Solutions -- Midterm 2021 (Grad Algorithms)

Question 1.

Consider the recurrence $T(n) = nT(\sqrt{n}) + n, T(1) = 1$.

Solution. $T(n) = O(n^2 \log n)$ and $T(n) = O(n^3)$ are both correct, $T(n) = O(n \log n)$ is not.

First, note that $T(n) \geq n$ by definition (it's just the second term in the recurrence). Thus $T(\sqrt{n}) \geq \sqrt{n}$, which then implies that $T(n) \geq n^{3/2}$, and thus $O(n \log n)$ can't be a valid answer.

Second, suppose that $T(k) \leq Ck^2 \log k$ holds for numbers $k < n$, with some setting of C. (It certainly holds for $k = 2$). We will show that it also holds for n. This is because we can plug the bound we get for $k = \sqrt{n}$, and obtain $T(n) = nT(\sqrt{n}) + n \leq n(Cn \log \left(\sqrt{n}\right)) + n \leq Cn^2 \frac{\log n}{2} + n \leq Cn^2 \log n$, because the n term is of smaller order.

This shows that $T(n) = O(n^2 \log n)$, which also implies that $T(n) = O(n^3)$.

Alternate solution. You can actually do plug and chug here...

$T(n) = n + nT(n^{1/2}) = n + n(n^{1/2} + T(n^{1/4}))$

$= n + n^{1+1/2} + n^{1+1/2} \cdot T(n^{1/4})$

$= n + n^{1+1/2} + n^{1+1/2+1/4} + n^{1+1/2+1/4} \cdot T(n^{1/8})$

= ...

Suppose this continues until $n^{1/2^r}$ becomes 1. Then we can bound the entire sum by simply the last two terms (because the last terms are dominant). This gives a bound $2n^{1+1/2+1/4+\ldots} = 2n^2$.

If you don't want to use the fact that the last terms dominate, you can do an easier bound -- each term $\leq n^2$, and there are $\log \log n$ terms, so we get a bound $T(n) \leq n^2 \log \log n$ (for $n \geq 4$).

Question 3.

The idea is to mimic merge-sort, and if the two elements at the "head" are equal, we increment the count. (Assume that the arrays are zero-indexed.)

```
- maintain two indices a, b = 0, answer=0
- while (a < m && b < n):
    if (A[a] == B[b]), increment a, b, and answer
    else if (A[a] < B[b]), increment a
    else if (A[a] > B[b]), increment b
- return answer
```

In each step, either 'a' or 'b' (or both) are incremented. Since we stop if either $a = m$ or $b = n$, the total number of steps is $< m + n$.

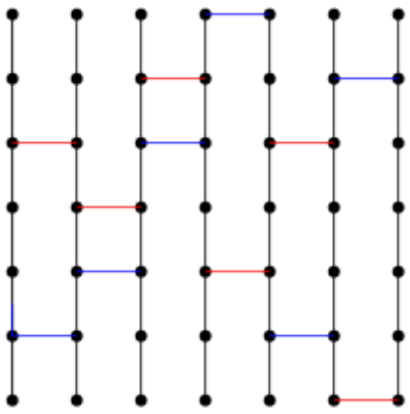
Question 4.

For each element x in A, perform a binary search to find x in the (sorted) array B. If you find x, increment the answer. As the time for each binary search is $O(\log n)$, the total running time is $O(m \log n)$.

Question 5.

The answer is exponential in n (and therefore none of the other options can be correct). There are many ways to see this, but here is an easy one:

Consider only spanning trees that have vertical lines but with different positions for the "connections" between adjacent lines. For example, the figure below shows two trees, the black edges are the vertical lines that are part of both trees. Apart from these, one tree has the blue edges and another has the red edges.



For each connection, there are n options, and we have $(n-1)$ such connections, so the number of trees is $\geq n^{(n-1)}$, which is $\exp(n)$.

Question 7.

The algorithm is simple once we use the selection algorithm (linear time). First, I'll assume that the (n/10)th and (9n/10)th elements are unique.

```
% Algorithm assuming the n/10th and 9n/10th elements are unique

- Compute L = (n/10)th smallest element of A, and U = (9n/10)th
smallest element of A using the O(n) selection algorithm.
- initialize sum = 0, num_vals = 0;
- For i = 0, ..., n-1:
    - if A[i] > L and A[i] < U, set sum = sum+A[i] and increment
num_vals.
- Return sum / (num_vals).
```

In general, we need to have a pesky correction term. One way to do this is to count the number of elements in A that are $\leq L$. If this number (call it k) is $> n/10$, the correction term on the 'lower' side is $(k-(n/10))L$. Similarly you get a correction term on the right side. These need to be added to the sum and the answer should be divided by (8n/10).

(Another case to deal with is $L = U$, in which case the answer is L.)

Grading. You will receive full credit even if you don't consider these correction terms.

Question 8.

The rounding is not valid. Consider the example with the fractions being 0.6, 0.6, 0.6, 0.2. The sum of the fractions is 2. The trivial rounding gives 1, 1, 1, 0, whose sum is 3.

Question 10.

The idea is simple, either we round the ith fraction to 0 or to 1. In the first case, the objective value is $\max(f_i, BestRound(i+1, s))$, and in the second, the value is $\max(1-f_i, BestRound(i+1, s-1))$. Define the first of these to be v_0 and the second of these as v_1. Then the recurrence is:

BestRound(i, s) = min $\{v_0, v_1\}$, using the expressions defined above.

The base cases are: $i = n$ and $s = 0$. When $s = 0$, we are forced to round the fractions $i, \ldots, n$ to 0 so we can return $\max(f_i, f_{i+1}, \ldots, f_n)$. And for $i = n$, we do the following: if $s = 1$ we return $(1 - f_n)$, if $s = 0$ we return f_n, and otherwise, we return ∞).

Question 11.

The number of possible values of the first parameter is n and that for the second parameter is k. Thus the total number of sub-problems is $O(nk)$. Since k can be much smaller than n, $O(k^2)$ is not a valid answer.

Question 13.

The algorithm is not optimal. Consider the points $A = \{(1, 0), (2, 0)\}$ and $B = \{(-1, 1), (0, 1), (1, 1)\}$. We will first match (1, 0) with (1, 1), and then we are forced to match (2, 0) with (0,1). This incurs a cost of $1 + \sqrt{5} \approx 3.236$. On the other hand, suppose we match (1, 0) with (0, 1) and (2, 0) with (1, 1), then we incur a cost of $2\sqrt{2} \approx 2.828$.

(You don't really need to do the calculation. You can simply observe that "uncrossing" leads to a smaller value due to the triangle inequality.)

Question 14.

In the counterexample above, we can remove the point (-1, 1) from B and the discussion will still hold, so having $m = n$ doesn't really help.

But meanwhile, if $m = n$, the only solution without "crossing" edges is the one given by the lazy algorithm. As mentioned above, the optimal solution cannot have two matching pairs that cross (e.g., in the figure below, replacing ad and bc with ac and bd will result in a smaller total length). Thus the optimal solution is indeed the lazy one.