Solution - 7

1.

Adjacency list:

Q [] → S → W → T R [] → Y → U S [] → V T [] → X → Y U [] → Y V [] → W W [] → S X [] → Z Y [] → Q Z [] → X

Adjacency matrix for {v, s, w, q}:

	Q	**S**	**T**	**V**	**W**
Q	0	1	1	0	1
S	0	0	0	1	0
T	0	0	0	0	0
V	0	0	0	0	1
W	0	1	0	0	0

Adjacency matrix for the entire graph would have space complexity Θ(n2), so there would be 102 = 100 entries.

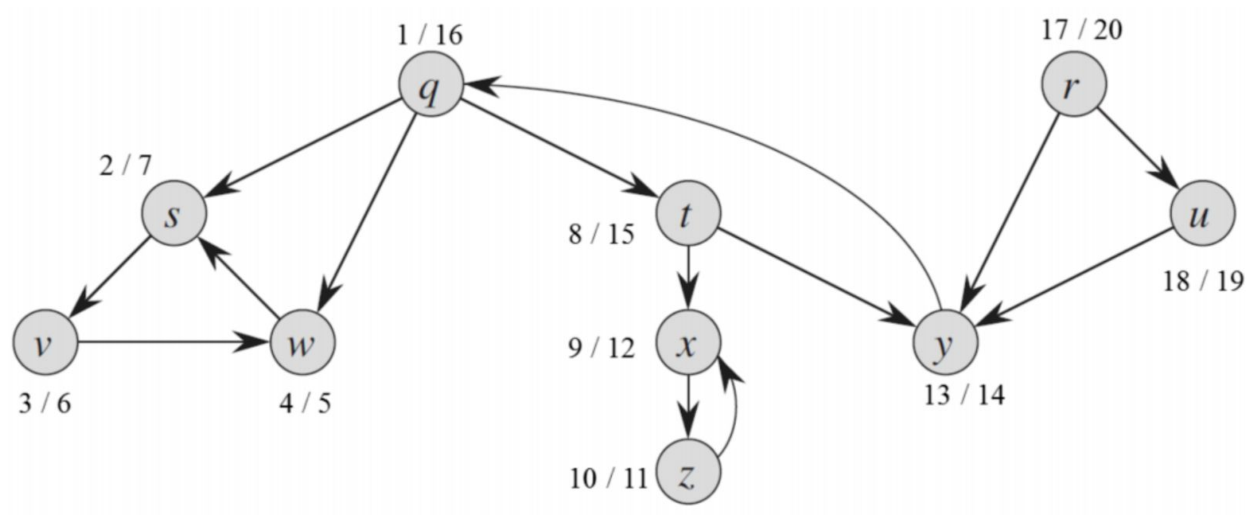
2.

All values are expressed as d, pi Q: nil, nil S: 1, Q W: 1, Q T: 1, Q V: 2, S X: 2, T Y: 2, T Z: 3, X

3.

In the standard form of *BFS* for a graph G = (V, E), total running time is O(V + E). Per the textbook, the operations of enqueue and dequeue take O(1) time, and thus the total time devoted to queue operations is O(V). This does not change if we use adjacency matrices. Also per the book, the procedure scans the adjacency list of each vertex on when the vertex is dequeued, so it scans each adjacency list at most once. In our matrix implementation, we can think of a row of the matrix as an adjacency list of length |V|. For the regular implementation, the sum of the lengths of the adjacency list is Θ (E), so the total time spent in scanning them is O(E). Since our implementation uses "adjacency lists" of length |V|, this means that the combined length of the adjacency lists is $\Theta(V^2)$, so the total time spent in scanning them is $O(V^2)$. Per the book, the overhead for initialization is O(V), so with our matrix implementation the total running time of the BFS procedure is $O(V + V^2)$ which is asymptotically the same as $O(V^2)$.

4.



5

We keep the DFS(G) unchanged, and modify DFS – VISIT(G, u) with a stack to eliminate recursion:

```
DFS-VISIT'(G,u):
     init empty stack S
     S.push(u)
     while S is not empty:
          v = S.pop()
          if v.color == WHITE: // have not checked list
          time = time + 1
          v.d = time
          v.color = GRAY
          S.push(v)
          for each x in reverse_order(G.Adj[v]):
          // without reverse_order it completes a depth-first
          //search but not in the original order
               if x.color == WHITE
                    x.pi = v
                    S.push(x)
          elseif v.color == GRAY: // adjlist already checked
               time = time + 1
               v.f = time
               v.color = BLACK
```

(It's okay to not record time in this problem. But you should remember to change color of node or record visited nodes)

6.

Adjacency list: O(E+V)

Adjacency Matrix: O(V^2)

7.

(Above Solution is by the student Fan Bu)

http://www.algoqueue.com/algoqueue/default/view/1835008/divide-the-elements-of-an-array-into-group-of-3-or-7-instead-of-5-in-deterministic-selection-algorithm

8.

Assume there are n generals in total, and we are looking for the top (0.15n) generals. Since we can only use '<' operator, we can use deterministic select algorithm to find the (0.85n)th generals, and from (0.85n)th to nth general will be the top 15%.

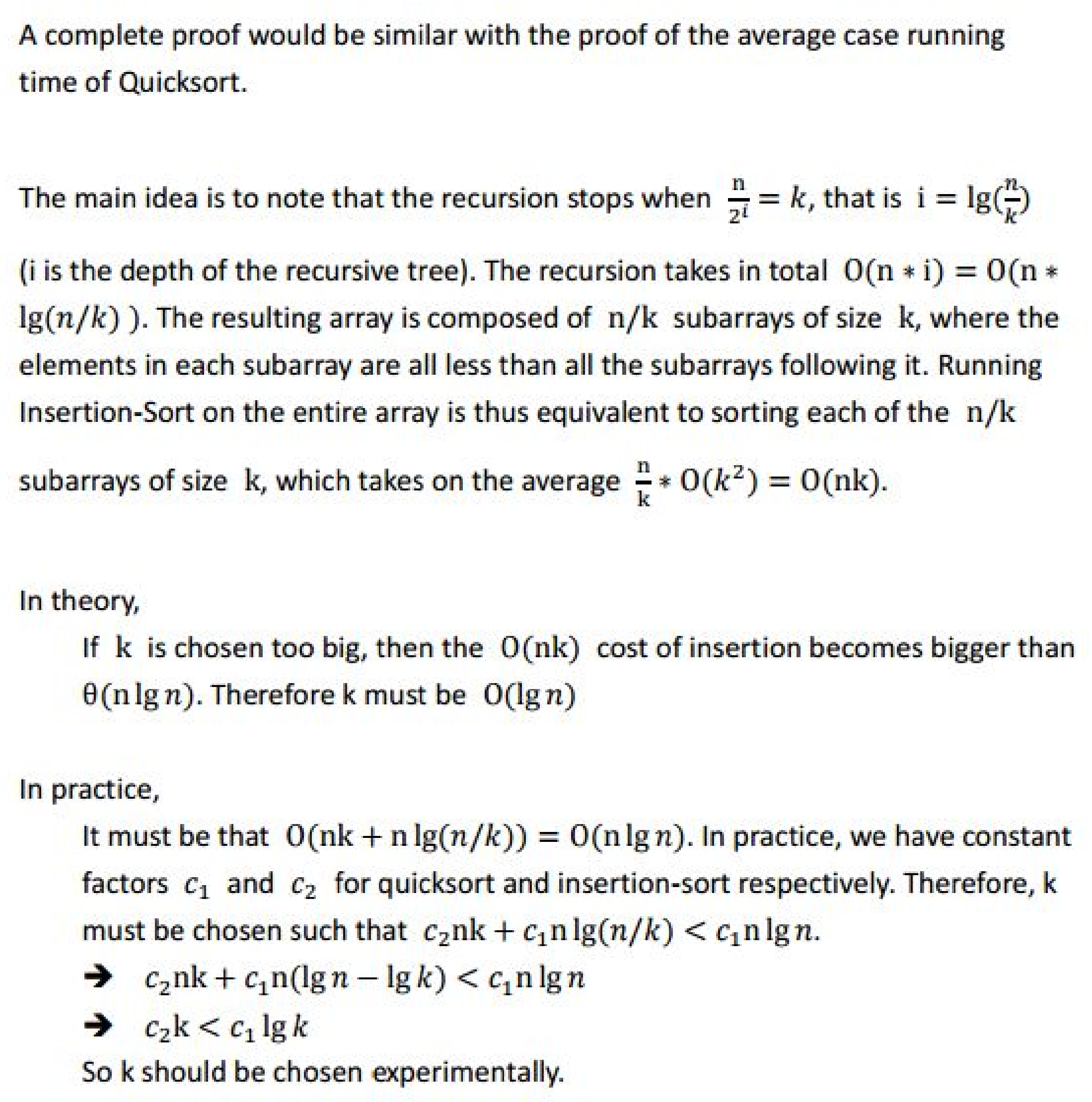
```
FIND-TOP(A,n,i) // Using DETERMINISTIC_SELECT
1. Divide the elements of the input array A into groups of 5.
2. Find the medium of each group of 5 items and put them into another array B
3. x = DETERMINISTIC_SELECT_RANGE(B, n/5, n/10) // finding the medium of the mediums
4. Partition A - {x} into two sets A1, A2 such that A1 = {k | k < x} and A2 = {k | x < k}
5. if i = |A1|+1
6.        add x into A2;
7.        return A.A2 //returns the partition of elements greater than x (including x)
8. otherwise if i < |A1|+1
9.        return DETERMINISTIC_SELECT_RANGE(A1, |A1|, i)
10. else
11. return DETERMINISTIC_SELECT_RANGE(A2, |A2|, i - |A1| - 1)
```

Main function call:

top15 = FIND-TOP(A, n, 0.85n)

The worst case time complexity of this Algorithm is O(n).

9.



A complete proof would be similar with the proof of the average case running time of Quicksort.

The main idea is to note that the recursion stops when $\frac{\mathrm{n}}{2^i} = k$, that is $\mathrm{i} = \lg(\frac{n}{k})$ (i is the depth of the recursive tree). The recursion takes in total $\mathrm{O}(\mathrm{n} * \mathrm{i}) = \mathrm{O}(\mathrm{n} * \lg(n/k)$). The resulting array is composed of $\mathrm{n/k}$ subarrays of size k, where the elements in each subarray are all less than all the subarrays following it. Running Insertion-Sort on the entire array is thus equivalent to sorting each of the $\mathrm{n/k}$ subarrays of size k, which takes on the average $\frac{\mathrm{n}}{\mathrm{k}} * \mathrm{O}(k^2) = \mathrm{O}(\mathrm{nk})$.

In theory,

If k is chosen too big, then the $\mathrm{O}(\mathrm{nk})$ cost of insertion becomes bigger than $\theta(\mathrm{n}\lg n)$. Therefore k must be $\mathrm{O}(\lg n)$

In practice,

It must be that $\mathrm{O}(\mathrm{nk} + \mathrm{n}\lg(n/k)) = \mathrm{O}(\mathrm{n}\lg n)$. In practice, we have constant factors c_1 and c_2 for quicksort and insertion-sort respectively. Therefore, k must be chosen such that $c_2\mathrm{nk} + c_1\mathrm{n}\lg(n/k) < c_1\mathrm{n}\lg n$.

➔ $c_2\mathrm{nk} + c_1\mathrm{n}(\lg n - \lg k) < c_1\mathrm{n}\lg n$

➔ $c_2\mathrm{k} < c_1\lg k$

So k should be chosen experimentally.

10.

This problem can be solved by using BFS or DFS.

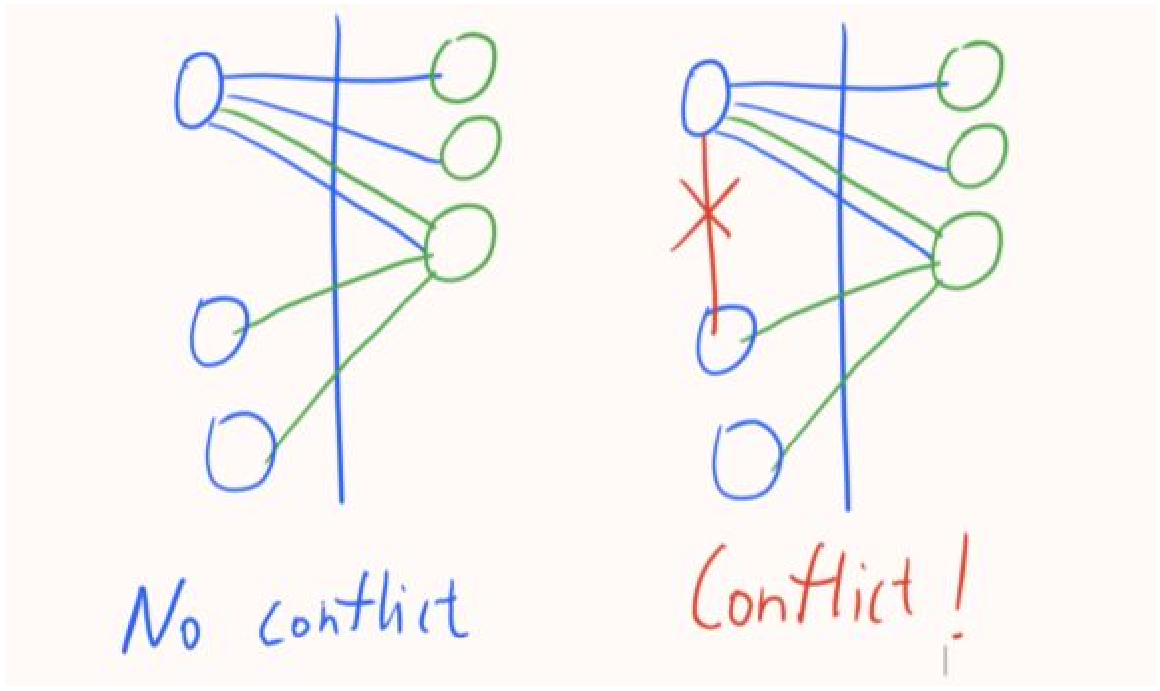
First, let's construct the graph. Every vertex stands for a person and every edge represents a grudge relationship between two people. The grudge relationship can be either one-sided or mutual. In this solution, we will treat it as a mutual relationship and the graph is undirected.

After we built our graph, we can traverse it using BFS. What's new here is that every vertex will have an additional field named color. We dye vertices into two colors (for example, black and white) during the graph traversal. For each edge, we make sure the two vertices on its two sides are colored differently.

If for some edge, we can't meet this requirement, this means that the partition is impossible.

On the other hand, if after we traverse the graph and nothing conflicts, we know there exists an valid assignment. We assign people with the same color to the same room.

Since it's just graph traversing. It takes O(|E|+|V|), which is, from how we construct the graph, O(n+g).



11.

Use quicksort to sort the n music files, if the result is 0, let the computer to calculate again until the result is 1 or -1, in this case, the worst running time is also O(nlogn), because the expectation of comparison time to get one correct answer is 2.