1. [3 points] Fill in the blanks in the following to fulfill the comments.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class Args {
    /** A pattern that matches a time containing minutes and hours on a 24-hour
     *  clock.  Minutes must be two-digit decimal numerals 00 through
     *  59, and hours must consist of one or two digits (0 through
     *  23). There must be two captured groups: one for hours and one
     *  for minutes.  Time 0:00 (or 00:00) denotes midnight.  Times such as
     *  24:00, 25:01, 12:60, or 002:22 are not valid.
     */
    static final Pattern PATN =

        Pattern.compile("________________________________________________");

    /** Prints the number of minutes after midnight denoted by TEXT, a
     *  time in the format described by PATN.
     */
    public static void printMinutes(String text) {
        Matcher matcher = PATN.matcher(text);
        matcher.matches();

        int hours = ______________________________________;

        int minutes = ______________________________________;

        System.out.println(______________________________);
    }

}
```

2. [3 points]

In the following, assume in each case that `y` ***is a non-negative Java*** `int` ***variable.***

a. What can you say about `y` if the following expression is true?

```
y == (((y & 0xaaaaaaaa) >> 1) | (y & 0xaaaaaaaa))
```

(Choose one)

◯ `y == 0`.
◯ `y` is odd.
◯ `y` is divisible by 5.
◯ `y` is divisible by 3.
◯ `y` is divisible by 10.
◯ None of the above.
◯ All of the above.

b. Does the answer to (a) change if we change `>>` to `>>>`?
◯ Yes. ◯ No.

c. For what ranges of `y` is the following true?

```
y * 144 == ((y<<4) ^ (y<<7))
```

(Pick the largest range that applies.)

◯ `0 <= y < 16`.
◯ `0 <= y <= 16`.
◯ `0 <= y <= 8`.
◯ `0 <= y <= 4`.
◯ All non-negative integers less than 2^{24}.

3. [4 points] Fill in the skeleton below to fulfill its comment. You may use any of the classes in package `java.util.*`. Do not introduce any arrays. Your solution must take $\Theta(N)$ time, where N is the length of the input list. You may make no assumptions about the dynamic type of the input list, and therefore ***you* cannot assume that `.get` runs in constant time, and will need an iterator.** You need not use all lines of the skeleton.

```
import java.util.*;
class tri {
    /** Non-destructively return the list of lists of values
     *  [ [ a[0] ], [ a[1], a[2] ], [ a[3], a[4], a[5] ], ...],
     *  where a[i] denotes the value at index i in A. Each list gets
     *  longer by one, except that the last should only be as long
     *  as needed to hold the remaining values of A.
     *  For example, given the input [ 1, 2, 3, 4, 5, 6, 7, 8 ], the
     *  result would be [ [1], [2, 3], [4, 5, 6], [7, 8] ]. */
    static List<List<Integer>> triangle(List<Integer> A) {

        ______________________ result = new  _____________________;

        ____________________________________________________________

        ____________________________________________________________

        for (int k = 1; _____________________________; k += 1) {

            ____________________________________________________________

            ____________________________________________________________

            ____________________________________________________________

            ____________________________________________________________

            ____________________________________________________________

            ____________________________________________________________
        }
        return result;
    }
}
```

4. [8 points] Fill in the appropriate bubbles for the following.

a. Which of the following can be caused by providing a heuristic for A* search that underestimates the distance to the target node? Mark all that apply.

- ☐ Cause the search to fail to reach the target.
- ☐ Cause the search to find a path to the target that is not the shortest.
- ☐ Cause the search to consider more nodes than necessary.
- ☐ Force the search into an infinite loop.
- ☐ None of the above.

b. Which of the following can be caused by providing a heuristic for A* search that overestimates the distance to the target node? Mark all that apply.

- ☐ Cause the search to fail to reach the target.
- ☐ Cause the search to find a path to the target that is not the shortest.
- ☐ Cause the search to consider more nodes than necessary.
- ☐ Force the search into an infinite loop.
- ☐ None of the above.

c. Consider a game-tree search that starts from a position where the maximizing player is to move and searches down $2N$ moves (N for each player, so that it is again the maximizing player's turn in the bottom positions). Suppose that during the search, the alpha-beta criteria causes the maximizing player to prune a certain branch. If we repeat the search, going two moves deeper (one for each side, for a total $2N+2$), must that same branch be pruned? Choose one answer.

- ○ Yes; extending the search will yield results consistent with the first search.
- ○ Yes, as long as the deeper search does not discover a win at level $2N+2$.
- ○ No; a position at level $2N$ (where the search stopped before) may have increased its value.
- ○ No; a position at level $2N$ may have decreased its value.
- ○ No, unless the deeper search discovers a win at level $2N+2$.

d. Consider a complete binary max-heap with $2^n - 1 > 16$ values, all distinct, stored in the usual way in an array, `H` in positions `H[1 .. `2^n-1`]`. Where can the third-smallest value be? Choose the best answer.

- ○ Anywhere in `H` except `H[1]`.
- ○ Anywhere in `H[`2^{n-1} .. 2^n-1`]`.
- ○ Anywhere in `H[`2^{n-2} .. 2^n-1`]`.
- ○ Anywhere in `H[`2^{n-3} .. 2^n-1`]`.
- ○ Anywhere in `H[`2^{n-1} .. $2^{n-1}+2$`]`.
- ○ Anywhere in `H[`2^{n-2} .. $2^{n-1}-1$`]`.
- ○ Anywhere in `H[`2^{n-3} .. $2^{n-2}-1$`]`.
- ○ Anywhere in `H[`2^{n-3} .. $2^{n-1}-1$`]`.

e. A student creates a hashing function $h(x)$ for integers in an externally chained table with 1000000 buckets that returns `(int) (1000000 * y)`, where `y` is the fifth value generated by the `.nextDouble` method on a `Random` object that has been seeded with the value `x`. Is this likely to work reasonably well? Choose the best answer.

◯ Yes, because all the values of h will be distinct for distinct inputs.
◯ Yes, with relatively high probability, distinct values of x will hash to different values.
◯ No, because a hashing function must not be non-deterministic.
◯ No, because for some values of x, the random-number generator will have a short period.
◯ No, because of patterns in the numbers generated by the generator.

f. If we apply LZW encoding to encode the string `"aaaa..."`, where 'a' is repeated M times, how many codewords will be in the output?

◯ About $M/2$. ◯ About $\sqrt{M}$. ◯ About M. ◯ About $\lg M$.

g. Consider implementing a deque (double-ended queue) with a linked list versus using the best representation you can think of in an array. Which of the following are true for an arbitrary, sufficiently long mix of N operations (insertions in front, insertions at the end, deletions from the front, deletions from the end) when comparing times from the linked-list representation to the times from the array representation? Choose all that apply.

☐ The times required for executing the entire sequence will be within a constant factor of each other.
☐ The times spent performing the first $M < N$ operations will be within a constant factor of each other.
☐ The times spent on any given insertion will be within a constant factor of each other.
☐ The times spent on any given deletion will be within a constant factor of each other.
☐ The time spent on a particular deletion may be much larger for the array representation because of having to move the remaining elements.

5. [6 points] In the following questions, notations such as $A \subseteq B$ mean that every function in the set of functions A is also in the set of functions B. Likewise, $A = B$ means that A and B are the same set of functions, and $A \cup B$ is the union of the sets of functions A and B. Fill in the appropriate bubbles.

a. True or false: $\Theta(f(N)) \subseteq O(f(N))$.

◯ True ◯ False

b. True or false: $\Omega(f(N)) \cup O(f(N))$ includes all functions of N.

◯ True ◯ False

c. True or false: $1/2^N \in \Theta(1)$.

◯ True ◯ False

d. What is the worst-case running time of the call `funcD(N, true)` as a function of `N`? Assume `p` takes constant time.

```
public void funcD(int n, boolean which) {
    if (n < 1) {
        doSomething();
    } else if (which && p(n)) {
        funcD(n / 2, true);
        funcD(n, false);
    } else {
        funcD(n - 1, false);
    }
}
```

◯ $\Theta(1)$ ◯ $\Theta(\lg N)$ ◯ $\Theta(N)$ ◯ $\Theta(N \lg N)$ ◯ $\Theta(N^2)$ ◯ $\Theta(N^3)$ ◯ $\Theta(2^N)$

e. What is the worst-case running time of the call `funcE(N)` as a function of N? Assume that the function `h` takes constant time.

```
public void funcE(int n) {
    for (int i = n; i > 0; i /= 2) {
        for (int j = i; j > 0; j -= 1) {
            h(i, j, n);
        }
    }
}
```

○ $\Theta(1)$ ○ $\Theta(\lg N)$ ○ $\Theta(N)$ ○ $\Theta(N \lg N)$ ○ $\Theta(N^2)$ ○ $\Theta(N^3)$ ○ $\Theta(2^N)$

f. What is the worst-case running time of the call `funcF(N, M)` as a function of N? Assume calls to `h` take constant time and that $M > 1$ is a constant.

```
public void funcF(int n, int b) {
    if (n <= 1) {
       h();
    } else {
       for (int i = 0; i < b; i += 1) {
           funcF(n / b, b);
       }
    }
}
```

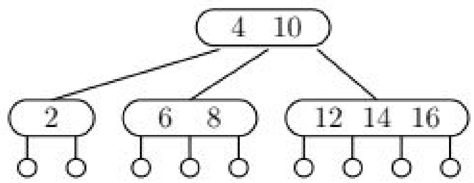
○ $\Theta(1)$ ○ $\Theta(\lg N)$ ○ $\Theta(N)$ ○ $\Theta(N \lg N)$ ○ $\Theta(N^2)$ ○ $\Theta(N^3)$ ○ $\Theta(2^N)$

6. [1 point] What commercial product resulted from an accident with a batch of photo-sensitive glass?

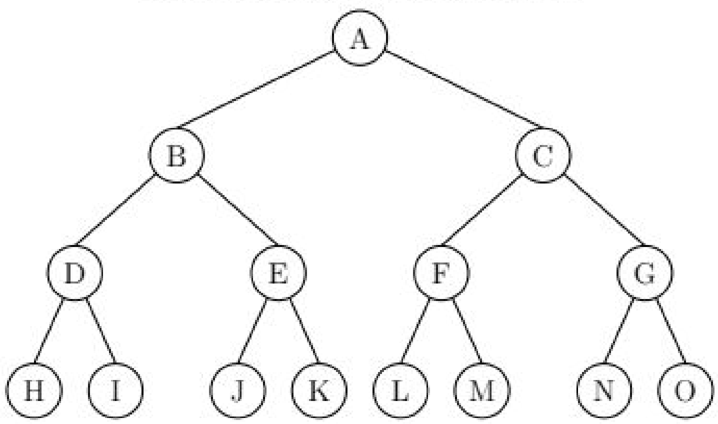
7. [5 points]

a. Fill in the unique left-leaning red-black tree corresponding to the (2, 4) tree below. For each lettered node, indicate the key it contains and indicate whether it is red. Not all lettered nodes are actually needed. Leaving one blank indicates that it is actually an empty (null) node.

(2, 4) Tree



Left-Leaning Red-Black Tree



A.	______	◯ Red?	B.	______	◯ Red?	C.	______	◯ Red?
D.	______	◯ Red?	E.	______	◯ Red?	F.	______	◯ Red?
G.	______	◯ Red?	H.	______	◯ Red?	I.	______	◯ Red?
J.	______	◯ Red?	K.	______	◯ Red?	L.	______	◯ Red?
M.	______	◯ Red?	N.	______	◯ Red?	O.	______	◯ Red?

More parts on the following page.

b. Consider a new kind a balanced tree: a red-green-black tree. This is a binary search tree whose nodes can be colored either black, red, or green, according to the following constraints, of which (a), (b), and (c) are the same as for red-black trees:

   a. The root is black, and empty nodes are considered black as well.
   b. The number of black nodes in any path from the root to the leaves is the same.
   c. A black node may have only black and red children.
   d. A red node may have only black and green children.
   e. A green node may have only black children.
   f. Except for null nodes and possibly the root, all black nodes must have two red children.

   Like red-black trees, these trees correspond to (i.e., can represent) a kind of B-tree. Which kind?

   ◯ (2, 5) trees ◯ (3, 6) trees ◯ (3, 7) trees ◯ (4,8) trees.

c. Assuming a search tree meets the constraints in part (b), will it be sufficiently balanced that searching for a value will require $O(\lg N)$ time for a tree with N keys?

   ◯ True ◯ False

   Why or why not?

8. [6 points] Below you will find some intermediate steps in performing various sorting algorithms on the same input list. The steps do not necessarily represent consecutive steps in the algorithm (that is, many steps are missing), but they are in the correct sequence. For each of them, select the algorithm (by filling in the appropriate bubble) from among the following choices: insertion sort, straight selection sort, mergesort, quicksort, heapsort, LSD radix and MSD radix sort. For quicksort, the pivot is the median value of the first, last, and middle element (specifically, the index of the middle element is the average of the indices of the left and right elements, rounded down). Algorithms may appear twice.

In all these cases, the final step of the algorithm will be the sorted sequence:

```
 43,  62, 191, 193, 265, 414, 482, 523, 592, 615, 674, 759, 834, 894, 920
```

but it might not be shown.

Input List:

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
```

a. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
 43,  62, 193, 191, 265, 482, 414, 523, 592, 674, 615, 759, 834, 894, 920
 43,  62, 191, 193, 265, 482, 414, 523, 592, 674, 615, 759, 834, 894, 920
 43,  62, 191, 193, 265, 414, 482, 523, 592, 615, 674, 759, 834, 894, 920
```

b. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
 62, 193, 523, 482, 191,  43, 265, 414, 592, 834, 920, 674, 759, 894, 615
 62, 193, 523, 482, 191,  43, 265, 414, 592, 615, 674, 834, 759, 894, 920
 62, 193, 523, 482, 191,  43, 265, 414, 592, 615, 674, 759, 834, 894, 920
 62, 193, 523, 482, 191,  43, 265, 414, 592, 615, 674, 759, 834, 894, 920
 62, 193, 191,  43, 265, 414, 523, 482, 592, 615, 674, 759, 834, 894, 920
```

c. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
 62, 674, 894, 615, 482, 759, 834, 193, 592,  43, 265, 414, 523, 191, 920
 62, 674, 834, 615, 482, 759, 191, 193, 592,  43, 265, 414, 523, 894, 920
 62, 674, 759, 615, 482, 523, 191, 193, 592,  43, 265, 414, 834, 894, 920
 62, 674, 523, 615, 482, 414, 191, 193, 592,  43, 265, 759, 834, 894, 920
265, 615, 523, 592, 482, 414, 191, 193,  62,  43, 674, 759, 834, 894, 920
```

Continued on next page.

d. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 674, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 482, 523, 674, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 482, 523, 674, 834, 920, 191, 592, 615,  43, 265, 414, 759, 894,  62
191, 193, 482, 523, 674, 834, 920, 592, 615,  43, 265, 414, 759, 894,  62
 43, 191, 193, 265, 414, 482, 523, 592, 615, 674, 834, 920, 759, 894,  62
```

e. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

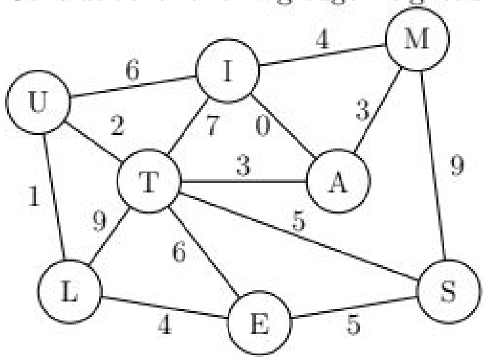
```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
920, 191, 482, 592,  62, 193, 523,  43, 674, 834, 414, 894, 615, 265, 759
414, 615, 920, 523, 834,  43, 759,  62, 265, 674, 482, 191, 592, 193, 894
 43,  62, 191, 193, 265, 414, 482, 523, 592, 615, 674, 759, 834, 894, 920
```

f. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
674, 193, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 674, 523, 482, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 674, 482, 523, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 482, 523, 674, 920, 834, 191, 592, 615,  43, 265, 414, 759, 894,  62
193, 482, 523, 674, 191, 592, 834, 920, 615,  43, 265, 414, 759, 894,  62
191, 193, 482, 523, 592, 674, 834, 920, 615,  43, 265, 414, 759, 894,  62
```

9. [5 points]

Consider the following edge-weighted undirected graph:



In the following questions, break ties in the algorithms in favor of vertices that come first in alphabetical order, or edges that are listed first.

a. Run Prim's algorithm starting from node L. Indicate the vertices that are in the fringe in order of decreasing priority just after node T is removed from the fringe and processed. Break ties in priority by having the node that is first alphabetically have highest priority.

○ A, E, M, I
○ A, E, S, I
○ A, E, M, S, I
○ A, E, S
○ A, I, S, M

b. What are the edges in the shortest-path tree from L? Mark all that apply.

☐ (A, I) ☐ (A, M) ☐ (A, T) ☐ (E, L) ☐ (E, S) ☐ (E, T)
☐ (I, M) ☐ (I, U) ☐ (I, T) ☐ (L, T) ☐ (L, U) ☐ (M, S)
☐ (S, T) ☐ (T, U)

c. Dijkstra's algorithm assumes that all edge weights are non-negative. To illustrate why this is so, suppose we run Dijkstra's algorithm from node L, and terminate the algorithm as soon as it removes node I from the fringe. Which edge's weight could we change to -6 to produce an ***incorrect*** shortest path to I? Assume that nodes are never updated after being removed from the fringe. Choose one.

○ (A, I) ○ (A, M) ○ (A, T) ○ (E, L) ○ (I, U) ○ (I, T)
○ (L, T) ○ (M, S) ○ (S, T)

d. Suppose that when running Kruskal's algorithm on the graph above, we fail to check whether an edge joins a subtree to itself. What is the first edge to be added to the resulting tree erroneously? Choose one.

○ (A, I) ○ (A, M) ○ (A, T) ○ (E, L) ○ (E, S) ○ (E, T)
○ (I, M) ○ (I, U) ○ (I, T) ○ (L, T) ○ (L, U) ○ (M, S)
○ (S, T) ○ (T, U)

[redacted]

10. [6 points] You are given the following interface to a general directed graph. The vertices of these graphs are simply positive integers in some contiguous range starting at 1.

```
public interface DiGraph {
    /** Returns the number of vertices in this DiGraph. The vertices
     *  are represented by integers 1..size(). */
    int size();

    /** Returns a list of the successor vertices of V in this
     *  graph. */
    List<Integer> successors(int v);

    /** Returns a list of the predecessor vertices of V in this
     *  graph. */
    List<Integer> predecessors(int v);

    /** Returns a view of this graph in which all edges are
     *  reversed.  Changes to this graph will be reflected in the
     *  result and vice-versa. */
    public DiGraph reverse();
}
```

a. Implement the class `Traverser`. The public `traverse` method of `Traverser` performs a depth-first traversal of a `DiGraph` by performing a depth-first traversal starting from each vertex in turn, in order, and traversing only vertices that have not been processed in the previous traversals. As it does so, it calls the `Traverser`'s `previsit` method on the vertices it traverses in preorder, and calls the `postvisit` method on the vertices it traverses in postorder. You need not use all the spaces provided.

b. Assuming that the implementation of `Traverser` is correct, implement the method `Algs.topoSort`, using the `Traverser` class. You need not use all the spaces provided.

```
public abstract class Traverser {
    /** Action to take on visiting vertex V in G in preorder. */
    protected abstract void previsit(DiGraph G, int v);
    /** Action to take on visiting vertex V in G in postorder. */
    protected abstract void postvisit(DiGraph G, int v);

    /** Perform a depth-first traversal of G, calling .previsit and
     *  .postvisit on all traversed nodes, as described in part (a). */
    public void traverse(DiGraph G) {
        boolean[] marks = new boolean[G.size() + 1];

        for (_______________________________________________________) {

            __________________________________________________________

            __________________________________________________________
        }
    }

    /** Perform a depth-first traversal of G starting from vertex V
     *  and traversing only vertices w such that marked[w] is false.
     *  Set marked[w] to true for all visited vertices, and call
     *  .previsit on all traversed nodes in preorder and .postvisit on all
     *  traversed nodes in postorder, as described in part (a). */
    private void traverse(DiGraph G, int v, boolean[] marks) {
        if (_________________________________________________) {

            ____________________________________________________

            ____________________________________________________

            ____________________________________________________

            for (_______________________________________________) {

                __________________________________________________
            }

            _______________________________________________________

            _______________________________________________________
        }
    }
}
```

```
import java.util.*;

public class Algs {

    /** Return a list of the vertices of G, sorted topologically. Assumes
     *  that G is acyclic. */
    public static List<Integer> topoSort(DiGraph G) {
        TopoTraverser topo = new TopoTraverser();

        ---------------------------------------------------------------

        ---------------------------------------------------------------

        return ----------------------------------------------------------
    }

    private static class TopoTraverser extends Traverser {

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------

        --------------------------------------------------------------
    }
}
```