n\}$
  If z_i fits in some of the bins created so far
    Choose one of them arbitrarily and add z_i to it
  Else create a new bin

return the number of created bins.

(a) Prove that the above packing algorithm is optimal for this problem.

(b) Prove that

$$\mathbb{E}[Z] \leq \frac{n}{3} + 1$$

(c) Prove that

$$\mathbb{P}\left(Z \geq \frac{n}{3} + \sqrt{n}\right) \leq \frac{1}{e}$$

Problem-12: Sampling without Erasures

Ben Bitdiddle wants to implement the Reservoir Sampling algorithm from the class. Recall that this algorithm samples a uniformly random element from the stream $x_1, \ldots, x_n$ while not knowing n ahead of time and storing only a single element of the stream at any given moment. To achieve this, one starts with $x = x_1$ and then, in each round i, one replaces the stored element x with the element x_i with probability $p_i = \frac{1}{i}$, independently at random.

Ben realized, however, that the memory on his computer is non-erasable and as a result, instead of replacing the stored element, he has to append that element to the list of already stored elements, for as many elements as he picks from the stream.

(a) Show that with probability at least $\frac{3}{4}$, Ben will end up storing $O(\log n)$ elements.

Note: Recall that $\sum_{j=1}^{k} 1/j = \Theta(\ln k)$.

(b) Assume now a different scenario. Ben knows the length of the stream n ahead of time, but he wants to avoid having to store any unnecessary elements. Specifically, he wants to have an algorithm that takes a pass through the stream and for each element x_i for $i \in \{1, \ldots, n\}$ selects this element with some probability $\hat{p}_i$ as the uniformly randomly chosen element of the stream. From that point on, he ignores the rest of the stream.

What should the probabilities $\hat{p}_i$ be to ensure that this algorithm is correct? Justify your answer.