1. a1{ 24 12 } // left is bottom of stack, right is top
a2 { 3 6 12 12 }
a3 { 24 12 }

A1and a3 will be identical at the end because they refer to the same object. They are not copies of each other they are each other.
Filler will fill a1 with numbers 24 through 3, in that order, ignoring 48 as it is larger than 40. The code will push 3 and 6 to a2, popping them from a1, then push the top of the stack, 12, of a1 without popping. A3 will pop from 1, and then push the same number to a3, resulting in nothing changing since a1 and a3 are the same.
The top of a1 and a3 (12) will then be popped and pushed to a2, which is 12.

2. Array underlying queue is
 0 5
 1 4
 2 3
 3 2
 4 1
 5 null
 6 null
 7 8
 8 7
 9 6

The queue is {8 7 6 5 4 3 2 1} where items are added at the 1 and removed from 8.

3.
search(4, arr);

searchUtil(4, arr, 0, 19);

searchUtil(4, arr, 0, 8);

searchUtil(4, arr, 0, 3);

searchUtil(4, arr, 2, 3);

searchUtil(4, arr, 3, 3);

return 3;

Search (377, arr);

searchUtil(337, arr, 0, 19);

searchUtil(337, arr, 10, 19);

searchUtil(337, arr, 10, 13);

searchUtil(337, arr, 12, 13);

searchUtil(337, arr, 12, 11);

Return -1;

4. public class StackPQ<V> extends PriorityQueue<K,V> implements StackInterface<V>{
  public boolean push(V value) {
      return offer(size(), value);
  }
  public V pop() {
      return poll();
  }
}

5.
Using a super lo-res awful representation for trees

Pt. 1

3

--
20
3

--
99
3, 20

--
99
75, 20
3

--
110
75, 99
17,4, 20 60
3

Pt 2. array = {110, 75, 99, 17, 4, 20, 60, 3, , , ...}

Pt 3.

75
17, 60
3, 4, 20
Array is {75, 17, 60, 3, 4, 20, , , ...}

6.

```
public class Student implements Comparable<Student> {
private String name;
protected int age;
public int compareTo(Student other) {
   int aa = name.compareTo(other.name);
   if (aa!=0) return aa;
   return age-other.age;
   }
}
```

7.

```
public class BigRec {

   public void recPrint(int num, int cnt) {
      if (num == 0)
         return;
      System.out.println(cnt + " " + num % 10);
      recPrint(num / 10, cnt+1);
   }

   public void recPrintr(int num, int cnt) {
      if (num == 0)
         return;
      recPrintr(num / 10, cnt+1);
      System.out.println(cnt + " " + num % 10);
   }

   public static void main(String[] args) {
      BigRec br = new BigRec();
      br.recPrint(123, 1);
      br.recPrintr(123, 1);
   }

}
```

8.

This program will print out this because it will first go to doo1, call doo2 with (5, 1, 1), meaning it will go through 5 times (num of items in the row/column,) and print out cr * ti. For the first row, that is ti. When it calls doo2 for the second time in doo1, it will continually increment ti by one. So in the second row, it multiplies each by 2.

2 *2 = 4, 3 * 2 = 6, etc. etc. row 3 multiples by 3 , row 4 by 4 , etc.

In short, the program will go row by row and multiple the column by the number of the row, all recursively.