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

---

1. (a) Answer: $\Theta(n^{\log_2 3})$.

   Justification: by the master theorem, since $n^{3/2}$ has a lower growth rate than $n^{\log_2 3-\varepsilon}$ (even without a calculator, one can see $3/2 < \log_2 3$, since $2^{3/2} < 3$, i.e., $2^3 < 3^2$).

   [Alternative justification: by unfolding the recurrence (and ignoring floors), $T(n) = 3T(n/2)+n^{3/2} = 9T(n/4)+3(n/2)^{3/2}+n^{3/2} = \cdots = 3^k T(n/2^k)+\sum_{i=0}^{k-1} 3^i(n/2^i)^{3/2} = 3^k T(n/2^k)+n^{3/2}\sum_{i=0}^{k-1}(3/2^{3/2})^i = 3^k T(n/2^k)+\Theta(n^{3/2}(3/2^{3/2})^k)$. Setting $k = \log_2(n/2022)$, we get $T(n) \le \Theta(3^k T(2022)+n^{3/2}n^{\log_2(3/2^{3/2})}) = \Theta(n^{\log_2 3}+n^{\log_2 3}) = \Theta(n^{\log_2 3})$.]

   (b) False.

   Justification: Quicksort has $O(n \log n)$ expected running time, and may occasionally run in $O(n^2)$ time, but mergesort always runs in $O(n \log n)$ time.

   (c) $O(n)$.

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

   Justification: X, Y, Z are $O(n)$ bits long, since $F_n \le 2^n$. 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) True.

   Justification: if there is an edge between a vertex u at level i and a vertex v at level $i+3$, then v would have been placed at level $i+1$ (or earlier) by BFS: contradiction.

   [Or: the level of a vertex v in a BFS tree rooted at s is just the shortest-path distance $\delta(s,v)$ from s to v. But if (u,v) is an edge, then $\delta(s,v) \le \delta(s,u)+1$. So, if the level of u is i, then the level of v is at most $i+1$.]

   (f) n.

   Justification: in a DAG, the strongly connected components are just the n singletons (sets of size 1).

   (g) True.

   Justification: If G does not have a cycle, then G is an undirected acylic graph, i.e., a forest, and so $m \le n-1$, where m is the number of edges and n is the number of vertices. But if every vertex has degree at least two, then the total degree is at least $2n$ and so $m \ge n$: contradiction.

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

   (h) Repeatedly run Dijkstra's single-source shortest paths algorithm from every source vertex. Since Dijkstra's algorithm (with a Fibonacci heap implementation) takes $O(n \log n + m)$ time, the total time for the n runs is $O(n(n \log n + m)) = O(n^{5/2})$ when $m = n^{3/2}$. [Note: Floyd–Warshall would require $O(n^3)$ time, which is slower.]

2. Note the correction: "find the *minimum* k such that..." (for otherwise the problem would be trivial).

   We assume that all a_i's are positive (because negative elements don't help and may be removed). And we assume that $a_1 + \cdots + a_n \geq S$ (for otherwise there would be no answer).

   Pseudocode. The following procedure returns the minimum such k:

   search($\{a_1, \ldots, a_n\}, S$):
   1. if $n = 1$ then return 1
   2. let m be the $\lceil n/2 \rceil$-th largest in $\{a_1, \ldots, a_n\}$
   3. let $L = \{a_i : a_i < m\}$ and $R = \{a_i : a_i \geq m\}$
   4. let x be the sum of the elements in R
   5. if $x \geq S$ then return search(R, S)
   6. else return search($L, S - x$) + $\lceil n/2 \rceil$

   Analysis. Let $T(n)$ be the running time of search($\{a_1, \ldots, a_n\}, S$). Line 2 takes $O(n)$ time by the linear-time median-finding (or selection) algorithm from class. Lines 3 and 4 takes $O(n)$ time by a linear scan. Only one recursive call is made in lines 4–5, which take at most $T(\lceil n/2 \rceil)$ time. So,

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

   Unrolling the recurrence (and ignoring ceilings) gives $T(n) = O(n + n/2 + n/4 + \cdots) = O(n)$ by a geometric series. So, the algorithm runs in $O(n)$ time.

   [Alternatively, we can invoke the master theorem (since n obviously has a higher growth rate than $n^{\log_2 1 + \varepsilon}$, as $\log_2 1 = 0$).]

3. (a) *Pseudocode.*

   1. for $i = n$ down to 1 do *[Evaluation order: decreasing i.]*
   2. $\quad C[i] = 1$
   3. $\quad\quad$ for $j = i + 1$ to n do
   4. $\quad\quad\quad$ if $d(p_i, p_j) \leq L$ then $C[i] = \max\{C[i], C[j] + 1\}$
   5. return $\max_{i \in \{1,\ldots,n\}} C[i]$

   Run time analysis. Lines 1–4 take $O(n^2)$ time. Line 5 takes $O(n)$ time. Total time: $O(n^2)$.

   (b) *Definition of subproblems.* For each $i \in \{1, \ldots, n\}$ and $k \in \{0, \ldots, r\}$, let $C(i, k)$ be the max value of the optimal subsequence of $\langle a_i, \ldots, a_n \rangle$ that starts at a_i and has exactly k consecutive pairs of distance greater than L.

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

   Base case. $C(i, -1) = -\infty$ for all $i \in \{1, \ldots, n\}$.

   Recursive formula. For each $i \in \{1, \ldots, n\}$ and $k \in \{0, \ldots, r\}$,

$$C(i,k) = \max \begin{cases} 0 \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } d(p_i,p_j) > L} (C(j, k-1) + d(p_i, p_j)) \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } d(p_i,p_j) \leq L} (C(j, k) + d(p_i, p_j)) \end{cases}$$

Evaluation order. Decreasing i.

Run time analysis. The number of subproblems or table entries is $O(nr)$. The cost to compute each entry is $O(n)$. Total time: $O(n^2 r)$. [Note: $O(n^3)$ would also be correct.]

4. (a) Define a new weighted DAG G' as follows:

   - The vertices and the edges are the same as the given DAG G.
   - For each edge $(u, v) \in E$, define the weight of (u, v) in G' to be -3 if (u, v) is red, and $+1$ if (u, v) is blue.

   We compute the shortest path from s to t in G'. The answer is yes iff the shortest path has weight ≤ 0.

   Justification. "at least 25% of the edges are red" is equivalent to "the number of blue edges is at most 3 times the number of red edges".

   Run time analysis. The graph G' has n vertices and m edges, and can obviously be constructed in $O(m + n)$ time. As explained in class, there is a single-source shortest path algorithm for DAGs by dynamic programming with $O(m + n)$ running time. Total time: $O(m + n)$.

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

   - For each $v \in V$ and each $i \in \{-n, \ldots, n\}$, create a new vertex (v, i) in G'.
   - For each edge $(u, v) \in E$ with v red and each $i \in \{-n, \ldots, n-1\}$, create an edge from (u, i) to $(v, i + 1)$ with weight 1 in G'.
   - For each edge $(u, v) \in E$ with v blue and each $i \in \{-(n-1), \ldots, n\}$, create an edge from (u, i) to $(v, i - 1)$ with weight 0 in G'.

   Run a single-source shortest path algorithm to compute the shortest path from $(s, 0)$ to (t, i) in G' for every i. Take the shortest among these paths over all $i \in \{1, \ldots, n\}$ if s is blue, and over all $i \in \{0, \ldots, n\}$ if s is red, and return the corresponding walk in G.

   (*Justification.* A path in G' from $(s, 0)$ to (t, i) corresponds to a walk in G from s to t where i equals the number of red vertices minus the number of blue vertices, excluding s itself. The weight of the path in G' corresponds to the number of red vertices in the walk, excluding s itself.)

   Runtime. The graph has $N = O(n^2)$ vertices and $M = O(mn)$ edges, and can be constructed in $O(n^2 + mn)$ time. Dijkstra's single-source shortest path algorithm (which is applicable here since all weights are nonnegative) has running time $O(N \log N + M) = O(n^2 \log n + mn)$. Total: $O(n^2 \log n + mn)$.
   (Actually, since all the edge weights in G' are 0 or 1, a variant of BFS can do slightly better: $O(N + M) = O(n^2 + mn)$.)