Section B

Q1	7 /7
Q2	9,5/13
Q3	7/12
Q4	10/18

Compilation penalty
No style penalty applied

Total for Section B 33.5 / 50

Username: jlk21

Compilation: 1 / 1

Model Answer's Tests - ListArrayBasedTest: 5 / 5

Model Answer's Tests - PriorityQueueTest: 7 / 8

Model Answer's Tests - NineTailsWeightedGraphTest: 6 / 7

There are Google style violations of one kind:
  [ModifierOrder], 1 violation(s), including:
    [WARN] ⬋
/home/kgk/Desktop/U22/Java/JIT2223/JIT-SectionB/jlk21/src/ListArrayBased.java:8:17: ⬋
'static' modifier out of order with the JLS suggestions. [ModifierOrder]

Style penalty (capped at -3): -1

Note: if it is below the cap, your total style penalty could be higher if the marker has ⬋
stylistic concerns that go beyond what Checkstyle identifies automatically.

```java
/**
 * You must implement the <code>add</code> method.
 */

public class ListArrayBased<T> implements ListInterface<T> {

  private int maxList = 512;
  private final static int RESIZING_FACTOR = 2;
  private T[] list;
  private int length;

  @SuppressWarnings("unchecked")
  public ListArrayBased() {
    length = 0;
    list = (T[]) new Object[maxList];
  }

  /**
   * Returns true if the list is empty, otherwise returns false.
   */
  public boolean isEmpty() {
    return (length == 0);
  }

  /**
   * Returns the number of elements in the list.
   */
  public int size() {
    return length;
  }

  /**
   * Returns the element in the list at the given position.
   *
   * @param givenPosition the position in the list for which a element is required
   * @throws ListIndexOutOfBoundsException if position is less than 1 or greater than the size of
   *                                       the list
   */
  public T get(int givenPosition) throws ListIndexOutOfBoundsException {
    if (givenPosition >= 1 && givenPosition <= length) {
      return list[translate(givenPosition)];
    } else {
      throw new ListIndexOutOfBoundsException("Position out of range");
    }
  }

  /**
   * <strong>Implement this method for Question 1</strong>
   * Adds the given new element at the given position.
   *
   * @param newItem       the new element to add to the list
   * @param givenPosition the position in the list where the new element has to be added
   * @throws ListIndexOutOfBoundsException if position is less than 1 or greater than size+1 of the
   *                                       list
   */
  public void add(int givenPosition, T newItem)
      throws ListIndexOutOfBoundsException {
    if (givenPosition < 1 || givenPosition > length + 1) {
      throw new ListIndexOutOfBoundsException("Specified index is out of bounds");
    }
    if (givenPosition < length + 1) {
      makeRoom(givenPosition);
    }
    if (list.length == length) {
      list = (T[]) new Object[list.length * RESIZING_FACTOR];
    }
    list[translate(givenPosition)] = newItem;
    length++;
  }

  /**
   * Removes the element in the list at a given position.
   *
   * @param givenPosition the position in the list where the element to be removed is
   * @throws ListIndexOutOfBoundsException if position is less than 1 or greater than the size of
   *                                       the list
   */
  public void remove(int givenPosition) throws ListIndexOutOfBoundsException {
    if (givenPosition >= 1 && givenPosition <= length) {
      if (givenPosition < length) {
        removeGap(givenPosition);
```



```java
      }
      length--;
    } else {
      throw new ListIndexOutOfBoundsException("Position out of range");
    }
  }

  /**
   * Prints the elements in the list.
   */
  public void display() {
    for (int pos = 1; pos <= length; pos++) {
      System.out.println(list[translate(pos)]);
    }
  }

  private void removeGap(int givenPosition) {
    for (int pos = givenPosition + 1; pos <= length; pos++) { // shift left
      // items at
      // position
      // >
      // givenPosition
      list[translate(pos - 1)] = list[translate(pos)];
    }
  }


  private void makeRoom(int givenPosition) {
    for (int pos = length; pos >= givenPosition; pos--) { // shift right items
      // at position >=
      // givenPosition
      list[translate(pos + 1)] = list[translate(pos)];
    } // end for
  }

  private int translate(int givenPosition) {
    return givenPosition - 1;
  }

  /**
   * Returns true if the given element is in the list. False otherwise
   *
   * @param item
   *     the element to search for in the list
   */
  @Override
  public boolean contains(T item) {
    for (int i = 0; i < length; i++) {
      if (list[i].equals(item)) {
        return true;
      }
    }
    return false;
  }
}
```

```java
/**
 * You must implement the <code>constructGraph</code> and <code>constructMinimumSpanningTree</code>
 * methods.
 */

public class NineTailsWeightedGraph {

  private static final int NUM_CONFIGURATIONS = 512;
  private static final int TERMINAL_CONFIGURATION_INDEX = 512;
  private ListInterface<PriorityQueueInterface<WeightedEdge>> configurations;
  private MinimumSpanningTree mst;


  /**
   * Constructs the weighted graph for the nine tails problem, and constructs the minimum spanning
   * tree starting from the target configuration.
   */
  public NineTailsWeightedGraph() {
    constructGraph();
    constructMinimumSpanningTree();
    /* The following versions deviate slightly from the spec, but is more efficient
    constructGraph2();
    constructMinimumSpanningTree2();
    */
  }

  /**
   * <strong>Implement this method for Question 3.</strong>
   */
  private void constructGraph() {
    configurations = new ListArrayBased<>();
    for (int index = 1; index <= NUM_CONFIGURATIONS; index++) {
      configurations.add(index, generateParents(index));
    }
  }

  private void constructGraph2() {
    // Each priority queue stores all the edges with the index as parent,
    // not as child (as in the spec)
    configurations = new ListArrayBased<>();
    for (int index = 1; index <= NUM_CONFIGURATIONS; index++) {
      configurations.add(index, new PriorityQueue<>());
    }
    for (int index = 1; index <= NUM_CONFIGURATIONS; index++) {
      char[] config = indexToConfiguration(index);
      for (int pos = 0; pos < 9; pos++) {
        if (config[pos] == 'H') {
          FlipResult res = flipConfiguration(index, pos);
          WeightedEdge edge = new WeightedEdge(res.newIndex, index, res.numFlips);
          configurations.get(res.newIndex).add(edge);
        }
      }
    }
  }


  /**
   * Returns a copy of the entire weighted graph.
   */
  private ListInterface<PriorityQueueInterface<WeightedEdge>> getConfigurationsCopy() {
    ListInterface<PriorityQueueInterface<WeightedEdge>> copy =
        new ListArrayBased<PriorityQueueInterface<WeightedEdge>>();

    for (int i = 1; i <= NUM_CONFIGURATIONS; i++) {
      copy.add(i, configurations.get(i).clone());
    }

    return copy;
  }

  /**
   * Returns a priority queue of weighted edges that correspond to the legal moves whose child is
   * the configuration with index equal to the given index.
   *
   * @param index the configuration index of a child configuration
   */
  private PriorityQueueInterface<WeightedEdge> generateParents(int index) {
    PriorityQueueInterface<WeightedEdge> parents = new PriorityQueue<WeightedEdge>();
    char[] conf = indexToConfiguration(index);

    for (int pos = 0; pos < 9; pos++) {
      if (conf[pos] == 'H') {
```

2/2 specify element type

what happens when parents is non-empty? 5/10

specify element type


```java
        FlipResult res = flipConfiguration(index, pos);
        // The following line does not do what it is supposed to do:
        // it generates all the edges with index as parent, not as child
        // WeightedEdge edge = new WeightedEdge(index, res.newIndex, res.numFlips);

        // The line has been corrected below:
        WeightedEdge edge = new WeightedEdge(res.newIndex, index, res.numFlips);
        parents.add(edge);
      }
    }
    return parents;
  }

  // **** helper functions ******

  /**
   * Returns the wrapper object flipResult of the configuration generated by applying a legal move
   * to the given configuration index at the given position.
   *
   * <p>flipResult includes the index of the generated configuration and the number of flips.</p>
   *
   * @param confIndex index of the configuration to where a legal move is applied
   * @param position  position in the string of Hs and Ts where the legal move is applied NB:
   *                  position is 0-based
   */
  private FlipResult flipConfiguration(int confIndex, int position) {
    char[] conf = indexToConfiguration(confIndex);
    int row = position / 3;
    int column = position % 3;

    int count = 0;
    if (flipCell(conf, row, column)) {
      count++;
    }
    if (flipCell(conf, row - 1, column)) {
      count++;
    }
    if (flipCell(conf, row + 1, column)) {
      count++;
    }
    if (flipCell(conf, row, column - 1)) {
      count++;
    }
    if (flipCell(conf, row, column + 1)) {
      count++;
    }

    return new FlipResult(configurationToIndex(conf), count);
  }


  private boolean flipCell(char[] conf, int row, int col) {
    if (row >= 0 && row <= 2 && col >= 0 && col <= 2) {
      conf[row * 3 + col] = conf[row * 3 + col] == 'H' ? 'T' : 'H';
      return true;
    } else {
      return false;
    }
  }

  /**
   * Returns the configuration of coin's Hs and Ts that corresponds to a given index.
   *
   * @param index index of a configuration NB: index is a 1-base integer
   */
  public char[] indexToConfiguration(int index) {
    index--; // make it a 0-based index
    char[] conf = new char[9];

    for (int i = 0; i < 9; i++) {
      int digit = index % 2;
      if (digit == 0) {
        conf[8 - i] = 'H';
      } else {
        conf[8 - i] = 'T';
      }
      index = index / 2;
    }
    return conf;
  }

```

```
163:   /**
164:    * Returns the configuration index that corresponds to a given configuration of coin's Hs and Ts.
165:    *
166:    * @param conf configuration of coin's of Hs and Ts NB: result is a 1-base integer
167:    */
168:   public int configurationToIndex(char[] conf) {
169:     int index = 0;
170:     for (int i = 0; i < 9; i++) {
171:       if (conf[i] == 'T') {
172:         index = index * 2 + 1;
173:       } else {
174:         index = index * 2 + 0;
175:       }
176:     }
177:     index++; // make it back to 1-based index
178:     return index;
179:   }
180:
181:
182:   public void printParentsTest(int index) {
183:     printConfiguration(index);
184:     PriorityQueue<WeightedEdge> parents = ((PriorityQueue<WeightedEdge>) configurations
185:         .get(index)).clone();
186:     System.out.println(parents.getSize() + " parents" + ": ");
187:     System.out.println("------------------------------");
188:     while (!parents.isEmpty()) {
189:       WeightedEdge edge = parents.peek();
190:       parents.remove();
191:
192:       printConfiguration(edge.parent);
193:       System.out.println("with weight of " + edge.weight + ".");
194:     }
195:     System.out.println("------------------------------");
196:   }
197:
198:   /**
199:    * Print a configuration with index equal to the given index, as a 3x3 matrix.
200:    *
201:    * @param index index of the configuration of coin's of Hs and Ts to print
202:    */
203:   public void printConfiguration(int index) {
204:     System.out.println("Configuration " + index + ":");
205:     char[] conf = indexToConfiguration(index);
206:     for (int pos = 0; pos < 9; pos++) {
207:       System.out.print(conf[pos]);
208:       System.out.print((pos + 1) % 3 == 0 ? '\n' : " ");
209:     }
210:   }
211:
212:   /**
213:    * Print the shortest path from a given configuration, with index equal to the given source, to
214:    * the target configuration.
215:    *
216:    * @param source index of a given configuration of coin's of Hs and Ts
217:    */
218:   public void printShortestPath(int source) {
219:     int confIndex = source;
220:     System.out.println(
221:         "Shortest path from " + source + " to target configuration (" +
TERMINAL_CONFIGURATION_INDEX
222:             + ") has cost of " + mst.costs[source] + ":");
223:     System.out.println("------------------------------");
224:     while (confIndex != TERMINAL_CONFIGURATION_INDEX) {
225:       printConfiguration(confIndex);
226:       System.out.println("- - - - - - -");
227:       confIndex = mst.nextMoves[confIndex];
228:     }
229:     printConfiguration(TERMINAL_CONFIGURATION_INDEX);
230:     System.out.println("------------------------------");
231:   }
232:
233:   public int getShortestPath(int source) {
234:     return mst.costs[source];
235:   }
236:
237:   /**
238:    * <strong>Implement the rest of this method for Question 4.</strong>
239:    */
240:   private void constructMinimumSpanningTree() {
241:     ListInterface<Integer> visited = new ListArrayBased<Integer>();
242:     int[] nextMoves = new int[NUM_CONFIGURATIONS + 1];
```



```
243:     int[] costs = new int[NUM_CONFIGURATIONS + 1];
244:     // init
245:     for (int i = 1; i <= NUM_CONFIGURATIONS; i++) {
246:       nextMoves[i] = -1; // -1 means not visited yet (i.e., no parental
247:       // information)
248:       costs[i] = Integer.MAX_VALUE;
249:     }
250:
251:     ListInterface<PriorityQueueInterface<WeightedEdge>> confCopy = getConfigurationsCopy();
252:
253:     visited.add(1, TERMINAL_CONFIGURATION_INDEX);
254:     costs[TERMINAL_CONFIGURATION_INDEX] = 0;
255:     // The effect of the following line is already achieved in the for loop above,
256:     // but kept here for clarity
257:     nextMoves[TERMINAL_CONFIGURATION_INDEX] = -1;
258:
259:     while (visited.size() < NUM_CONFIGURATIONS) {
260:       int minCost = Integer.MAX_VALUE;
261:       int bestChild = -1;
262:       int bestParent = -1;
263:
264:       for (int index = 1; index <= NUM_CONFIGURATIONS; index++) {
265:         if (visited.contains(index)) {
266:           continue;
267:         }
268:         PriorityQueueInterface<WeightedEdge> edges = confCopy.get(index);
269:         while (!edges.isEmpty()) {
270:           WeightedEdge minEdge = edges.peek();
271:           if (!visited.contains(minEdge.parent)) {
272:             edges.remove();
273:             continue;
274:           }
275:           int cost = costs[minEdge.parent] + minEdge.weight;
276:           if (cost < minCost) {
277:             minCost = cost;
278:             bestChild = minEdge.child;
279:             bestParent = minEdge.parent;
280:           }
281:           break;
282:         }
283:       }
284:
285:       visited.add(visited.size(), bestChild);
286:       costs[bestChild] = minCost;
287:       nextMoves[bestChild] = bestParent;
288:
289:       // Some removed edges might be relevant in our next iteration, so we need to recover them
290:       confCopy = getConfigurationsCopy();
291:     }
292:
293:     mst = new MinimumSpanningTree(nextMoves, costs);
294:   }
295:
296:   // Compared to constructMinimumSpanningTree:
297:   // 1. We do not need to clone the configurations after every iteration
298:   // 2. We check fewer edges during every iteration
299:   private void constructMinimumSpanningTree2() {
300:     ListInterface<Integer> visited = new ListArrayBased<>();
301:     int[] nextMoves = new int[NUM_CONFIGURATIONS + 1];
302:     int[] costs = new int[NUM_CONFIGURATIONS + 1];
303:     // init
304:     for (int i = 1; i <= NUM_CONFIGURATIONS; i++) {
305:       nextMoves[i] = -1; // -1 means not visited yet (i.e., no parental information)
306:       costs[i] = Integer.MAX_VALUE;
307:     }
308:
309:     visited.add(1, TERMINAL_CONFIGURATION_INDEX);
310:     costs[TERMINAL_CONFIGURATION_INDEX] = 0;
311:     nextMoves[TERMINAL_CONFIGURATION_INDEX] = -1;
312:
313:     while (visited.size() < NUM_CONFIGURATIONS) {
314:       int minCost = Integer.MAX_VALUE;
315:       int bestChild = -1;
316:       int bestParent = -1;
317:
318:       for (int i = 1; i <= visited.size(); i++) {
319:         int parent = visited.get(i);
320:         PriorityQueueInterface<WeightedEdge> edges = configurations.get(parent);
321:         while (!edges.isEmpty()) {
322:           WeightedEdge minEdge = edges.peek();
323:           if (visited.contains(minEdge.child)) {
```

10/18

correct intuition - have you checked all edge cases?

```
            // The child node is already in the shortest path tree
            edges.remove();
            continue;
          }
          int cost = minEdge.weight + costs[parent];
          if (cost < minCost) {
            minCost = cost;
            bestChild = minEdge.child;
            bestParent = parent;
          }
          break;
        }
      }

      if (bestChild == -1) {
        break;
      }
      visited.add(visited.size(), bestChild);
      costs[bestChild] = minCost;
      nextMoves[bestChild] = bestParent;
    }

    mst = new MinimumSpanningTree(nextMoves, costs);
  }

  // *** helper classes

  // MinimumSpanningTree
  private class MinimumSpanningTree {

    int[] nextMoves;
    int[] costs;

    public MinimumSpanningTree(int[] ms, int[] cs) {
      nextMoves = ms;
      costs = cs;
    }
  }

  private class FlipResult {

    int newIndex;
    int numFlips;

    FlipResult(int idx, int flips) {
      newIndex = idx;
      numFlips = flips;
    }
  }


  // WeightedEdge
  @SuppressWarnings("overrides")
  private class WeightedEdge implements Comparable<WeightedEdge> {

    private int weight;
    private int parent;
    private int child;

    public WeightedEdge(int p, int c, int w) {
      this.parent = p;
      this.child = c;
      this.weight = w;
    }

    public int compareTo(WeightedEdge edge) {
      if (weight < edge.weight) {
        return -1;
      } else if (weight == edge.weight) {
        return 0;
      } else {
        return 1;
      }
    }

    @Override
    public String toString() {
      return "Edge: parent=" + parent + ", child=" + child + ", weight="
          + weight + ".";
    }

```



```
    @Override
    public boolean equals(Object obj) {
      if (!(obj instanceof WeightedEdge)) {
        return false;
      }
      WeightedEdge otherEdge = (WeightedEdge) obj;
      return (otherEdge.parent == this.parent && otherEdge.child == this.child
          && otherEdge.weight == this.weight);
    }
  }
}
```

```java
import java.util.Iterator;

/**
 * You must implement the <code>add</code> and <code>PQRebuild</code> methods.
 */

public class PriorityQueue<T extends Comparable<T>> implements
    PriorityQueueInterface<T> {

  private static final int max_size = 512;
  private T[] items;              //a priority queue of elements T
  private int size;               // number of elements in the priority queue.


  @SuppressWarnings({"unchecked", "rawtypes"})
  public PriorityQueue() {
    items = (T[]) new Comparable[max_size];
    size = 0;
  }

  /**
   * Returns true if the priority queue is empty. False otherwise.
   */
  public boolean isEmpty() {
    return size == 0;
  }

  /**
   * Returns the size of the priority queue.
   */
  public int getSize() {
    return size;
  }

  /**
   * Returns the element with highest priority or null if the priority. queue is empty. The priority
   * queue is left unchanged
   */
  public T peek() {
    T root = null;
    if (!isEmpty()) {
      root = items[0];
    }
    return root;
  }

  /**
   * <strong>Implement this method for Question 2</strong>
   * Adds a new entry to the priority queue according to the priority value.
   *
   * @param newEntry the new element to add to the priority queue
   * @throws PriorityQueueException if the priority queue is full
   */
  public void add(T newEntry) throws PriorityQueueException {
    if (size == max_size) {
      throw new PriorityQueueException("Priority queue is full");
    }
    items[size] = newEntry;
    swim(size);
    size++;
  }

  /**
   * "Swims" (or percolates up) the item at the given index
   * such that the min heap property is attained.
   *
   * @param index The index of the item to swim upwards.
   */
  private void swim(int index) {
    int parent = (index - 1) / 2;
    if (items[index].compareTo(items[parent]) < 0) {
      swap(index, parent);
      swim(parent);
    }
  }

  /**
   * Swaps the two items at the specified indices.
   *
   * @param index1 The first index.
```

queue full check - good

6/6

good, coherent code


```java
   * @param index2 The second index.
   */
  private void swap(int index1, int index2) {
    T temp = items[index1];
    items[index1] = items[index2];
    items[index2] = temp;
  }

  /**
   * Removes the element with highest priority.
   */
  public void remove() {
    if (!isEmpty()) {
      items[0] = items[size - 1];
      size--;
      priorityQueueRebuild(0);
    }
  }

  /**
   * <strong>Implement this method for Question 2.</strong>
   */
  private void priorityQueueRebuild(int root) {
    int left = 2 * root + 1;
    int right = 2 * root + 2;
    if (items[left] == null) {
      // This implies items[right] == null as well since min heaps are left complete,
      // which implies root has no children
      return;
    }
    int smaller = (items[right] == null || items[left].compareTo(items[right]) < 0) ? left : right;
    if (items[root].compareTo(items[smaller]) > 0) {
      swap(root, smaller);
      priorityQueueRebuild(smaller);
    }
  }

  public Iterator<Object> iterator() {
    return new PriorityQueueIterator<Object>();
  }

  /**
   * Returns a priority queue that is a clone of the current priority queue.
   */
  @SuppressWarnings({"unchecked", "rawtypes"})
  public PriorityQueue<T> clone() {
    PriorityQueue<T> clone = new PriorityQueue<T>();
    clone.size = this.size;
    clone.items = (T[]) new Comparable[max_size];
    System.arraycopy(this.items, 0, clone.items, 0, size);
    return clone;
  }

  private class PriorityQueueIterator<T> implements Iterator<Object> {

    private int position = 0;

    public boolean hasNext() {
      return position < size;
    }

    public Object next() {
      Object temp = items[position];
      position++;
      return temp;
    }

    public void remove() {
      throw new IllegalStateException();
    }

  }

}
```

3.5/7

Correct thinking, double check implementation
Priority queue order not always respected

```
ListArrayBasedTest - Warnings exist.
src/ListArrayBased.java:65: warning: [unchecked] unchecked cast
      list = (T[]) new Object[list.length * RESIZING_FACTOR];
                   ^
  required: T[]
  found:    Object[]
  where T is a type-variable:
    T extends Object declared in class ListArrayBased
1 warning
Model Answer's Tests - ListArrayBasedTest works!

JUnit version 4.12
.....
Time: 0.02

OK (5 tests)


PriorityQueueTest - Warnings exist.
src/ListArrayBased.java:65: warning: [unchecked] unchecked cast
      list = (T[]) new Object[list.length * RESIZING_FACTOR];
                   ^
  required: T[]
  found:    Object[]
  where T is a type-variable:
    T extends Object declared in class ListArrayBased
1 warning
JUnit version 4.12
...E.....
Time: 0.011
There was 1 failure:
1) rebuild1(PriorityQueueTest)
java.lang.AssertionError: expected:<5> but was:<4>
  at org.junit.Assert.fail(Assert.java:88)
  at org.junit.Assert.failNotEquals(Assert.java:834)
  at org.junit.Assert.assertEquals(Assert.java:645)
  at org.junit.Assert.assertEquals(Assert.java:631)
  at PriorityQueueTest.rebuild1(PriorityQueueTest.java:163)
  at ↙
java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
  at java.base/java.lang.reflect.Method.invoke(Method.java:578)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.junit.runners.Suite.runChild(Suite.java:128)
  at org.junit.runners.Suite.runChild(Suite.java:27)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
  at org.junit.runner.JUnitCore.run(JUnitCore.java:115)
  at org.junit.runner.JUnitCore.runMain(JUnitCore.java:77)
  at org.junit.runner.JUnitCore.main(JUnitCore.java:36)

FAILURES!!!
Tests run: 8,  Failures: 1


NineTailsWeightedGraphTest - Warnings exist.
src/ListArrayBased.java:65: warning: [unchecked] unchecked cast
      list = (T[]) new Object[list.length * RESIZING_FACTOR];
                   ^
  required: T[]
  found:    Object[]
  where T is a type-variable:
    T extends Object declared in class ListArrayBased
1 warning
```



```
JUnit version 4.12
.....E..
Time: 1.99
There was 1 failure:
1) shortestPath398(NineTailsWeightedGraphTest)
java.lang.AssertionError: expected:<8> but was:<22>
  at org.junit.Assert.fail(Assert.java:88)
  at org.junit.Assert.failNotEquals(Assert.java:834)
  at org.junit.Assert.assertEquals(Assert.java:645)
  at org.junit.Assert.assertEquals(Assert.java:631)
  at NineTailsWeightedGraphTest.shortestPath398(NineTailsWeightedGraphTest.java:63)
  at ↙
java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
  at java.base/java.lang.reflect.Method.invoke(Method.java:578)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.junit.runners.Suite.runChild(Suite.java:128)
  at org.junit.runners.Suite.runChild(Suite.java:27)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
  at org.junit.runner.JUnitCore.run(JUnitCore.java:115)
  at org.junit.runner.JUnitCore.runMain(JUnitCore.java:77)
  at org.junit.runner.JUnitCore.main(JUnitCore.java:36)

FAILURES!!!
Tests run: 7,  Failures: 1


AutoTest - Warnings exist.
src/ListArrayBased.java:65: warning: [unchecked] unchecked cast
      list = (T[]) new Object[list.length * RESIZING_FACTOR];
                   ^
  required: T[]
  found:    Object[]
  where T is a type-variable:
    T extends Object declared in class ListArrayBased
1 warning
01  OK  LAB Add at 0

02  OK  LAB Fill array to 512

03  OK  LAB Extend array to 2560

04  OK  PQ  Add evens reversed

05  OK  PQ  Add evens ordered

06  OK  PQ  Add evens random

07  OK  PQ  Add reversed odds then evens

08  OK  PQ  Interleave reversed odds with evens

09  Fail: PQRebuild() not respecting priority PQ  Simple Rebuild Evens

10  OK  PQ  Rebuild Full Odds and Evens


Finished Testing
Result: 9 / 10
Tests: 9 / 10

```