Practice Quiz 1

- The following practice quiz is a compilation of relevant problems from previous semesters.
- This practice quiz may not be taken as a strict gauge for the difficulty level of the actual quiz.
- The quiz contains multiple problems, several with multiple parts. You have 120 minutes to complete this quiz.
- Write your solutions in the space provided. If you run out of space, continue on a scratch page and make a notation.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to *give an algorithm* in this quiz, 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.
- Do not spend too much time on any one problem.
- **Good Luck!**

Problem-1: T/F Questions

Answer *true* or *false* to each of the following. **No justification is required for your answer**. Space is provided for scratch work.

(a) The solution to the recurrence $T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{7n}{9}\right) + O(n)$ is $O(n)$. You may assume $T(n) = 1$ for n smaller than some constant c.

(b) Given a set of points $\{x_1, ..., x_n\}$, and a degree n bounded polynomial $P(x) = a_0 + a_1 x + ... + a_{n-1} x_{n-1}$, we can evaluate $P(x)$ at $\{x_1, ..., x_n\}$ in $O(n \log n)$ time using the FFT algorithm presented in class.

(c) In a weighted connected graph $G = (V, E)$, each edge with the minimum weight belongs to some minimum spanning tree of G.

(d) For a flow network with an *integer* capacity on every edge, the *Ford-Fulkerson algorithm* runs in time $O((|V| + |E|)|f|)$, where $|f|$ is the *maximum* flow.

(e) If the primal Linear Program is feasible and has a bounded optimum, then the dual Linear Program is feasible.

(f) In a weighted, connected graph G, if s is a starting node in *Prim's algorithm*, then for any other vertex v, the path on the resulting minimum spanning tree from s to v is the shortest path.

(g) Let $G = (V, E)$ and let H be the subgraph of G induced by some set of vertices $V' \subset V$ – i.e., $H = (V', E')$ where E' consists of all edges both of whose endpoints are in V'. Then every minimum spanning tree of H is a subgraph of some minimum spanning tree of G.

(h) When performing competetive analysis on an *online* algorithm, we must know an *optimal, offline* algorithm relative to which we are assessing our *online* algorithm.

(i) 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-competetive.

Problem-2: Well Spaced Triples

Suppose we are given a bit string $B[1 \ldots n]$. A triple of distinct indices $1 \le i < j < k \le n$ is called a *well-spaced triple* in B if $B[i] = B[j] = B[k] = 1$ and $k - j = j - i$.

(a) Describe an algorithm to determine whether B has a well-spaced triple in $O(n^2)$ time.

(b) Describe an algorithm to determine the number of well-spaced triples in B in $O(n \log n)$ time. (**Hint:** Use FFT)

Problem-3: Minimum Spanning Trees

(a) In the graph depicted below, prove that:

i. The edge between B and E is not in *any* minimum spanning tree.

ii. In the graph shown in Figure 1, prove that the edge between E and G is in *every* minimum spanning tree.

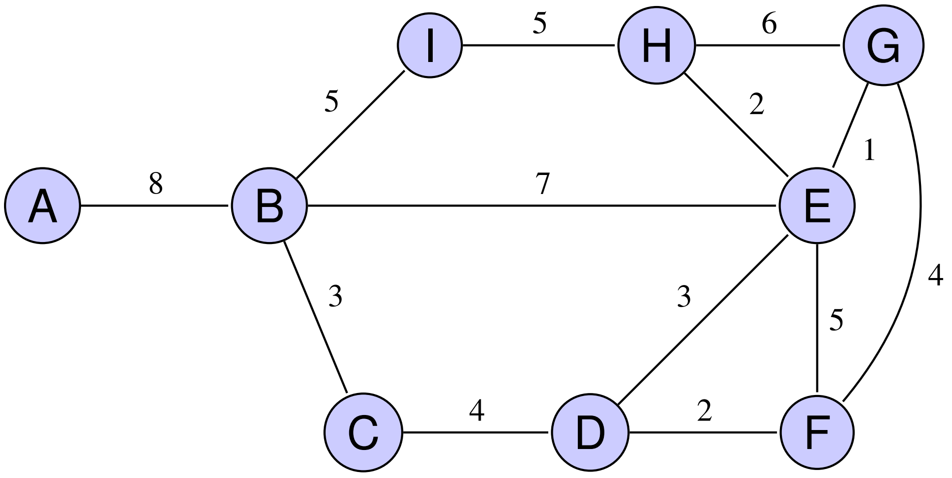

Figure 1: Example graph for part (a).

(b) Suppose that we are given a minimum spanning tree T of a graph $G = (V, E)$ that is connected, undirected, and has distinct positive integer weights (i.e. all edge weights are different positive integers). We want to design an algorithm that updates the minimum spanning tree when a new edge $e = (u, v)$ is added to E, for some $u, v \in V$.

Alyssa P. Hacker suggests the following algorithm: to compute the minimum spanning tree T' of graph $G' = (V, E \cup \{e\})$ from T, we **(1)** add edge e to T, and then **(2)** delete the heaviest edge in the cycle that is created in T with the addition of edge e.

Prove that this algorithm constructs a minimum spanning tree for G'. (i.e. give a proof of correctness.)

(c) After observing Alyssa's algorithm, Ben Bitdiddle wants to design an algorithm for updating the minimum spanning tree of a graph when new vertices are inserted. Specifically, Ben wants an algorithm that takes as input:

- A minimum spanning tree T of some undirected graph $G = (V, E)$,
- A new vertex, v'.
- A set of new (weighted) edges, E', between v' and V,

and computes a minimum spanning tree T' of the graph $G' = (V \cup \{v'\}, E \cup E')$.

Design an $O(V \log V)$ time algorithm for Ben's problem. As before, assume that all edge weights are distinct integers.

Problem-4: 6.046 Carpool

The n people in your dorm want to carpool to 34-101 during the m days of 6.046. On day i, some subset S_i of people actually want to carpool (i.e., attend lecture), and the driver d_i must be selected from S_i. Each person j has a limited number of days ℓ_j they are willing to drive.

Give an algorithm to find a driver assignment $d_i \in S_i$ for each day i such that no person j has to drive more than their limit ℓ_j. (The algorithm should output "no" if there is no such assignment.)

For example, for the following input with $n = 3$ and $m = 3$, the algorithm could assign Penny to Day 1 and Day 2, and Leonard to Day 3.

Person	Day 1	Day 2	Day 3	Driving limit
1 (Penny)	X	X	X	2
2 (Leonard)	X		X	1
3 (Sheldon)		X	X	0

Hint: Use network-flow.

Problem-5: Candy Land.

The Candy Corporation of Candy Land owns 3 candy factories f_1, f_2, and f_3 that manufacture 20 kinds of candy $c_1, c_2, .., c_{20}$. Each factory f_i manufactures m_{ij} pounds of each candy c_j. The Candy Corporation has 103 local stores $s_1, s_2, .., s_{103}$ that sell candy. Each store s_k requires d_{kj} pounds of candy c_j. The factories only produce what is needed, and hence we have

$$\sum_{i=1}^{3}\sum_{j=1}^{20} m_{ij} = \sum_{k=1}^{103}\sum_{j=1}^{20} d_{kj}$$

A directed graph $G = (V, E)$ represents the road map of Candy Land, where V represents road intersections and E represents the roads themselves. Factories and stores are located at intersections: $f_i \in V$ for $i = 1, 2, 3$ and $s_k \in V$ for $k = 1, 2, .., 103$. King Kandys tax collector, Lord Licorice, has designated a tax rate $c(e)$ for each road $e \in E$: if x pounds of candy (of any kind) are shipped along road e, Lord Licorice must be paid $x \cdot c(e)$ gumdrops.

Princess Lolly, the supply manager of the Candy Corporation, wants to determine how to deliver the candy from the factories to the local stores while minimizing the tax paid to Lord Licorice. Give a linear-programming formulation of this problem.

Problem-6: Majority Element Suppose we have a list of n objects, where $n = 2^k$ for some positive integer k. The only operation that we can use on these objects is the following: $isEqual(i, j)$, which runs in constant time and returns True if and only if two objects i and j are equal. *The objects are neither hashable nor comparable.*

Our goal is to find if there exists a majority (more than $n/2$ occurrences) of the same element and if so, return it.

Design an $O(n)$ time algorithm that finds whether a majority element exists, and if so, returns it. Provide arguments for your algorithm's correctness and runtime.

Hint: Consider implementing and using a subroutine $Reduce(L)$ which converts the list L into an $n/2$ sized list of each adjacent pairs.

For example, consider the list:

$$L = [\ddagger, \oslash, \star, \ddagger, \sqcap, \sqcap, \ddagger, \ddagger, \ddagger, \oslash, \ddagger, \bigcirc, \ddagger, \uplus, \ddagger, \ddagger]$$

A single iteration of $Reduce(L)$ would create a list L' as follows:

$$L' = [(\ddagger, \oslash), (\star, \ddagger), (\sqcap, \sqcap), (\ddagger, \ddagger), (\ddagger, \oslash), (\ddagger, \bigcirc), (\ddagger, \uplus), (\ddagger, \ddagger)]$$

Problem-7: 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.