

1 (10) Part A. Use binary search to find the value “14” in the following list. Write the sequence of numbers (and indexes for the numbers) you examined. There are 31 elements in the list; the top numbers are the indexes and the bottom numbers are the actual values. No code is required.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  
1 1 2 2 2 3 3 4 5 5 6 7 9 12 12 13 14 16 18 19 21 23 28 32 45 85 87 89 94 95 98

Part B: Now show the sequence of steps to **try** to find the value “24”.

2. (20) Part A: Sort the following list using Insertion Sort. Show all steps. Is this sort stable? Why? No code is required.

Index: 0 1 2 3 4 5 6  
Value: 9 3 1 8 5 7 3

Part B: Sort the same list using QuickSort. Show all steps. Is this sort stable? Why?

3 (20). You want to write a program to find the solution to a maze. The maze is contained in a two-dimensional array with width WIDTH and height HEIGHT. An example maze is pictured below; 0s represent passage, 1s represent walls, 2s represent explored passage, and 3 is the exit. The starting point will be provided (start and exit can be anywhere on the board).

```
1 1 1 1 1 1 1 1 3 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1
1 0 1 1 1 1 1 0 0 1 1 1
1 0 0 0 1 0 1 0 1 1 1 1
1 1 1 0 1 0 1 0 0 0 0 1
1 0 0 0 1 0 1 1 1 1 0 1
1 0 1 1 1 0 1 1 1 1 0 1
1 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 0 1
```

Using recursion or a stack, return the location of the exit (by navigating the maze from the start). The top-left corner is given by (0, 0). You only need to write the findExit method and any necessary recursive methods. Remember that the Point class has two public member variables x and y and a constructor of the form Point(int x, int y).

```
public class MazeSolver {
    public static final int WIDTH = ###;
    public static final int HEIGHT = ###;

    private int[][] grid;

    ... Methods to initialize maze here...

    public Point findExit(Point start) {

        }
        // If you choose recursion, treat findExit() as a helper method and write
        // your recursive method(s) on separate page.
    }
}
```



4 (15). You want to find the word that occurs most often in a String. Assume that the String has only words and spaces, and has no punctuation. You must use a Hashtable to solve this problem. Specifically, you will use the get and put methods. Remember to specify types for your Hashtable (using bracket notation). Complete the method below.

```
public String findMostFrequentWord(String words) {
```

```
}
```

5 (15). You have the following definitions for LinkedList and Node

<pre>public class LinkedList {     protected Node head;      public Node getFirst() {         return head;     }      public void setFirst(Node head) {         this.head = head;     } }</pre>	<pre>class Node {     private Object element;     private Node next;      public Node(Object element, Node next) {         this.element = element;         this.next = next;     }      public Object getElement() {         return element;     }      public Node getNext() {         return next;     }      public void setNext(Node next) {         this.next = next;     } }</pre>
---	--

Write a class "Queue" that extends LinkedList and implements the following methods (to model a FIFO queue):

```
public Object pop()
```

```
public void push(Object element)
```

```
public Boolean isEmpty()
```



6 (15). Part A. Imagine that you are a tester for a vending company and you need to test a soda machine. The machine in question dispenses  $n$  kinds of soda and has one button for each. The machine takes coins and dollar bills, and will only dispense a soda when at least  $X$  cents have been inserted. Write a brief procedure for 3 tests you would perform. Remember to include set-up/tear-down steps where appropriate.

Part B. Specify two pre-conditions for the method below. Write Java *assert* statements to make sure these conditions are not violated. Assume (for Part B only) that the machine accepts only dollars and, for change, returns only quarters.

```
// Use these variables in your method.
private final static double PRICE_OF_SODA = 1.25;
private int[] canInventory;
private int numQuartersAvailable;

// This method deducts a can from the inventory and quarters from the change repository.
// Precondition 1:
// Precondition 2:
public void makeDeductions(int inventoryIndex, double amountPaid) {

}
```

7 (5). Explain the difference between daemon and user threads.