Review Assignment

You should try to solve these problems by yourself. Consult the answer sheet only after you have given it a real attempt. If you do check the answer sheet along the way, set it aside and try to write up a solution without it.

This is not intended to be a comprehensive review. This review was designed based on focus areas from our quizzes, and to get you practice with the newer graph topics, such as Dijkstra's, which have not been fully covered in our homework assignments.

Problem 1: Warm-Up: True or False

(a) _____ Instead of using bucket sort to sort digits in the radix sort algorithm, we can use any valid sorting algorithm and radix sort will still sort correctly.

(b) _____ Consider the forest (vertex-disjoint set of trees) returned by a complete run of DFS on a directed graph. The number of trees in this forest will always be the same, regardless of the order in which you visit nodes in the outermost loop of DFS.

(c) _____ Consider the forest (vertex-disjoint set of trees) returned by a complete run of DFS on an undirected graph. The number of trees in this forest will always be the same, regardless of the order in which you visit nodes in the outermost loop of DFS.

(d) _____ Suppose you have a sorted list of n numbers, to which you add 3 extra numbers in arbitrary places. One can sort this list in $O(n)$ time.

(e) _____ Every directed acyclic graph has exactly one topological ordering.

(f) _____ If a directed graph G is cyclic but can be made acyclic by removing one edge, then a depth-first search in G will encounter exactly one back edge.

(g) _____ In every directed acyclic graph, BFS and DFS visit the vertices in the same order.

(h) _____ If a topological sort exists for the vertices in a directed graph, then a DFS on the graph will produce no back edges.

(i) _____ You can find the shortest path between two vertices in a directed graph, where the weights of all edges are equal in $O(V + E)$ time.

Problem 2: Loop Analysis

For each of the following program fragments, compute the worst-case asymptotic time complexity (as a function of n). Whenever it says "loop body" you can assume that a constant number of lines of code are there. Remember, the best way to reason about loops is to consider how many times the internal statement will be executed.

(a)

```
for(i = 1; i <= n^2; i++){
   for(j = 1; j <= 2n - 1; j++){
      //loop body
   }
}
```

(b)

```
for(i = 1; i <= n; i++){
   //loop body
}
i = 1
while(i < n){
   //loop body
   i = i * 2
}
```

(c)

```
i = 1
while(i < n){
   //loop body
   i = i + 5
}
```

(d)

```
i = 1
while(i < n){
   //loop body
   i = i + n
}
```

Problem 3: Recurrences, recurrences, recurrences

In this problem, you are to analyze the asymptotic time complexity $T(n)$ for the following divide and conquer algorithms, solve $T(n)$ using the master method.

(a)

```
bar(A, i, j) {
   if A[i] > A[j] { exchange(A[i], A[j]) }
   if ((i + 1) >= j) { return }
   k = (j - i + 1)/3
   bar(A, i, j - k)
   bar(A, i+k, j)
   bar(A, i, j - k)
}
```

(b)

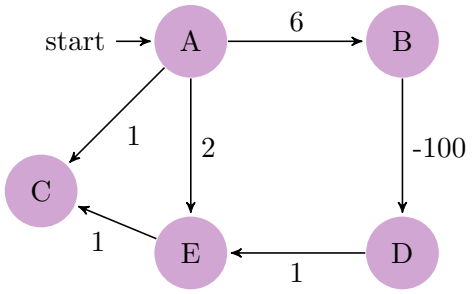
```
int [] foo(A){
   n =  A.length
   if(n = 1) { return A[0] }
   int half = n/2
   int A1 = new int[half]
   int A2 = new int[n-half]
   for(int i = 0; i < half - 1; i++){
      for(int j = 0; j < i; j++){
          if (j = 0) { A1[i] = A[2i] }
         else { A2[i] = A[2i + 1]}
       }
   }
   A2[n-half-1] = A[n-1]
   b1 = foo(A1)
   b2 = foo(A2)
   if (b1 > b2) { return b1 }
   else return b2
}
```

(c)

```
int [] bar(A){
   n =  A.length
   if(n = 1) { return A[0] }
   int half = n/2
   int A1,A2,A3,A4 = new int[n/2]
   for(int i = 0; i < n/2; i++){
      for(int j = 0; j < n/2; j++){
          A1[i] = A[i]
          A2[i] = A[i+j]
          A3[i] = A[n/2 + j]
          A4[i] = A[j]
       }
   }
   b1 = bar(A1)
   b2 = bar(A2)
   b3 = bar(A3)
   b4 = bar(A4)
   return b1*b2*b3*b4
}
```

Problem 4: Getting to know Dijkstra's

(a) Dijkstra's algorithm may fail when a graph has negative weight edges. Consider the following graph:



Fill in the corresponding table following the steps of Dijkstra's algorithm and pinpoint where the algorithm fails. In the table, S s the set of vertices that are "done" and Q is $V - S$ (the set of vertices you have to choose from for the next iteration).

	S	Q	$d[A]$	$d[B]$	$d[C]$	$d[D]$	$d[E]$
Initialize	-	$\{A, B, C, D, E\}$	0	∞	∞	∞	∞
Step 1							
Step 2							
Step 3							
Step 4							
Step 5							

(b) The following is the proof of correctness for Dijkstra's algorithm:
Claim: Dijkstra's algorithm terminates with $u.d = \delta(s, u)$ for all $u \in V$.
Base case: $|S| = 1$ is trivial.
Inductive hypothesis: Assume true for $|S| = k \geq 1$

- Let v be the next node added to S and let u-v be the chosen edge.
- The shortest s-u path plus (u, v) is an s-v path of length $\delta(v)$.
- Consider any s-v path P. We'll see that it is no shorter than $\delta(v)$.
- Let (x, y) be the first edge in P that leaves S, and let P' be the subpath s-x.
- P is already too long as soon as it leaves S:

$$w(P) \geq w(P') + w(x, y) \geq d(x) + w(x, y) \geq \delta(y) \geq \delta(v)$$

- The last step above is because the loop chose v instead of y as the next vertex to include.

Why does this proof fail when the graph has negative weight edges?

(c) Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, but all other edge weights are non-negative and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds the shortest paths from s in this special case.

(d) Consider a weighted directed graph $G = (V, E, w)$ and let X be a shortest s-t path for $s, t \epsilon V$. If we double the weight of every edge in the graph, setting $w'(e) = 2w(e)$ for each $e \epsilon E$, then X will still be a shortest s-t path in (V, E, w').

Problem 5: Graph Extravaganza

(a) K-colorable Graphs

An undirected graph $G = (V, E)$ is said to be k-colorable if all of the vertices of G can be colored one of k different colors such that no two adjacent vertices are assigned the same color. Design an algorithm that either colors a graph with 2 colors or determines that two colors are not sufficient. Argue that your algorithm is correct.

(b) Coloring with Dijkstras

Each edge in a connected, unweighted graph G is colored either red or blue. Present an algorithm to compute a path between s and t that traverses the fewest number of red edges. Analyze its running time.

(c) Shortest Path in Weighted Graph

Suppose you want to get from s to t on a weighted graph G with non-negative edge weights, but you would like to stop by u if it isn't too inconvenient. (Here too inconvenient means that it increases the length of your travel by more than 10%.)

Describe an efficient algorithm that would determine an optimal s to t path given your preference for stopping at u along the way if not too inconvenient. It should either return the shortest path from s to t, or the shortest path from s to t containing u.