Data Structures
CSCI H343, Fall 2021

Final

Name: ____________________

This exam has 11 questions, for a total of 100 points.

1. [8 points] Implement the `put` method for the following `HashTable` class that uses separate chaining. The `put` method should change the hashtable so that the next time the `get` method is invoked with the same `key`, the given `value` will be returned. You may use the `hash` method (without implementing) to compute the hash of a key. You may assume that each linked list in the `table` is not null.

```
class Entry<K,V> {
  K key;
  V value;
  Entry(K k, V v) { key = k; value = v; }
}

public class HashTable<K,V> implements Map<K,V> {
  private LinkedList<Entry<K,V>>[] table;
  private int hash(K k) { ... }
  private V get(K k) { ... }
  public void put(K key, V value) {
```

Solution:

```
    public void put(K key, V value) {
      int h = hash(key);                              // 1 point
      LinkedList<Entry<K,V> > L = table[h];           // 1 point
      Entry e = null;
      for (Entry<K,V> e2 : L)                         // 2 points
          if (e2.key.equals(key))
              e = e2;
      if (e == null) {
          Entry<K,V> new_e = new Entry(key, value);    // 1 point
          table[h].add(new_e);                         // 2 points
      } else {
          e.value = value;                            // 1 point
      }
  }
```

2. $\boxed{\text{10 points}}$ Fill in the blanks of the `insert` method of the `BinomialQueue` class. Recall that a Binomial Queue maintains a forest (list) of Binomial Heaps ordered by their height (low to high), where each Binomial Heap in the forest has a unique height. The `insert` method of the `BinomialQueue` takes a Binomial Heap and a forest and returns a new forest that includes the given binomial heap and that is still ordered by height and each heap in the forest has a unique height.

   The `link` method of the `BinomialHeap` class takes two Binomial Heaps of the same height and combines them to form a Binomial Heap of one greater height. We represent a forest of binomial heaps with the `PList` class and `null` represents an empty forest.

```
class BinomialHeap<K> {
    int height;
    BinomialHeap<K> link(BinomialHeap<K> other);
    ...
}
class PList<T> {
    static <T> PList<T> addFront(T first, PList<T> rest);
    static <T> T getFirst(PList<T> n);
    static <T> PList<T> getNext(PList<T> n);
    ...
}
class BinomialQueue<K> {
    static <K> PList<BinomialHeap<K>>
    insert(BinomialHeap<K> n, PList<BinomialHeap<K>> ns) {
        if (ns == null) {


            return ___(a)___;
        } else {


            if (n.height < ___(b)___) {
                return PList.addFront(n, ns);
            } else if (n.height == PList.getFirst(ns).height) {


                return insert(___(c)___, PList.getNext(ns));
            } else {


                PList<BinomialHeap<K>> rest = insert(n, ___(d)___);


                return PList.addFront(___(e)___, rest);
            }
        }
    }
    ...
}
```

Solution: (2 points each)

```
(a) PList.addFront(n, null)
(b) PList.getFirst(ns).height
(c) n.link(PList.getFirst(ns))
(d) PList.getNext(ns)
(e) PList.getFirst(ns)
```

3. $\boxed{\text{12 points}}$ 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 initial array and 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).

$$[9, 5, 7, 1, 5]$$

Solution:

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

4. $\boxed{\text{8 points}}$ The following questions are about the Adjacency Matrix representation of graph with n nodes and m edges.

   1. What is the time complexity of inserting an edge between two given nodes?

   2. What is the time complexity of removing an edge between two given nodes?

   3. What is the time complexity of inspecting all the out-edges of one node?

   4. How much space does the Adjacency Matrix consume? (answer using big-O)

Solution: (2 points each)

1. $O(1)$
2. $O(1)$
3. $O(n)$
4. $O(n^2)$

Name: ____________________

5. $\boxed{\text{12 points}}$ What is an optimal DNA sequence aligment for the sequences AGCT and ACG? When computing the score, use +2 for a match, -2 for a mismatch, and -1 for the gap penalty. Show your work by filling in the below dynamic programming table.

		A	G	C	T
A					
C					
G					

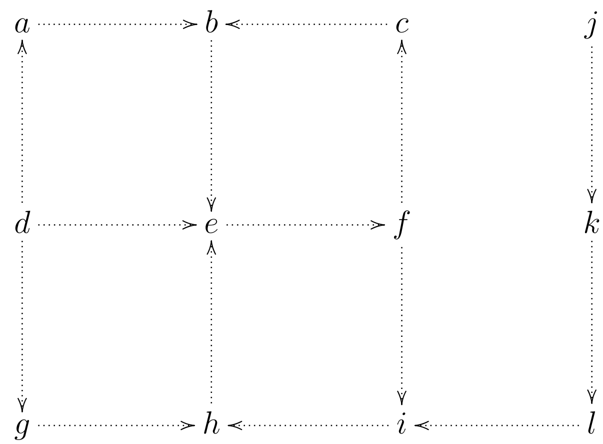
Solution: One of several solutions is

```
AGCT_
A_C_G
```

		A	G	C	T
	0	←-1	←-2	←-3	←-4
A	↑-1	↖ 2	←1	←0	←-1
C	↑-2	↑1	↑0	↖3	←2
G	↑-3	↑0	↖3	↑2	↑1

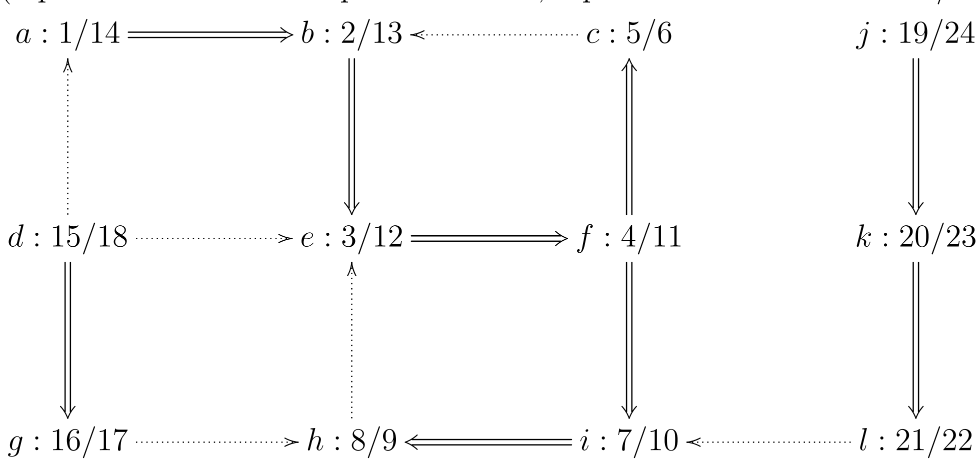
(3 points for correct initialization of the table. 6 points for the rest of the table. 3 points for correct result from traceback.)

6. $\boxed{\text{10 points}}$ Perform depth-first search on the following graph, marking the edges in the depth-first forest and recording the discover time and finish time for each node that is visited by the DFS. Start the times at 1. When choosing the order in which to process and visit nodes, give priority to nodes that are earlier in the alphabet (`a` before `b`).



Solution:

(6 points for a correct depth-first forest, 4 points for correct discover/finish times)



7. 10 points Fill in the blanks to complete the following implementation of Breadth-First Search for the Routing Wires project. The `BFS` function should return `true` if it finds a path from the `start` to the `end` of `endpoints` and returns `false` otherwise. `BFS` should record the path in the `parents` map. Recall that the `Board` class has a method named `adj` that returns the list of adjacent coordinates of the given coordinate. The `Queue` class has methods `add` and `remove` for pushing and popping elements from the queue.

```
static boolean
BFS(Board board, Endpoints endpoints, Map<Coord, Coord> parents) {
    Queue<Coord> queue = new LinkedList<>();
    Set<Coord> visited = new HashSet<>();


    ___(a)___;
    while (!queue.isEmpty()) {


        Coord current = ___(b)___;
        visited.add(current);
        if (current.equals(endpoints.end))
        return true;


        for (Coord adj : ___(c)___) {
            if ((board.getValue(adj) == 0 || adj.equals(endpoints.end))
                    && ___(d)___) {


                queue.add(adj);


                parents.put(adj, ___(e)___);
            }
        }
    }
    return false;
}
```

Solution: (2 points each)

```
(a) queue.add(endpoints.start)
(b) queue.remove()
(c) board.adj(current)
(d) ! visited.contains(adj)
(e) current
```

8. $\boxed{\text{10 points}}$ What is the time complexity of building a Huffman Tree given a frequency table for n characters. Explain your answer.

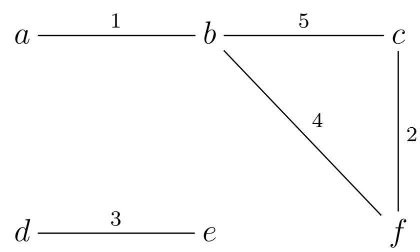
Solution:

1. Fill an array with the n single-character trees: $O(n)$. (2 points)
2. Turn the array into a heap: $O(n)$. (2 points)
3. While there is more than one tree in the heap, pop two, combine them, and push them back into the heap. The push and pop into the heap is $O(\log n)$, and we do that $O(n)$ times, so this step is $O(n \log n)$. (4 points)

The max of the three steps is $O(n \log n)$, so the total time complexity is $O(n \log n)$. (2 points)

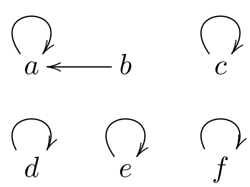
Name: ______________________

9. $\boxed{\text{7 points}}$ Find the connected components of the following graph using Disjoint Sets, but without optimizations (no path compression, no union-by-rank). Draw the Disjoint Sets forest after each `union` operation. Process the edges of the graph in the order written on the edges, i.e. start with edge a–b, then c–f, etc. When choosing a new root, give priority to nodes that are lower in the alphabet.

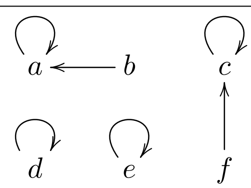


Solution: (1 point each)

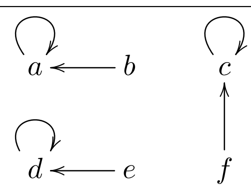
Union $a - b$:



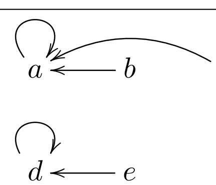
Union $c - f$:



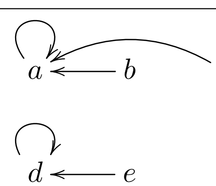
Union $d - e$:



Union $b - f$:



Union $b - c$: (do nothing)



The connected components are: $\{a, b, c, f\}$ and $\{d, e\}$. (2 points)

10. [10 points] Fill in the blanks to complete the following implementation of the Rod Cutting algorithm. Recall that it solves the optimization problem of cutting a rod of length n into smaller pieces in a way that maximizes the total amount that the smaller pieces can be sold for. The array `P` maps rod lengths to their market price. The algorithm fills in the array `R`, which maps each possible input rod length to a `CutResult` object. The algorithm returns the best `CutResult`.

```
class CutResult {
   CutResult(int c, int amt, CutResult r) { cut = c; price = amt; rest = r; }
   int cut;
   int price;
   CutResult rest;
}

static CutResult cut_rod(int[] P, int n, CutResult[] R) {
   if (R[n] != null) {


      return ___(a)___;
   } else if (n == 0) {
      R[n] = new CutResult(0, 0, null);
      return R[n];
   } else {
      CutResult best = null;
      for (int i = 1; i != n+1; ++i) {


          CutResult rest = ___(b)___;
          int price = P[i] + rest.price;


          if (best == null || ___(c)___) {


             best = ___(d)___;
          }
      }


      ___(e)___
      return best;
   }
}
```

Solution: (2 points each)

```
(a) R[n]
(b) cut_rod(P, n - i, R)
(c) best.price < price
(d) new CutResult(i, price, rest)
(e) R[n] = best;
```

11. $\boxed{\text{3 points}}$ What advice would you give a student taking Data Structures next year?

Solution: Open ended.