CS/ECE 374 A (Spring 2022)
Midterm 2 Solutions

---

1. (a) Answer: $\Theta(n^2)$.

   Justification: by the master theorem, since $n + 4n^2 = \Theta(n^2)$ has a higher growth rate than $n^{\log_2 3+\varepsilon}$.

   [Alternative justification: by unfolding the recurrence (and ignoring floors), $T(n) = 3T(n/2)+O(n^2) = 9T(n/4)+O(3(n/2)^2+n^2) = \cdots = 3^kT(n/2^k)+O(\sum_{i=0}^{k-1} 3^i(n/2^i)^2)$. Setting $k = \log_2(n/374)$, we get $T(n) \leq 3^{\log_2(n/374)}T(374) + O(n^2\sum_i(3/4)^i) = O((n/374)^{\log_2 3} + n^2) = O(n^2)$. (And trivially, $T(n) = \Omega(n^2)$.)]

   (b) True.

   Justification: The median-of-medians-of-5 algorithm runs in $O(n)$ time, while mergesort runs in $\Theta(n \log n)$ time. (And $n = o(n \log n)$.)

   (c) $O(n)$.

   (d) $O(n^2)$.

   Justification: X has $O(n)$ bits. Thus, line 3 takes $O(n)$ time. So, the total time is $O(n^2)$. [Or, to be more precise: X has $\Theta(i)$ bits at iteration i. So, the total time is $\Theta(\sum_{i=1}^{n} i) = \Theta(n^2)$.]

   (e) False.

   Counterexample: take any directed graph that contains a cycle, and take any edge (u, v) that is a back edge.

   [Or more concretely, take a directed graph with 3 vertices forming a cycle a, b, c, a. The DFS tree starting at a yields a back edge (c, a), and c finishes before a.]

   (f) True.

   Justification: in a DAG, there are no back edges.

   (g) True.

   Justification: take any sequence of vertices $v_0, v_1, v_2, \ldots$ where v_{i+1} is an out-neighbor of v_i (which must exist when the out-degree of v_i is at least one). Some vertex must be repeated in this sequence, yielding a cycle.

   [Or: Suppose G does not have a cycle. Then it is a DAG. And as we know, any DAG has a sink with out-degree 0: contradiction.]

   (h) Repeatedly run BFS from every vertex (since BFS solves the single-source shortest paths problem for an unweighted graph). Since each BFS takes $O(m + n)$ time, the total time for the n runs is $O(n(m + n)) = O(n^2)$ when $m = O(n)$. And $O(n^2)$ time is the fastest possible (since the output size is $\Theta(n^2)$).

   [Note: Floyd–Warshall would require $O(n^3)$ time, which is slower; running Dijkstra n times would require $O(n^2 \log n)$ time, which is slightly slower.]

2. The idea is to recursively split the first half of the array and the second half of the array, which gives

$$\langle A[1], A[3], \ldots, A[n/2-1], A[2], A[4], \ldots, A[n/2],$$
$$A[n/2+1], A[n/2+3], \ldots, A[n-1], A[n/2+2], A[n/2+4], \ldots, A[n]\rangle,$$

and then swap the second and third quarters of the array to yield

$$\langle A[1], A[3], \ldots, A[n/2-1], A[n/2+1], A[n/2+3], \ldots, A[n-1],$$
$$A[2], A[4], \ldots, A[n/2], A[n/2+2], A[n/2+4], \ldots, A[n]\rangle,$$

as desired.

Pseudocode.

split($A[1, \ldots, n]$):
1. if $n = 1$ return
2. split($A[1, \ldots, n/2]$)
3. split($A[n/2+1, \ldots, n]$)
4. for $i = 1$ to $n/4$ do *[now swap the second and third quarters]*
5.   swap $A[n/4+i]$ with $A[n/2+i]$

Analysis. Let $T(n)$ be the running time of split($A[1, \ldots, n]$). Lines 2–3 take $2T(n/2)$ time. Lines 4–5 take $O(n)$ time. So,

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

This is the same recurrence as mergesort, and so $T(n) = O(n \log n)$.

The swap in line 5 only requires one temporary variable. So, lines 4–5 require only $O(1)$ extra space. So, the whole algorithm requires only $O(1)$ extra space (minor technicality: if one takes the recursion stack into account, the space usage is actually $O(\log n)$, but it can be made $O(1)$ with a more careful implementation).

(*Note.* There are actually $O(n)$-time algorithms for this kind of problem about permuting an array with $O(1)$ extra space, by using more advanced ideas (chasing cycles)...)

3. (a) *Pseudocode.*

for $i = 1$ to n do $C[i, -1] = -\infty$
for $i = n$ down to 1 do *[Evaluation order: decreasing i.]*
    for $k = 0$ to K do
        $C[i, k] = 1$
        for $j = i + 1$ to n do
            if $a_j > a_i$ then $C[i, k] = \max\{C[i, k], C[j, k-1] + 1\}$
            else $C[i, k] = \max\{C[i, k], C[j, k] + 1\}$
return $\max_{i \in \{1, \ldots, n\}} C[i, K]$

Run time analysis. Lines 1 take $O(n)$ time. Lines 2–7 take $O(n^2K)$ time. Line 8 takes $O(n)$ time. Total time: $O(n^2K)$.

(b) *Definition of subproblems.* For each $i \in \{1, \ldots, n\}$ and $k \in \{-(n-1), \ldots, n-1\}$ and $\ell \in \{0, \ldots, L\}$, let $C(i, k, \ell)$ be the sum of the max-sum subsequence of $\langle a_i, \ldots, a_n \rangle$ such that the first element is a_i and the number of up-turns minus the number of down-turns is exactly k, and the length is exactly ℓ.

The answer we want is $\max_{i \in \{1,\ldots,n\}} C(i, 0, L)$.

Base case. $C(i, -n, \ell) = C(i, n, \ell) = -\infty$ for all $i \in \{1, \ldots, n\}$ and $\ell \in \{0, \ldots, L\}$. And $C(i, k, -1) = -\infty$ for all $i \in \{1, \ldots, n\}$ and $k \in \{-(n-1), \ldots, n-1\}$.

Recursive formula. For each $i \in \{1, \ldots, n\}$ and $k \in \{-(n-1), \ldots, n-1\}$ and $\ell \in \{0, \ldots, L\}$,

$$C(i, k, \ell) = \max \begin{cases} a_i \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } a_j > a_i} (C(j, k-1, \ell-1) + a_i) \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } a_j < a_i} (C(j, k+1, \ell-1) + a_i) \end{cases}$$

Evaluation order. Decreasing i. (Alternatively: increasing ℓ would also work.)

Run time analysis. The number of subproblems or table entries is $O(n^2L)$. The cost to compute each entry is $O(n)$. Total time: $O(n^3L)$.

4. (a) Construct the strongly connected components (or the meta-graph) by the known algorithm from class (by Kosaraju and Sharir). Return yes iff there is a component that has at least $n/2$ vertices.

Justification. If a component has at least $n/2$ vertices, then there is a closed walk visiting all vertices of the component and thus visits at least $n/2$ distinct vertices. Conversely, if there is a closed walk visiting at least $n/2$ distinct vertices, the walk resides in a component which has at least $n/2$ vertices.

Runtime. The strongly connected components algorithm takes $O(m)$ time. The components' sizes can be computed in $O(n)$ time. Total: $O(m)$.

(b) Given $G = (V, E)$, we construct a new weighted directed graph G' as follows:

- For each $v \in V$, create two new vertices $(v, 0)$ and $(v, 1)$ in G'.
- For each edge $(u, v) \in E$ with weight $w(u, v) > 0$, create an edge from $(u, 0)$ to $(v, 1)$ with weight $-w(u, v)$.
- For each edge $(u, v) \in E$ with weight $w(u, v) < 0$, create an edge from $(u, 1)$ to $(v, 0)$ with weight $-w(u, v)$.

Run a single-source shortest path algorithm to compute shortest paths from $(s, 0)$ to $(t, 0)$, from $(s, 0)$ to $(t, 1)$, from $(s, 1)$ to $(t, 0)$, and from $(s, 1)$ to $(t, 1)$ in G'. If all four paths have nonnegative total weight, return no. Otherwise, take one of these paths with negative total weight and return the corresponding walk in G.

(*Justification.* A path in G' corresponds to a walk in G that alternates between positive- and negative-weight edges, and the weight of the path in G' is the negation of the weight of the corresponding walk in G. Thus, there is a walk from s to t in G of positive total weight satisfying the alternation condition iff one of the four shortest paths in G' have negative total weight.)

Runtime. The graph has $N = 2n$ vertices and $M = m$ edges. By the Bellman–Ford algorithm, the running time for each of the four shortest path computations is $O(MN) = O(mn)$. Total time: $O(mn)$.

(Alternatively, we could use Floyd–Warshall's APSP algorithm. The running time would be $O(n^3)$, which is worse for sparse graphs. We can't use Dijkstra's algorithm, because of negative weights.)