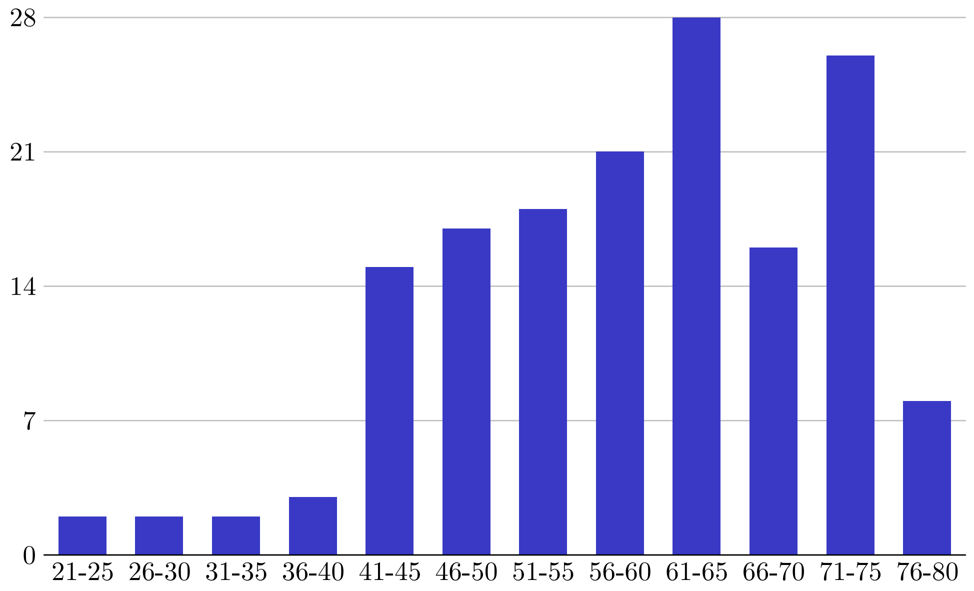
Quiz 1 Solutions



Mean: 58; Median: 60; Standard deviation: 12

Problem 0. Name. [1 point] Write your name on every page of this exam booklet! Don't forget the cover.

Problem 1. True or False. [24 points] (8 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, and briefly explain why. Your justification is worth more points than your true-or-false designation.

(a) T F The solution to the recurrence $T(n) = 2^n T(n-1)$ is $T(n) = \Theta((\sqrt{2})^{n^2+n})$. (Assume $T(n) = 1$ for n smaller than some constant c).

Solution: [3 points] True. Let $T(0) = 1$.
Then $T(n) = 2^n \cdot 2^{n-1} \cdot 2^{n-2} \ldots 2^1 = 2^{(n+(n-1)+(n-2)+\ldots+1)}$.
Because $\sum_{i=1}^{n} i = n(n+1)/2$, we therefore have $T(n) = 2^{(n^2+n)/2} = (\sqrt{2})^{n^2+n}$.
Some students also correctly solved the problem by using the substitution method.
Some students made the mistake of multiplying the exponents instead of adding them. Also, it has to be noted that $2^{n^2/2} \neq \Theta(2^{n^2})$.

(b) T F The solution to the recurrence $T(n) = T(n/6) + T(7n/9) + O(n)$ is $O(n)$. (Assume $T(n) = 1$ for n smaller than some constant c).

Solution: [3 points] True. Using the substitution method:

$$\begin{aligned} T(n) &\leq cn/6 + 7cn/9 + an \\ &\leq 17cn/18 + an \\ &\leq cn - (cn/18 - an) \end{aligned}$$

This holds if $c/18 - a \geq 0$, so it holds for any constant c such that $c \geq 18a$.
Full credit was also given for solutions that uses a recursion tree, noting that the total work at level i is $(17/18)^i n$, which onverges to $O(n)$.

(c) T F In a simple, undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph is in no minimum spanning tree.

Solution: [3 points] False. If the heaviest edge in the graph is the only edge connecting some vertex to the rest of the graph, then it must be in every minimum spanning tree.

(d) T F The weighted task scheduling problem with weights in the set $\{1, 2\}$ can be solved optimally by the same greedy algorithm used for the unweighted case.

Solution: [3 points] False. The algorithm will fail given the set of tasks (given in the form $((s_i, f_i), w_i)$:
$\{((0, 1), 1), ((0, 2), 2)\}$.

(e) T F Two polynomials p, q of degree at most $n - 1$ are given by their coefficients, and a number x is given. Then one can compute the multiplication $p(x) \cdot q(x)$ in time $O(\log n)$.

Solution: [3 points] False. We need at least $\Theta(n)$ time to evaluate each polynomial on x and to multiply the results. Some students argued incorrectly that it must take $O(n \log n)$ using FFT, but FFT overkills because it computes all coefficients, not just one.

(f) T F Suppose we are given an array A of n distinct elements, and we want to find $n/2$ elements in the array whose median is also the median of A. Any algorithm that does this must take $\Omega(n \log n)$ time.

Solution: [3 points] False. It's possible to do this in linear time using SELECT: first find the median of A in $\Theta(n)$ time, and then partition A around its median. Then we can take $n/4$ elements from either side to get a total of $n/2$ elements in A whose median is also the median of A.

(g) T F There is a density $0 < \rho < 1$ such that the asymptotic running time of the Floyd-Warshall algorithm on graphs $G = (V, E)$ where $|E| = \rho |V|^2$ is better than that of Johnson's algorithm.

Solution: [3 points] False. The asymptotic running time of Floyd-Warshall is $O(V^3)$, which is at best the same asymptotic running time of Johnson's (which runs in $O(VE + V \log V)$ time), since $E = O(V^2)$

(h) T F Consider the all pairs shortest paths problem where there are also weights on the vertices, and the weight of a path is the sum of the weights on the edges and vertices on the path. Then, the following algorithm finds the weights of the shortest paths between all pairs in the graph:

```
APSP-WITH-WEIGHTED-VERTICES$(G, w)$:
1  for $(u, v) \in E$
2       Set $w'(u, v) = (w(u) + w(v))/2 + w(u, v)$
3  Run Johnson's algorithm on $G, w'$ to compute the distances $\delta'(u, v)$ for all $u, v \in V$.
4  for $u, v \in V$
5       Set $d_{uv} = \delta'(u, v) + \frac{1}{2}(w(u) + w(v))$
```

Solution: [3 points] True. Any shortest path from u to v in the original graph is still a shortest path in the new graph. For some path $\{v_0, v_1, \ldots, v_k\}$, we have:

$$\begin{aligned} w'(v_0 \leadsto v_k) &= \sum_{i=0}^{k-1} ((w(v_i) + w(v_{i+1}))/2 + w(v_i, v_{i+1})) \\ &= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) + \frac{1}{2}\left(\sum_{i=0}^{k-1} w(v_i) + \sum_{i=1}^{k} w(v_i)\right) \\ &= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) + \sum_{i=0}^{k} w(v_i) - \frac{1}{2}(w(v_0) + w(v_k)) \\ &= w(v_0 \leadsto v_k) - \frac{1}{2}(w(v_0) + w(v_k)) \end{aligned}$$

Therefore, the order of all paths from v_0 to v_k remains unchanged so Johnson's algorithm in line 3. finds the correct path, and the adjustment in line 5 finds the correct length $d_{v_0 v_k}$.

Problem 2. Translation [25 points] (5 parts)

You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages: n languages in total. Your boss wants the entire documentation to be available in all n languages.

There are m different translators for hire. Some of those translators are volunteers that do not get any money for their services. Each translator knows *exactly* two different languages and can translate back and forth between them. Each translator has a non-negative hiring cost (some may work for free). Unfortunately, your budget is too small to hire one translator for each pair of languages. Instead, you must rely on chains of translators: an English-Spanish translator and a Spanish-French translator, working together, can translate between English and French. Your goal is to find a minimum-cost set of translators that will let you translate between every pair of languages.

We may formulate this problem as a connected undirected graph $G = (V, E)$ with non-negative (i.e., zero or positive) edge weights w. The vertices V are the languages for which you wish to generate translations. The edges E are the translators. The edge weight $w(e)$ for a translator e gives the cost for hiring the translator $w(e)$. A subset $S \subseteq E$ of translators can be used to translate between $a, b \in V$ if and only if the subgraph $G_S = (V, S)$ contains a path between a and b. The set $S \subseteq E$ is a translation network if and only if S can be used to translate between all pairs $a, b \in V$.

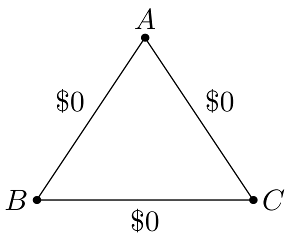
(a) Prove that each minimum spanning tree of G is also a minimum-cost translation network.

Solution: [5 points] Let T be some minimum spanning tree of G. Because T is a spanning tree, there is a path in T between any pair of vertices. Hence, T is a translation network.

For the sake of contradiction, suppose that T is not a minimum-cost translation network. Then there must be some translation network S with total cost strictly smaller than T. Because S connects every pair of vertices, it must have some spanning tree T^* as a subgraph. Because all edge weights are nonnegative, we have $w(T^*) \leq w(S) < w(T)$. Hence, T is not the minimum spanning tree. This contradiction means that all spanning trees are also minimum-cost translation networks.

(b) Give an example of a minimum-cost translation network that is not a minimum spanning tree of G.

Solution: [5 points] If the graph of translators contains a cycle of translators all willing to work for \$0, then it is possible to hire all of the translators in the cycle without increasing the overall cost of the translation network. The smallest example of this is the following:



All three MSTs of the graph have total cost \$0, so any translation network of cost \$0 has minimum cost. Hence, we can take all translators in the cycle to get a minimum-cost translation network that is not an MST.

(c) Give an efficient algorithm that takes G as input, and outputs a minimum-cost translation network of G. State the runtime of your algorithm in terms of the number of languages n and the number of potential translators m.

Solution: [5 points] We saw in part **(a)** that every minimum spanning tree is a minimum-cost translation network of G. Hence, to find a minimum-cost translation network, it is sufficient to find a minimum spanning tree of G. We may do so using Kruskal's algorithm, for a runtime of $\Theta(m \log n)$, or using Prim's algorithm, for a runtime of $\Theta(m + n \log n)$.

Your bosses have decided that the previous approach to translation doesn't work. When attempting to translate between Spanish and Portuguese — two relatively similar languages — it degrades the translation quality to translate from Spanish to Tagalog to Mandarin to Portuguese. There are certain clusters of languages that are more closely related than others. When translating between two languages that lie within the same cluster, such as Spanish and Portuguese, the translation is of high quality when the sequence of languages used to translate between them is completely contained within the cluster.

More formally, the language set V can be divided into disjoint clusters $C_1, \ldots, C_k$. Each cluster C_i contains languages that are fairly similar; each language is contained in exactly one cluster. Your bosses have decided that a translation between $a, b \in C_i$ is high-quality if and only if all of the languages used on the path from a to b are also in C_i. The translator set S is a high-quality translation network if and only if it is a translation network, and for any language cluster C_i and any languages $a, b \in C_i$, S can be used for a high-quality translation between a and b.

(d) Suppose that S is a minimum-cost high-quality translation network. Let $S_i = S \cap (C_i \times C_i)$ be the part of the network S that lies within the cluster C_i. Show that S_i is a minimum-cost translation network for the cluster C_i.

Solution: [5 points] Let S_i^* be a minimum-cost translation network for C_i. Because S is a high-quality translation network, S_i must contain a path between every pair of nodes in C_i, so S_i is a translation network for C_i. For the sake of contradiction, assume that S_i is not minimum-cost. Then $w(S_i) > w(S_i^*)$.

Consider the translation network $S^* = (S - S_i) \cup S_i^*$. Then $w(S^*) = w(S) - w(S_i) + w(S_i^*) < w(S)$. Because S_i^* is a translation network of C_i, replacing S_i with S_i^* will not disconnect any pair of vertices in the graph. Furthermore, any pair of vertices connected by a path in S that lay inside a particular cluster will be connected by a path in S^* that lies within the same cluster. Hence, S^* is a high-quality translation network with cost strictly less than S. This contradicts the definition of S, so S_i must be a minimum-cost translation network.

(e) Give an efficient algorithm for computing a minimum-cost high-quality translation network. Analyze the runtime of your algorithm in terms of the number of languages n and the number of translators m.

Solution: [5 points] The idea behind this algorithm is to first compute one MST for each individual cluster, and then to compute a global MST using the remaining edges. More specifically, we do the following:

1. For each edge (u, v), if there is some cluster C_i such that $u, v \in C_i$, then add (u, v) to the set E_i. Otherwise, add (u, v) to the set E_{global}.

2. For each cluster C_i, run Kruskal's algorithm on the graph (C_i, E_i) to get a minimum-cost translation network T_i for the cluster. Take the union of these minimum to get a forest T.
3. Construct an empty graph G_{global} on nodes $\{1, \ldots, k\}$. For each edge (u, v) in E_{global}, where $u \in C_i$ and $v \in C_j$, check whether the edge (i, j) is in the graph G_{global}. If so, set $w(i, j) = \min\{w(i, j), w(u, v)\}$. Otherwise, add the edge (i, j) to the graph G_{global}. In either case, keep a mapping $source(i, j) = (u^*, v^*)$ such that $w(i, j) = w(u^*, v^*)$.
4. Run Kruskal's algorithm on the graph G_{global} to get T_{global}.
5. For each edge $(i, j) \in T_{global}$ add the edge $source(i, j)$ to T.

We begin by examining the runtime of this algorithm. The first step requires us to be able to efficiently discover the cluster C_i that contains each vertex, which can be precomputed in time $\Theta(m)$ and stored with the vertices for efficient lookup. So this filtering step requires $\Theta(1)$ lookup per edge, for a total of $\Theta(m)$ time.

The second step is more complex. We run Kruskal's algorithm on each individual cluster. So for the cluster C_i, the runtime is $\Theta(|E_i| \lg |C_i!)$. The total runtime here is:

$$\sum_{i=1}^{k} a \cdot |E_i| \lg |C_i| \leq \sum_{i=1}^{k} a \cdot |E_i| \lg n = (a \lg n) \sum_{i=1}^{k} |E_i| \leq am \lg n$$

Hence, the total runtime for this step is $\Theta(m \lg n)$.

The third step also requires care. We can store the graph to allow us to efficiently lookup $w(\cdot, \cdot)$ and to tell whether an edge (i, j) has been added to the graph. We can also store $source(\cdot, \cdot)$ to allow for $\Theta(1)$ lookups. So the runtime here is bounded by $\Theta(m)$. The fourth step is Kruskal's again, only once, on a graph with $\leq n$ vertices and $\leq m$ edges, so the total runtime is $\Theta(m \lg n)$. The final step involves a lookup for each edge $(i, j) \in T$, for a total runtime of $\Theta(k) \leq \Theta(n)$. Hence, the runtime is dominated by the two steps involving Kruskal. So the total worst-case runtime is $\Theta(m \lg n)$.

Next we consider the correctness of this algorithm. Suppose that the result of this process is not the minimum-cost high-quality translator network. Then there must be a high-quality translator network S that has strictly smaller cost. Because S is a minimum-cost high-quality translator network, we know that the portion of S contained in the cluster C_i is a minimum-cost translator network for C_i, which has the same cost as the minimum spanning tree for that cluster computed in step 2 of the algorithm. So the total weight of all inter-cluster edges in S must be strictly less than the total weight of all inter-cluster edges in T. But the set of all inter-cluster edges in T formed a minimum spanning tree on the cluster, so any strictly smaller set of edges cannot span the set of all clusters. So S cannot be a translation network. This contradicts our assumption, and so T must be a minimum-cost high-quality translation network.

Problem 3. All Pairs Shortest Red/Blue Paths. [18 points] (3 parts)

You are given a directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$. In addition, each edge of the graph is either red or blue. The shortest red/blue path from vertex $i \in V$ to vertex $j \in V$ is defined as the shortest path from i to j among those paths that go through *exactly* one red edge (if there are no such paths, the length of the shortest red/blue path is ∞).

We can represent this graph with two $n \times n$ matrices of edge weights, W_r and W_b, where W_r contains the weights of all red edges, and W_b contains the weights of all blue edges.

(a) Given the Floyd-Warshall algorithm below, how would you modify the algorithm to obtain the lengths of the shortest paths that only go through blue edges?

```
FLOYD-WARSHALL($W$):
1  $n = W.rows$
2  $D^{(0)} = W$
3  for $k = 1$ to $n$
4      let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5      for $i = 1$ to $n$
6          for $j = 1$ to $n$
7              $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8  return $D^{(n)}$
```

Solution: [6 points] In order to find shortest paths going through only blue edges, it suffices to ignore the red edges and run Floyd-Warshall on only the blue edges of the graph. We make the following changes:

- Replace each occurrence of W with W_b in lines 1 and 2.
- Replace the matrices $D^{(k)}$ with blue versions $D_b^{(k)}$ in lines 2, 4, and 8.
- Replace the matrix elements $d_{ij}^{(k)}$ with blue versions $d_{b,ij}^{(k)}$ in lines 4 and 7.

While the second and third changes above are unnecessary for this part, they lay the groundwork for future parts below.

(b) How would you modify your algorithm from part (a) to keep track not only of shortest paths with only blue edges, but also those with exactly one red edge, and to output the lengths of the shortest red/blue paths for all pairs of vertices in this graph?

Solution: [6 points] Add a new set of matrices $D_r^{(k)}$ that give lengths of shortest paths with exactly one red edge and intermediate vertices up to k. The resulting pseudocode is as follows:

```
RED-BLUE-FLOYD-WARSHALL($W_r, W_b$):
1   $n = W_r.rows$
2   $D_b^{(0)} = W_b$
3   $D_r^{(0)} = W_r$
4   for $k = 1$ to $n$
5       let $D_b^{(k)} = (d_{b,ij}^{(k)})$, $D_r^{(k)} = (d_{r,ij}^{(k)})$ be new $n \times n$ matrices
6       for $i = 1$ to $n$
7           for $j = 1$ to $n$
8               $d_{b,ij}^{(k)} = \min(d_{b,ij}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{b,kj}^{(k-1)})$
9               $d_{r,ij}^{(k)} = \min(d_{r,ij}^{(k-1)}, d_{r,ik}^{(k-1)} + d_{b,kj}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{r,kj}^{(k-1)})$
10  return $D_r^{(n)}$
```

(c) Prove the correctness of your algorithm using a loop invariant.

Solution: [6 points] The procedure for keeping track of paths that go through only blue edges is exactly equivalent to running Floyd-Warshall on the subgraph that contains only the blue edges of G, which is sufficient to show the correctness of $D_b^{(k)}$ for all k.

Loop invariant: at the end of every iteration of the for loop from $k = 1$ **to** n, we have both the length of the shortest path for every pair (i, j) going through only blue edges and the length of the shortest path for every pair (i, j) going through exactly one red edge, in each using intermediate vertices only up to k.

Initialization: At initialization $k = 0$, there are no intermediate vertices. The only blue paths are blue edges, and the only paths with exactly one red edge (red/blue paths) are red edges, by definition.

Maintenance: Each iteration gets the shortest blue-edges-only path going from i to j using intermediate vertices up through k accurately, due to the correctness of the Floyd-Warshall algorithm.

For the paths including exactly one red edge, for a given pair (i, j), there are two cases: either the shortest red/blue path from i to j using intermediate vertices through k does not go through k, or it does. If it does not go through k, then the length of this path is equal to $d_{r,ij}^{(k-1)}$. If it does go through k, then the red edge on this path is either between i and k or between k and j. Because of the optimal substructure of shortest paths, we can therefore break this case down into two subcases: the shortest path length is equal to $\min(d_{r,ik}^{(k-1)} + d_{b,kj}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{r,kj}^{(k-1)})$. Line 9 in the algorithm above finds the minimum path length of these three different possibilities, so this iteration must find the shortest path length from i to j going through exactly one red edge, using only intermediate vertices through k.

Termination: After n passes through the loop, all paths with intermediate vertices up to n have been included, and as there are only n vertices, shortest paths using all vertices as intermediates will be discovered.

Problem 4. Telephone Psetting. [12 points] (3 parts)

Upon realizing that it was 8:30 PM on Wednesday and he had not yet started his 6.046 pset, Ben Bitdiddle found $n-1$ other students (for a total of n students) in the same situation, and they decided to do the pset, which coincidentally had n problems in total, together.

Their brilliant plan for finishing the pset in time was to sit in a circle and assign one problem to each student, so that for $i = 0, 1, \ldots, n-1$, student i did problem number i, and wrote up a solution for problem i meriting $p(i)$ points. Then, they each copied the solutions to all the other problems from the student next to them, so that student i was copying from student $i-1$ (and student 0 was copying from student $n-1$).

Unfortunately, they were in such a hurry that the copying chain degraded the quality of the solutions: by the time student i's solution to problem i reached student j, where $d = j - i \ (mod\ n)$, $d \in \{0, 1, \ldots, n-1\}$, the solution was only worth $\frac{1}{d+1}p(i)$ points.

(a) Write a formula that describes the total pset score $S(x)$ for student x, where the total pset score is the sum of the scores that student x got on each of the n problems.

Solution: [3 points] $S(x) = \displaystyle\sum_{i=0}^{n-1} \frac{1}{((x-i) \mod n) + 1} \cdot p(i)$

Alternatively, it is also correct to give the equivalent sum:

$$S(x) = \sum_{i=0}^{n-1} \frac{1}{i+1} \cdot p((x-i) \mod n)$$

(b) Describe a simple $O(n^2)$ algorithm to calculate the pset scores of all the students.

Solution: [3 points] Use the formula given in part (a) to calculate each student's score. Because calculating the score requires summing n numbers, it takes $O(n)$ time to calculate a single score, and therefore $O(n^2)$ time to calculate all the scores.

(c) Describe a $O(n \log n)$ algorithm to calculate the pset scores of all the students.

Solution: [6 points] The formula in the solution to part (a) describes a convolution: letting $g(y) = \frac{1}{y+1}$, the formula in part (a) can be written as $S : \mathbb{Z}_n \to \mathbb{R}$, where $S(x) = \sum_{i \in \mathbb{Z}_n} p(i) g(x-i) = (p \otimes g)(x)$. The algorithm is as follows:

1. Apply FFT on p and g to get $\hat{p}$ and $\hat{g}$ (time $\Theta(n \log n)$).
2. Compute the transformed convolution $\hat{S} = \hat{p} \cdot \hat{g}$ (time $\Theta(n)$).
3. Apply the inverse FFT to get back the convolution S (time $\Theta(n \log n)$).

Alternatively, define two polynomials:
$P_1(x) = \sum_{i=0}^{2n-1} p(i \mod n)x^i$ and $P_2(x) = \sum_{i=0}^{n-1} \frac{1}{i+1}x^i$. Then student j's pset score is equal to the coefficient of x^{n+j} in the product of the polynomials P_1 and P_2. Because we are multiplying two polynomials of degree at most $2n - 1$, we can apply the polynomial multiplication method seen in class, which is outlined above: apply the FFT to get the evaluations of P_1 and P_2 on the roots of unity of order $2n$, pointwise multiply, and finally apply the inverse FFT to get the coefficients of the product.

SCRATCH PAPER