Answers to Quiz 3

Distribution of scores  Average ≈ 13.28. Median = 14. Note: For the histogram, fractional scores were rounded down.

2–3:  2
4–5:  5
6–7:  7 7 7
8–9:  8 8 9 9 9 9 9 9
10–11: 10 10 10 10 11 11 11 11 11
12–13: 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13
14–15: 14 14 14 14 14 14 14 14 14 14 15 15 15 15 15
16–17: 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17
18–19: 18 18 18 18 18 18 18 19 19 19 19 19
20:  20 20

Problem 1. (6 points) Background: A two-coloring of an undirected graph is an assignment of colors (Red or Black) to the vertices so that no two adjacent vertices have the same color. Your problems:

a. (4 points) Modify DFS to devise an O(|V| + |E|)-time algorithm that, given a connected undirected graph G, finds a two-coloring of G if there is one, or else reports NONE, if there is no such coloring of G.

b. (2 points) Justify the O(|V| + |E|) run time bound. Hint: There are not very many possible two-colorings.

An answer for 1a: In a two-coloring of a graph, each edge must has both a RED and a BLACK endpoint. Hence, in a connected two-colorable graph, assigning a color to any vertex completely determines the colors of every other vertex. It thus suffices to do is DFS of the graph assigning alternating colors down paths; either we construct a two-coloring or else the search finds monochrom edge and reports NONE. Here is the detailed algorithm:

procedure twoCol(G)
    for each vertex u of G do color[u] ← BLANK; colorable ← true
    for each vertex u of G do
        if (color[u] = BLANK) then dfsVisit(u, RED)
        if colorable then return color else return NONE

procedure dfsVisit(u, tint1)
    color[u] ← tint1
    if (tint1 = RED) then tint2 ← BLACK else tint2 ← RED
    for each vertex v adjacent to u do
        if color[v] = BLANK then dfsVisit(v, tint2)
        else if color[v] = tint1 then colorable ← false

An answer for 1b: Since this is just a modification of DFS, the O(|V| + |E|) runtime follows from the analysis for DFS.

Problem 2. (8 points) Suppose G = (V, E) is a directed graph. The reverse of G is the graph $G^R = (V, E^R)$ where $E^R = \{(b, a) \mid (a, b) \in E\}$.

a. (4 points) Give an algorithm for computing $G^R$ from G where graphs are represented via adjacency lists. State and justify the $O(\cdot)$-runtime of your algorithm.

b. (4 points) Give an algorithm for computing $G^R$ from G where graphs are represented via adjacency matrices. State and justify the $O(\cdot)$-runtime of your algorithm.

An answer for 2a: We just scan down G's adjacency lists and build $G^R$'s adjacency lists as we go. Here is the algorithm. We assume $V = \{1, \ldots, n\}$.

function rev(adj[1..n])
    adjR ← an n-element array initialized to all nulls
    for i ← 1 to n do
        for each u in adjG[i] do add i to adjR[u] (*)
    return adjR

Runtime analysis: The line marked-*) is executed once per vertex and the line marked-*) is executed once per edge. Hence, the total time is $\Theta(|V| + |E|)$.

An answer for 2b: We just take the transpose of G's adjacency matrix. Here is the code. We assume $V = \{1, \ldots, n\}$.

function rev(AG[1..n, 1..n])
    AR ← a new n x n element array
    for i ← 1 to n do
        for j ← 1 to n do
            AR[i, j] ← AG[j, i]
    return AR

The nested-for loops take $\Theta(n \times n) = \Theta(|V|^2)$ time.

Problem 3. (6 points) Suppose we have airports numbered 1, 2, . . . , n and a function, $arrival$, such that

$$arrival(t, i, j) = \begin{cases} \text{the arrival time at } j, & \text{given you arrived at } i \text{ at time } t \text{ and took the} \\ \text{next possible flight from } i \text{ to } j \end{cases}$$

Note 1. Time is the number of minutes since some fixed point in the past.

Note 2. If there are no direct flights from $i$ to $j$, then $arrival(t, i, j) = +\infty$.

Note 3. For all times $t$: $arrival(t, i, i) = t$ and $arrival(t, i, i) > t$ when $i \neq j$.

Note 4. For all times $t$ and $t'$: if $t < t'$, then $arrival(t, i, j) \leq arrival(t', i, j)$.

Your problem: Modify Dijkstra's algorithm to build a table of the shortest traveling times from airport 1 to each of airports 2, . . . , n given that the traveler leaves at a pre-determined time $t_0$. 
Dijkstra’s Algorithm. Given: $G = ((1, \ldots, n), E)$ and a function $\text{len}$ that, for each $u, v \in \{1, \ldots, n\}$,
- if $(u, v) \in E$, then $\text{len}(u, v) > 0$, and
- if $(u, v) \notin E$, then $\text{len}(u, v) = \infty$.

Goal: For each $i \in \{2, \ldots, n\}$, compute $\text{dist}[i]$ = the length of a shortest path from 1 to $i$.

```
function Dijkstra(G, ℓ)
array dist[1..n]
ToDo ← \{2, \ldots, n\}
for i ← 2 to n do dist[i] ← len(1, i)
for i ← 1 to n − 1 do
  choose v ∈ ToDo with minimal dist[v]
  ToDo ← ToDo − \{v\}
  for each w ∈ ToDo do
    dist[w] ← min(dist[w], dist[v] + len(v, w))
return dist
```

A Version of Depth-First Search. $G$ is assumed to be represented by adjacency lists. Also, for each vertex $u$: $\pi[u]$ ends up with $u$’s parent in the DFS tree and $d[u]$ ends up with $u$’s discovery time.

```
procedure dfs(G)
for each vertex u of G do
  unvisited[u] ← true;  π[u] ← nil
  time ← 0
for each vertex u of G do
  if unvisited[u] then dfsVisit(u)
```

```
procedure dfsVisit(u)
unvisited[u] ← false
d[u] ← time
time ← time + 1
for each vertex v adjacent to u do
  if unvisited[v] then
    π[v] ← u
dfsVisit(v)
```