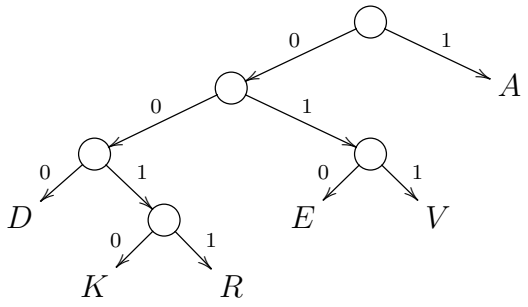
Data Structures
CSCI C343, Fall 2017

Final Exam [A]

Name: _____

This exam has 12 questions, for a total of 100 points.

1. [6 points] Given the following tree for a Huffman code



decode the following string:

$$101110001001001000010110011 1$$

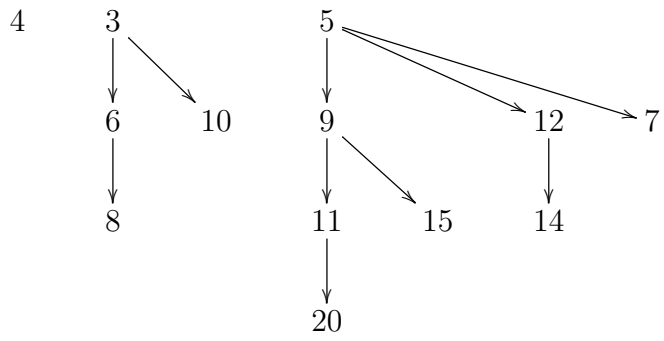
Solution: (0.5 point per correct letter)

AVADAKEDAVRA

2. [8 points] Draw a Binomial Priority Queue containing the following integers, where the smaller integers are given priority over larger ones (i.e., the binomial trees are min heaps).

$$\{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 20\}$$

Solution: Here's one possible solution.



- Does the binomial queue contain all the elements? (1 point)

- Does the result satisfy the min heap property? (2 points)
- Is each tree a valid binomial tree? (3 point)
- Are the trees all of different sizes? (2 points)

3. 8 points Apply the Partition algorithm to the following array, ensuring that all elements less or equal to the pivot element are in lower positions and all elements greater than the pivot are in greater positions. The pivot element starts out as the last element of the array. Write down the array after each step (each iteration of the loop), drawing two vertical lines to separate the three partitions (the less-than or equal region, the greater-than region, and the to-do region).

$$[4, 8, 7, 2, 9, 5]$$

Solution:

$[||\ 4, 8, 7, 2, 9 \mid 5]$
$[4 \mid\mid 8, 7, 2, 9 \mid 5]$ (1 point)
$[4 \mid 8 \mid 7, 2, 9 \mid 5]$ (1 point)
$[4 \mid 8, 7 \mid 2, 9 \mid 5]$ (1 point)
$[4, 2 \mid 7, 8 \mid 9 \mid 5]$ (2 points)
$[4, 2 \mid 7, 8, 9 \mid\mid 5]$ (1 point)
$[4, 2 \mid 5 \mid 8, 9, 7]$ (2 points)

4. 12 points Fill in the blanks to complete the following implementation of the `fillCache` method that applies dynamic programming to complete the best alignment between two DNA sequences.

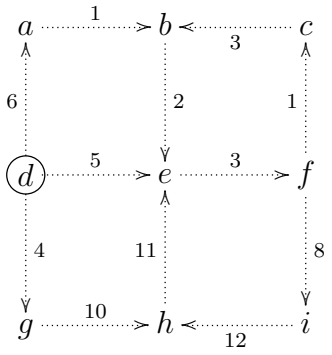
```
private void fillCache() {
  String x = " " + this.x, y = " " + this.y;
  int n = x.length(), m = y.length();
  cache[0][0] = new Result(___(a)___, Direction.NONE);
  for (int col = 1; col < m; col++)
    cache[0][col] = new Result(___(b)___, Direction.LEFT);
  for (int row = 1; row < n; row++)
    cache[row][0] = new Result(judge.getGapCost() * row, Direction.UP);
  for (int row = 1; row < n; row++) {
    for (int col = 1; col < m; col++) {
      char nextX = x.charAt(row), nextY = y.charAt(col);
      int diag = cache[row - 1][col - 1].getScore() + ___(c)___;
      int left = cache[row][col - 1].getScore() + judge.score(Constants.GAP_CHAR, nextY);
      int up = ___(d)___.getScore() + judge.score(nextX, Constants.GAP_CHAR);
      Result best = new Result(diag, Direction.DIAGONAL);
      if (left > best.getScore())
        best = new Result(left, ___(e)___);
      if (up > best.getScore())
        best = new Result(up, Direction.UP);
```

```
            cache[row][col] = ___(f)___;
        }
    }
}
```

Solution: (2 points each)

(a) 0
(b) judge.getGapCost() $*$ col
(c) judge.score(nextX, nextY)
(d) cache[row $-$ 1][col]
(e) Direction.LEFT
(f) best

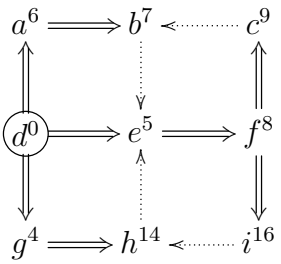
5. [10 points] Apply Dijkstra's shortest paths algorithm to the following graph, starting at vertex d. Show your work by recording what the priority queue looks like and which vertex gets popped at each iteration of the algorithm. Indicate the resulting shortest paths tree by making the tree edges into solid lines. Next to each vertex, write the distance of the shortest path from d to that vertex.



Solution: The priority queue at each step: (3 points)

$$\begin{aligned}
\{g:4, a:6, e:5\} &\quad \text{pop } g\\
\{e:5, a:6, h:14\} &\quad \text{pop } e\\
\{a:6, f:8, h:14\} &\quad \text{pop } a\\
\{b:7, f:8, h:14\} &\quad \text{pop } b\\
\{f:8, h:14\} &\quad \text{pop } f\\
\{c:9, h:14, i:16\} &\quad \text{pop } c\\
\{h:14, i:16\} &\quad \text{pop } h\\
\{i:16\} &\quad \text{pop } i
\end{aligned}$$

The shortest paths tree and shortest distances from i to all the other vertices:
(4 points for the correct tree)
(3 points for the correct distances)



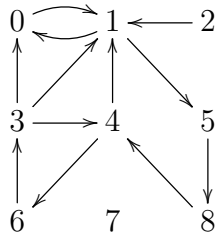
6. [12 points] Implement a dynamic programming (aka. bottom-up) solution to the rod-cutting problem. You are given a rod of length n and an array of prices P that maps rod-lengths to dollars. Return the maximum amount of money that you can obtain by cutting the rod into pieces and selling those pieces.

```
class RodCutting {
  static int max_from_rod_cutting(int[] P, int n) {
```

Solution:

```
static int max_from_rod_cutting(int[] P, int n) {
    int[] R = new int[n+1];
    R[0] = P[0];                          // (1 point) R[0] = 0; ok too
    for (int j = 1; j != n+1; ++j) {      // (1 point)
        int best = 0;                     // (1 point)
        for (int i = 1; i != j+1; ++i) {  // (1 point)
            int sales = P[i] + R[j - i];  // (3 points)
            if (best < sales)             // (2 point)
                best = sales;
        }
        R[j] = best;                      // (1 point)
    }
    return R[n];                          // (2 points)
}
```

7. [6 points] Draw the adjacency list representation of the following directed graph.



Solution: (0.5 point per correct edge)

0 → (1)

1 → (0) → (5)

2 → (1)

3 → (0) → (1) → (4)

4 → (1) → (6)

5 → (8)

6 → (3)

7

8 → (4)

8. [10 points] Fill in the blanks to complete the following implementation of Counting Sort. The elements of `A` are integers in the range of 0 to k.

```
static void counting_sort(int[] A, int[] B, int k) {
  int[] C = new int[k+1]; // The count for each element of A
  int[] L = new int[k+1];  // L[j] == number of elements less or equal j.
  for (int i = 0; i != A.length; ++i) {
    ++C[___(a)___];
  }
  L[0] = C[0];
  for (int j = 1; j != ___(b)___; ++j) {
    L[j] = ___(c)___ + L[j−1];
  }
  for (int j = ___(d)___; j != −1; −−j) {
    int elt = A[j];
    int num_le = L[elt];
    B[___(e)___] = elt;
    L[elt] = num_le − 1;
```

```
    }
  }
```

Solution: (2 points each)

(a) A[i]
(b) k + 1 or L.length
(c) C[j]
(d) A.length − 1
(e) num_le − 1

9. [6 points] What is the big-O time complexity of the Breadth-First Search algorithm shown below given a graph G with n vertices and m edges? Give a detailed argument in support of your answer and give the tightests (lowest) bound that you can. Note that for certainly applications, m may be much less than n^2, which is why m is a separate parameter to the time complexity.

```
static <V> void bfs(Graph<V> G, V start, Map<V,Boolean> visited, Map<V,V> parent){
    for (V v : G.vertices())
        visited.put(v, false);
    Queue<V> Q = new LinkedList<V>();
    Q.add(start);
    parent.put(start, start);
    visited.put(start, true);
    while (! Q.isEmpty()) {
        V u = Q.remove();
        for (V v : G.adjacent(u))
            if (! visited.get(v)) {
                parent.put(v, u);
                Q.add(v);
                visited.put(v, true);
            }
    }
}
```

Solution: The total time complexity is $O(m + n)$. The first **for** loop is $O(n)$ (1 point). Regarding the **while** loop, here are two analyses, with the second being more precise and worth more points.

- 2 points: The **while** loop does at most n iterations because each vertex in the graph may be inserted and then removed just once from the queue (1 point). The inner **for** loops does at most n iterations each time, for which one could deduce the bound of $O(n^2)$ for the entire **while** loop (1 point).
- 3 points: Each edge in the graph is only considered once by the combination of the **while** and nested **for** loop, so the time complexity is $O(m)$.

Thus, combining the cost of the first **for** loop and the **while** loop, we have $O(n + m)$ (2 points).

10. [10 points] Five new islands have appeared in the Florida Keys due a mysterious drop in ocean levels and the government wants to build bridges to connect them so that each island can be reached from any other one via one or more bridges. The cost of constructing a bridge is proportional to its length. The distances in miles between pairs of islands are given in the following table, in which the islands are named A through E. State which bridges to build so that the total construction cost is minimized. Additionally, state the total cost.

	A	B	C	D	E
A	-	25	21	35	28
B	-	-	26	15	22
C	-	-	-	27	8
D	-	-	-	-	16
E	-	-	-	-	-

Solution: This solution applies Kruskal's algorithm to compute the minimum spanning tree (MST). (3 points)

The first step is to order the edges by weight: (2 points)

$$(8)C-E, (16)D-E, (15)B-D, (21)A-C, (22)B-E, (25)A-B, (26)B-C,$$
$$(27)C-D, (28)A-E, (35)A-D$$

Next we process each edge, deciding to make it part of the MST or not, depending on whether the endpoints of each edge are in different disjoint sets.

$$C-E(yes), D-E(yes), B-D(yes), A-C(yes), B-E(no), A-B(no), B-C(no),$$
$$C-D(no), A-E(no), A-D(no)$$

So we build the bridges as indicated by the "yes" answers above. (3 points)
The total cost is 60. (2 points)

11. [9 points] Change the following implementation of the disjoint sets data structure to implement path compression and union by rank.

```
public class UnionFind<N> {
    Map<N,N> parent;

    public UnionFind() {
        parent = new HashMap<>();
    }

    public void make_set(N x) {
        parent.put(x, x);
    }

    public N find(N x) {
        if (x == parent.get(x))
            return x;
        else {
            return find(parent.get(x));
        }
    }

    public N union(N x, N y) {
        N rx = find(x);
        N ry = find(y);
        parent.put(ry, rx);
        return rx;
    }
}
```

Solution:

```
public class UnionFind<N> implements DisjointSets<N> {
    Map<N,N> parent;
    Map<N,Integer> rank;                        // (1 point)

    public UnionFind() {
        parent = new HashMap<>();
        rank = new HashMap<>();                 // (1 point)
    }

    public void make_set(N x) {
        parent.put(x, x);
        rank.put(x, 0);                         // (1 point)
    }

    public N find(N x) {
```

```
        if (x == parent.get(x))
            return x;
        else {
            N rep = find(parent.get(x));
            parent.put(x, rep);                    // (3 points)
            return rep;
        }
    }

    public N union(N x, N y) {
        N rx = find(x);
        N ry = find(y);
        if (rank.get(rx) > rank.get(ry)) {         // (2 points)
            parent.put(ry, rx);
            return rx;
        } else {
            parent.put(rx, ry);
            if (rank.get(ry) == rank.get(rx))
                rank.put(ry, rank.get(ry) + 1);    // (1 point)
            return ry;
        }
    }
}
```

12. [3 points] What advice would you give a student taking C343 Data Structures next year?

Solution: Open ended.