Name: ________________________________

NetID/Login (CAPITAL LETTERS): ____________

Honor code acknowledgment (signature) ____________________________

This test has 13 pages (with a help page at the end), be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes. Write clearly and legibly. If we cannot read it, we cannot grade it.

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages are imported in any code you write.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>40 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>24 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td>90 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1:  \textit{(Multiple choice! (10 points))}

Cross off three incorrect answers for partial credit.

1. Which data structure allows deleting data elements from the front and adding at the back?
   (a) Stack
   (b) Queue
   (c) Binary-search tree
   (d) Map
   (e) Hash Table

2. The pre-order traversal of a binary-search tree is DBACFE. What is the post-order traversal? (It is a search tree)
   (a) ABFCDE
   (b) ADBFEC
   (c) ABFECED
   (d) ACBEFD
   (e) ABDFCE

3. The in-order traversal of a binary tree is HFIEJGZ, and the post-order traversal of the same tree is HIFJZGE. What will be the total number of nodes in the left sub tree of the given tree? (It is NOT a search tree)
   (a) 1
   (b) 2
   (c) 3
   (d) 4
   (e) 5

4. What are the sequence of popped out values if the sequence of operations - push(1), push(2), pop, push(1), push(2), pop, pop, pop, push(2), pop are performed on a stack.
   (a) 2, 2, 1, 1, 2
   (b) 2, 2, 1, 2, 2
   (c) 2, 1, 2, 2, 1
   (d) 2, 1, 2, 2
   (e) 2, 1, 2, 1, 2

5. How many binary trees with 3 nodes, A, B, and C when traversed in post-order will give the sequence A, B, C? (It is NOT a search tree)
   (a) 3
   (b) 7
   (c) 5
   (d) 4
   (e) 6
PROBLEM 2:  (Timber! (40 points))

Part A: 2 points Draw the resulting binary-search tree when adding the following numbers in order to an initially empty search tree.

\[
\begin{array}{c}
7 \\
/ \ \ \\
1 \ 10 \\
\ \ / \\
6 \ 8
\end{array}
\]

Part B: 2 points Draw the resulting min-heap when adding the following numbers in order to an initially empty heap.

\[
\begin{array}{c}
1 \\
/ \ \ \\
6 \ 8 \\
/ \ \ \\
10 \ 7
\end{array}
\]
**Part C: 16 points** What is the average running time to perform the following operations with a balanced **Binary-Search Tree** and a **Min-Heap**, each of N elements? Complete the following chart with running-time and give a brief justification for each operation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary-Search Tree</th>
<th>Min-Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find if a specific element is in your dataset</td>
<td>$O(lgN)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>Search through height of tree ≠ nodes</td>
<td>Search through all nodes</td>
</tr>
<tr>
<td>Print the elements in ascending order</td>
<td>$O(N)$</td>
<td>$O(NlgN)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>in-order traversal</td>
<td>remove top node O(1), reheap O(lgN), repeat N times</td>
</tr>
<tr>
<td>Find the minimum element in your dataset</td>
<td>$O(lgN)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>bottom left-most node</td>
<td>root of tree</td>
</tr>
<tr>
<td>Find the maximum element in your dataset</td>
<td>$O(lgN)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>bottom right-most node</td>
<td>in a leaf, but look at all of the them</td>
</tr>
</tbody>
</table>
Part D: 4 points
Draw a binary-search tree of height 4 with the *minimum* number of nodes possible in a binary-search tree of height 4.

```
1
 \  
  2
   \ 
    3
     \ 
      4
```

Draw a binary-search tree of height 4 with the *maximum* number of nodes possible in a binary-search tree of height 4.
This is a full binary-search tree with the following number of nodes on each level:

1 node
2 nodes
4 nodes
8 nodes

Part E: 4 points
Draw a min-heap of height 4 with the *minimum* number of nodes possible in a heap of height 4.

```
1 node
2 nodes
4 nodes
1 node (the left-most node)
```

Draw a min-heap of height 4 with the *maximum* number of nodes possible in a heap of height 4.
This is a full heap with the following number of nodes on each level:

1 node
2 nodes
4 nodes
8 nodes
The ExamBST class has been started below. The ExamBST class is similar to the binary-search tree code that you used in lecture and recitation. You can assume that the tree rooted at myRoot is a binary-search tree.

```java
class ExamBST {
    Node myRoot = null;

class Node {
    int myValue;
    Node myLeft; // holds smaller tree nodes
    Node myRight; // holds larger tree nodes

    Node(int val) {
        myValue = val;
    }
}

public String toStringEven()
    return toStringEven(myRoot);
}

public String toStringEven(Node cur){
    if(cur == null){
        return "";
    }
    String leftString = toStringEven(cur.myLeft);
    String rightString = toStringEven(cur.myRight);
    if(cur.myValue %2 == 0){
        return leftString + " " + cur.myValue + " " + rightString;
    }
    return leftString + " " + rightString;
}
```

**Part B: 12 points** Complete the method `toStringEven`, in the ExamBST class, that returns a String of the even values in the binary-search tree, rooted at myRoot, in numerical order. Return an empty string if the tree contains no even nodes.
PROBLEM 3:  (*Lista! (24 points)*) 

The following questions use the ExamLinkedList class that has been started below.

```java
public class ExamLinkedList {
    private Node myHead;

    public class Node{
        private int myData;
        private Node myNext;

        public Node(int data, Node next){
            myData = data;
            myNext = next;
        }
    }
}
```

**Part A: 12 points** Complete the code below in the ExamLinkedList class, such that the method add(ExamLinkedList list) adds the linked-list list to the front of this ExamLinkedList. For example:

- this: \(1 \rightarrow 2 \rightarrow 3\)
- list: \(4 \rightarrow 5\)
- this after call add(list): \(4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3\)

```java
public void add(ExamLinkedList list){
    add(list.myHead);
}

public void add(Node list){
    Node cur = list;
    while(cur.myNext != null){
        cur = cur.myNext;
    }
    cur.myNext = this.myHead;
    this.myHead = list;
}
```

**Part B: 12 points** We implemented linked-lists in class two different ways. One implementation used only a head pointer pointing to the front of the list. The second used both a head and tail pointer pointing to the front and back of the list respectively. Complete the following chart computing the average running times based on the two implementation of linked-lists. The operation are performed on two linked lists, `listA` of length $N$ and `listB` of length $M$.

<table>
<thead>
<tr>
<th>Operation Description</th>
<th>Head pointer</th>
<th>Head and tail pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add <code>listB</code> to the beginning of <code>listA</code></td>
<td>$O(M)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>iterate through <code>listB</code> to find tail</td>
<td>go directly to tail</td>
</tr>
<tr>
<td>Add <code>listB</code> in the exact middle of <code>listA</code></td>
<td>$O(N + M)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>iterate to end of <code>listB</code> $O(M)$ iterate to middle of <code>listA</code> $N/2$ Remove constants Remove constants</td>
<td>go directly to tail of <code>listB</code> $O(1)$ iterate to middle of <code>listA</code> $N/2$</td>
</tr>
<tr>
<td>Add <code>listB</code> in the end of <code>listA</code></td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Justification</strong></td>
<td>iterate to end of <code>listA</code> $O(N)$</td>
<td>go directly to tail of <code>listB</code> and <code>listA</code> $O(1)$</td>
</tr>
</tbody>
</table>
PROBLEM 4: (GridGame! (16 points))

Below is an almost complete solution to the GridGame APT that we completed in class. Complete the code as denoted by the comments so that you will get an all green solution. There are four comments, and four places where you will add code. The GridGame APT writeup can be found at the end of this question.

```java
public class GridGame {
    public int winningMoves(String[] grid){
        char[][] charGrid = new char[4][4];

        for(int i = 0; i < grid.length; i++){
            charGrid[i] = grid[i].toCharArray();
        }

        return countWins(charGrid);
    }

    public int countWins(char[][] charGrid){
        int numWins = 0;

        for(int i = 0; i < charGrid.length; i++){
            for(int j = 0; j < charGrid[i].length; j++){
                if(canPlace(charGrid, i, j)){
                    //place something on the board
                    charGrid[i][j] = 'X';

                    //determine if the opponent wins
                    int oppwins = countWins(charGrid);
                    if (oppwins == 0) numWins += 1;

                    //backtrack
                    charGrid[i][j] = '.';
                }
            }
        }

        return numWins;
    }
}
```

Continued on next page.
public boolean canPlace(char[][] charGrid, int r, int c) {
    if(charGrid[r][c] == 'X') return false;
    // complete canPlace

    if( (r > 0) && charGrid[r-1][c] == 'X') return false;
    if( (r < 3) && charGrid[r+1][c] == 'X') return false;
    if( (c > 0) && charGrid[r][c-1] == 'X') return false;
    if( (c < 3) && charGrid[r][c+1] == 'X') return false;
    return true;
}
GridGame APT

Problem Statement

In a simple game, two players take turns placing 'X's in a 4x4 grid. Players may place 'X's in any available location ("." in the input) that is not horizontally or vertically adjacent to another 'X'. The player who places the last 'X' wins the game. It is your turn and you want to know how many of the moves you could make guarantee you will win the game, assuming you play perfectly.

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
</table>
| public class GridGame {
|     public int winningMoves(String[] grid){
|         // fill in code here
|     }
| } |

Constraints

- grid will contain exactly 4 elements.
- Each element of grid will contain 4 characters ("X"s or "."s), inclusive.
- There will be no two horizontally or vertically adjacent 'X's in grid.

Examples

1. 
   
   {".....",
    ".....",
    ".....",
    "....."}

   Returns: 0

   You can't win this game.

2. 
   
   {".....",
    ".....",
    ".X..",
    "....."}

   Returns: 11

   Any legal move guarantees you win the game.
String

- `.length()` Get the length of the String. $O(1)$.
- `.charAt(i)` Get the char at index $i$. $O(1)$.
- `.split(" ")` Split a string by spaces and store it in a string[].
- `.substring(i, j)` Get the substring between indices $i$ and $j$. Index $i$ is inclusive, and index $j$ is exclusive. $O(1)$. For example:

```java
String x = "abcdefg";
String y = x.substring(2, 4);
// y now has the value "cd"
```

ArrayList<T> // Where T is a type, like String or Integer

- `.add(i, X)` Add element $X$ to the list at index $i$. If no $i$ is provided, add an element to the end of the list. Adding to the end runs in $O(1)$.
- `.get(i)` Get the element at position $i$. Runs in $O(1)$.
- `.set(i, X)` Set the element at position $i$ to the value $X$. $O(1)$.
- `.size()` Get the number of elements. $O(1)$.

HashSet<T> // Where T is a type, like String or Integer

- `.size()` Compute the size. $O(1)$.
- `.add(X)` Add the value $X$ to the set. If it’s already in the set, do nothing. $O(1)$.
- `.contains(X)` Return a boolean indicating if $X$ is in the set. $O(1)$.
- `.remove(X)` Remove $X$ from the set. If $X$ was not in the set, do nothing. $O(1)$.

HashMap<K, V> // Where K and V are the key and value types, respectively.

- `.size()` Compute the size. $O(1)$.
- `.containsKey(X)` Determines if the map contains a value for the key $X$. To get that value, use `.get()`. $O(1)$.
- `.get(X)` Gets the value for the key $X$. If $X$ is not in the map, return null. $O(1)$.
- `.put(k, v)` Map the key $k$ to the value $v$. If there was already a value for $k$, replace it. $O(1)$.
- `.keySet()` Return a Set containing the keys in the map. Useful for iterating over. $O(1)$.

To iterate over a HashSet<T>, use

```
for (T v : nameOfSet) {
    // v is the current element of the set.
}
```

This can be combined with HashMap’s `.keySet()` to iterate over a HashMap.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$