1. A program is to be written that simulates and keeps track of the random motion of a point whose position is represented by coordinates (x, y). The point starts at $(0, 0)$ at time $= 0$. It is to move randomly a large, but unknown, number of times. A record of its (x, y) positions must be kept so as to be able to re-create any part of its path starting from a given previously recorded (x, y) position. The program is to print the point's (x, y) movements, forward or backward in time, from the given (x, y) position. You may assume that no point is visited more than once. Assuming the existence of a `Point` class that holds a pair of coordinates, which of the following is the best data structure for the task?

   (A) A one-dimensional array of `Point` objects
   (B) A two-dimensional array of integers in which the array indexes represent the position visited by the point and each integer cell of the array is a counter that keeps track of the number of moves to that position
   (C) A circular doubly linked list of `Point` objects
   (D) A stack of `Point` objects
   (E) A queue of `Point` objects

Questions 2–4 refer to the `TennisPlayer`, `GoodPlayer`, and `WeakPlayer` classes below. These classes are to be used in a program to simulate a game of tennis.

```
public abstract class TennisPlayer
{
    private String myName;

    //constructor
    public TennisPlayer(String name)
    { myName = name; }

    public String getName()
    { return myName; }

    public abstract boolean serve();
    public abstract boolean serviceReturn();
}

public class GoodPlayer extends TennisPlayer
{
    //constructor
    public GoodPlayer(String name)
    { /* implementation not shown */ }

    //Postcondition: Return true if serve is in (80% probability),
    //               false if serve is out (20% probability).
    public boolean serve()
    { /* implementation not shown */ }
```

```
    //Postcondition: Return true if serve is in (80% probability),
    //                false if serve is out (20% probability).
    public boolean serve()
    { /* implementation not shown */ }

    //Postcondition: Return true if service return is in
    //                (70% probability), false if service return
    //                is out (30% probability).
    public boolean serviceReturn()
    { /* implementation not shown */ }
}

public class WeakPlayer extends TennisPlayer
{
    //constructor
    public WeakPlayer(String name)
    { /* implementation not shown */ }

    //Postcondition: Return true if serve is in (45% probability),
    //                false if serve is out (55% probability).
    public boolean serve()
    { /* implementation not shown */ }

    //Postcondition: Return true if service return is in
    //                (30% probability), false if service return
    //                is out (70% probability).
    public boolean serviceReturn()
    { /* implementation not shown */ }
}
```

2. Which of the following declarations will cause an error? You may assume all the constructors are correctly implemented.

(A) `TennisPlayer t = new TennisPlayer("Smith");`
(B) `TennisPlayer g = new GoodPlayer("Jones");`
(C) `TennisPlayer w = new WeakPlayer("Henry");`
(D) `TennisPlayer p;`
(E) `WeakPlayer q = new WeakPlayer("Grady");`

3. Refer to the `serve` method in the `WeakPlayer` class:

```
//Postcondition: Return true if serve is in (45% probability),
//               false if serve is out (55% probability).
public boolean serve()
{ /* implementation */ }
```

Which of the following replacements for `/* implementation */` satisfy the postcondition of the `serve` method?

I
```
double value = Math.random();
return value >= 0 || value < 0.45;
```

II
```
double value = Math.random();
return value < 0.45;
```

III
```
int val = (int) (Math.random() * 100)
return val < 45;
```

3. Refer to the `serve` method in the `WeakPlayer` class:

```
//Postcondition: Return true if serve is in (45% probability),
//               false if serve is out (55% probability).
public boolean serve()
{ /* implementation */ }
```

Which of the following replacements for `/* implementation */` satisfy the post-condition of the `serve` method?

I
```
double value = Math.random();
return value >= 0 || value < 0.45;
```

II
```
double value = Math.random();
return value < 0.45;
```

III
```
int val = (int) (Math.random() * 100)
return val < 45;
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

4. Consider the following class definition:

```
public class Beginner extends WeakPlayer
{
    private double myCostOfLessons;

    //methods of Beginner class
        ...
}
```

Refer to the following declarations and method in a client program:

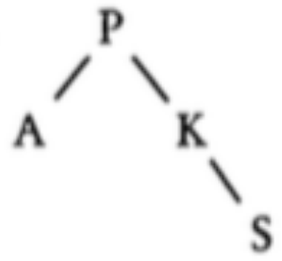
```
TennisPlayer g = new GoodPlayer("Sam");
TennisPlayer w = new WeakPlayer("Harry");
TennisPlayer b = new Beginner("Dick");

public void giveEncouragement(WeakPlayer t)
{ /* implementation not shown */ }
```

Which of the following method calls will *not* cause an error?

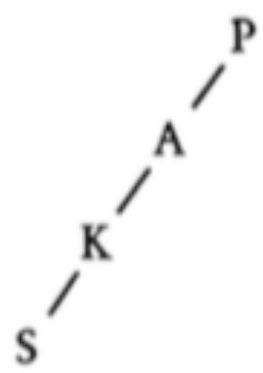
(A) `giveEncouragement((WeakPlayer) g);`
(B) `giveEncouragement((WeakPlayer) b);`
(C) `giveEncouragement((Beginner) w);`
(D) `giveEncouragement(w);`
(E) `giveEncouragement(b);`

5. Inorder and postorder traversals yield the same output for which of the following trees?

(A)

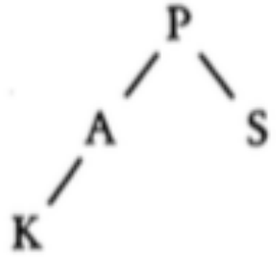


(B)

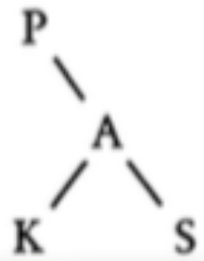


(C)

(D)



(E)



6. Worst case run time is *never* $O(n^2)$ for which of the following sorting algorithms?

I Mergesort
II Heapsort
III Quicksort

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III