CS146: Spring 2023
Final Exam
Due **Monday**, May 22, at 11:59PM
100 points

You will implement several methods for this final exam and answer some written questions. You must complete the programming component and the written questions **individually**. **You are not allowed to consult your classmates or any other student**. You may reach out to the instructor to ask clarifying questions. You are free to use the textbook, slides, class notes, homework and quiz assignments from class, and the Java API Documentation, but **YOU CANNOT** consult any other resources to help you solve any of the problems on this exam.

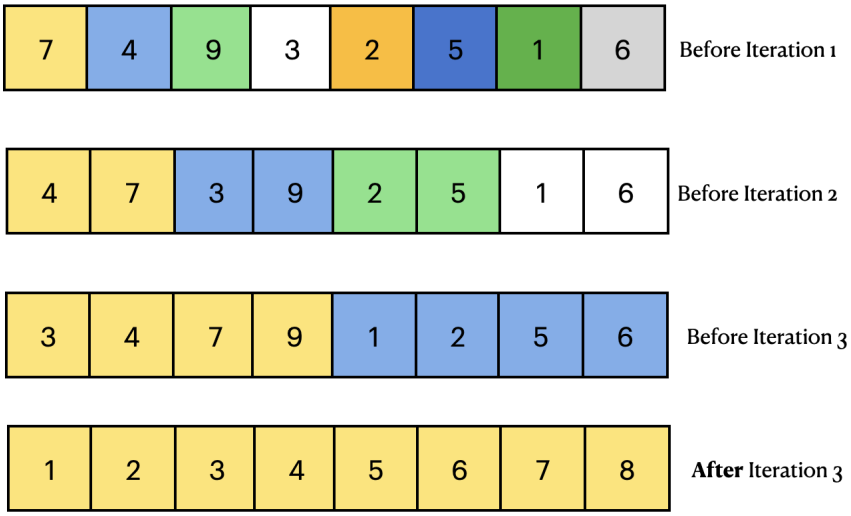
Programming Component (90 points)

The programming component will consist of four problems. To get started please load the starter files (Graph.java, MergeSort.java, BinaryTree.java, and PathException.java) into the finalExam package you create in a new Java Project. **You cannot modify any method signatures or change any of the given code, but you can add whatever helper methods, variables, inner classes, etc. that you think will be helpful in solving this problem.** You are free to use any IDE you like and to test your code however you prefer.

Your programs will be graded not only on correctness but also on efficiency. Efficiency not only encompasses runtime, but also the correct choice of data structures, solving the problem without extraneous passes through the data, etc.

Iterative Merge Sort (20 points)

You will implement a bottom up iterative version of merge sort. The algorithm works as follows. Instead of recursively dividing, think about how you can divide the array using a for loop. You can think of each individual element in the array as already sorted. You are free to reuse the merge function from either Homework 2 or Homework 4.
Use the following example to help guide you.



The first iteration of the loop will merge all pairs of elements to make 4 sorted pairs
The second iteration will merge each pair of element to make two sorted groups of 4
The third iteration will merge each group of 4 sorted elements.

Recreating a Binary Tree (20 points)

You will create a binary search tree given its preOrder traversal. Recall that in a preOrder traversal the node is visited before its children. You need to fill in the code for the constructor method. I've given you a toString method to help with debugging. For example the following array `[7,3,1,5,10,8,12]` would result in the following binary search tree

```
Tree:
7
  3
    1
    5
  10
    8
    12
```

```
public BinaryTree (int[] preOrder)
```

This method should initialize the root variable so that it points to the binary tree with the given preOrder traversal.

Graph Problems (50 points)

You will implement two new graph algorithms: Prim's Minimum Spanning Tree and the Bellman-Ford shortest path algorithm . The starter code looks a lot like your GraphImp class, but without the searching and sorting methods. The constructor will list the vertices sorted in increasing order based on value and the edges associated with that vertex are sorted in increasing order based on the destination (to) vertex. You can assume each vertex will contain a unique value. **You cannot modify the Vertex or Edge classes**.

`public Graph primsMST()`

You will implement Prim's minimum spanning tree algorithm outlined below. The returned Graph object should include all the vertices and the edge set that is associated with the minimum spanning tree. These edges should be bidirectional to represent the subset of the original undirected graph. The minimum spanning tree should originate from the lowest valued vertex in the vertex set.

For this method you must use the built-in priority queue in the Java Collections Framework. You are only allowed to use the methods poll and offer within the priority queue class along with an appropriate constructor. This means that you will have to modify the algorithm to condition the while loop on something other than the size of the priority queue. You will be graded on the efficiency of how you implement this algorithm. **Hint: I suggest you make your own comparator based on the discovery times of the vertex.** You can use this variable as a proxy for the weight of the edge to that vertex in the minimum spanning tree. Please write a comment at the top of your method that describes how you modified the algorithm to use Java's built in priority queue. This description is worth 5 points.

The edges can have weights -50 and to 50. The maximum number of vertices in the graph will not exceed 1000. You can assume that I will only test your methods on undirected, connected graphs consisting of at least 2 vertices. I created a method called unDirectedGraph to help you test your code. This method converts a Graph object into a new Graph object with bidirectional edges.

```
MST-PRIM(G, w, r)
  for each vertex u ∈ G.V
      u.key = ∞
      u.parent = NIL
  r.key = 0
  PQ = Ø
  for each vertex u ∈ G.V
      INSERT(PQ, u)
  while PQ ≠ Ø
      u = EXTRACT-MIN(PQ) // add u to the tree
      for each vertex v in G.Adj[u] // update keys of u's non-tree neighbors
            if v ∈ PQ and w(u, v) < v.key
                  v.parent = u
                  v.key = w(u, v)
                  DECREASE-KEY(Q, v, w(u, v))
```

```
public ArrayList<Integer> shortestPathBF(int source, int target) throws Path
Exception
```

Implement the Bellman-Ford shortest path algorithm described in class. You should return the sequence of values that represent the shortest path from the source to the target starting with the source and ending with the target. I should be able to look up the distance of the path in the discoveryTime instance variable of the target vertex. The edges can have weights -50 and to 50. If a negative cycle exists **on the path from the source to the target** or there is not a path from the source to target, you should throw a PathException. The maximum number of vertices in the graph will not exceed 1000. Please write a comment at the top of your method that describes how you determine if a negative cycle exists along the path from the source to the target. This description is worth 5 points.

```
BELLMAN-FORD(G, w, s)
     initializeSingleSource(G, s)
     for i = 1 to |G.Vertices| - 1
          for each edge (u, v) in G.Edges
               relax(u, v, w)
     for each edge (u, v) in G.Edges
          if v.d > u.d + w(u, v)
               return FALSE
     return TRUE
```

Written Problems (10 points)

Please submit your answers to these problems in a PDF document.

Recurrence Relations (5 points)

Give asymptotically tight upper and lower bounds for T (n) in each of the following algorithmic recurrences. Justify your answers.

A. $T(n) = T(7n/10) + n$

B. $T(n) = 16T(n/4) + n^2$

C. $T(n) = 2T(n/4) + \sqrt{n}$

D. $T(n) = 4T(n/2) + n^2\sqrt{n}$

E. $T(n) = 3T((n/3) - 2) + n/2$ (Hint: think about how you can use an assumption about the importance of the -2 to apply the Masters Theorem)

Reflection (5 points)

Please write a one paragraph summary detailing the grade you would give yourself in this class. I am interested in knowing about how well you feel like you comprehend the material, your ability to independently solve problems, the amount of effort you have put into the course, etc.

Submitting

Export your Eclipse project and written solution and combine them in one zip file that you upload to Canvas. You can export your Eclipse project as either a jar or a zip file. Make sure that the file you submit contains all your .java files. If you upload a file that is missing any of the java source files you will get an automatic deduction of 10 points.