1 EX 18.4 consider the following list:

90	8	7	56	123	235	9	1	653

Show a trace execution for:

(a) Selection Sort
(b) Insertion Sort
(c) Bubble Sort
(d) Quick Sort
(e) Merge Sort

Selection Sort:

Algorithm: Sort a list of values by repetitively putting a particular value into its final, sorted position.

90	8	7	56	123	235	9	1	653
1	8	7	56	123	235	9	90	653
1	7	8	56	123	235	9	90	653
1	7	8	9	123	235	56	90	653
1	7	8	9	56	235	123	90	653
1	7	8	9	56	90	123	235	653

Insertion Sort:
Algorithm: Sort a list of values by repetitively inserting a particular value into a subset of the list that has already been sorted.

90	8	7	56	123	235	9	1	653	**Shifted**
8	90	7	56	123	235	9	1	653	1
7	8	90	56	123	235	9	1	653	2
7	8	9	90	56	123	235	1	653	4
1	7	8	9	90	56	123	235	653	7
1	7	8	9	56	90	123	235	653	1

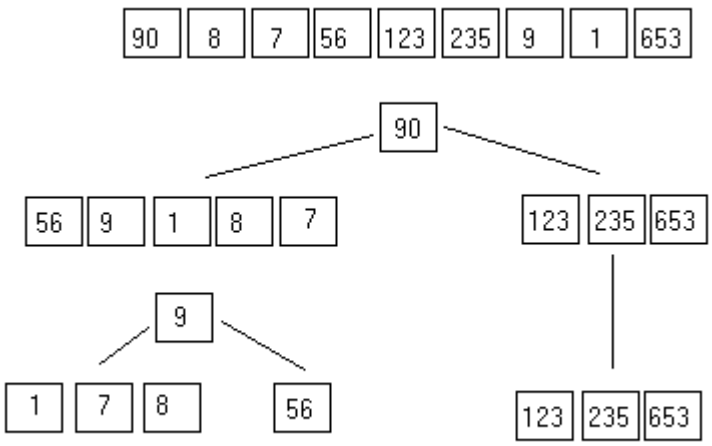
Bubble Sort:

Algorithm: Sort a list by repeatedly comparing neighboring elements and swapping them if necessary.

90	8	7	56	123	235	9	1	653
8	90	7	56	123	235	9	1	653
8	7	90	56	123	235	1	9	653
8	7	56	90	123	1	235	9	653
8	7	56	90	1	123	235	9	653
8	7	56	1	90	123	235	9	653
8	7	1	56	90	123	235	9	653
8	1	7	56	90	123	235	9	653
1	8	7	56	90	123	235	9	653
Second Pass								
1	8	7	56	90	123	235	9	653
1	8	7	56	90	123	235	9	653
1	8	7	56	90	123	9	235	653
1	8	7	56	90	9	123	235	653
1	8	7	56	9	90	123	235	653
1	8	7	9	56	90	123	235	653
1	8	7	9	56	90	123	235	653
1	7	8	9	56	90	123	235	653

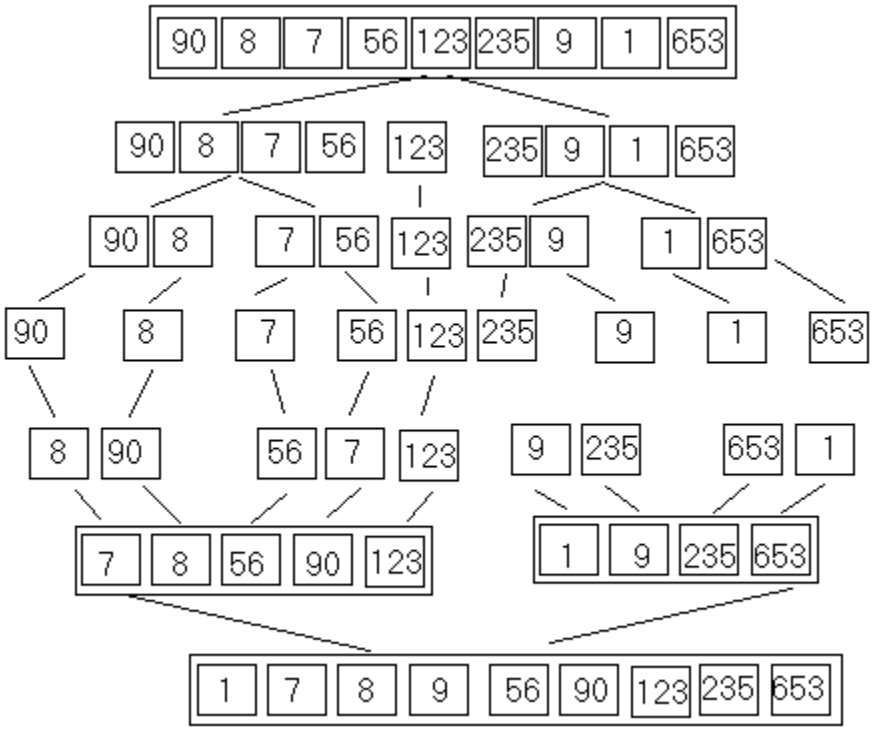
Quick Sort:

Algorithm: Sort a list by partitioning the list and then recursively sorting the two partitions.



Merge Sort:

Algorithm: Sort a list by recursively dividing the list in half until each sublist has one element and then merging these sublists into the sorted order.



PP 18.1 The bubble sort algorithm shown in this chapter is less efficient than it can be. If a pass is made throught the list without exchanging any elements, this means that the list is sorted and there is no reason to continue. Modify this alforithm so that it will stop as soon as it recognizes that the list is sorted. Do not use a break statement.

One possible solution:

```
/**
   * Sorts the specified array of objects using a bubble sort
   * algorithm.
   *
   * @param data the array to be sorted
   */
```

```
public static <T extends Comparable<T>>
            void bubbleSort(T[] data)
{
   int scan;    //This variable serves as a traverser
   int current = 0;  //This variable sets the dynamic starting position
   boolean sorted = false; //This variable keeps the sorting going
   T temp; //Not used
  while((current < (data.length))&&(sorted==false))
   {
      //Each iteration is one pass
      scan = (data.length - 1); //Sets traverser index to last index
      sorted = true;            //Sets sorted to true to keep the outer
                                  //loop going
      while(scan > current) //While traverser index > current position
      {
         //If last index data is smaller the data before last
          if (data[scan].compareTo(data[scan-1]) < 0)
         {
            sorted = false;//Set sorted to false
            swap(data, scan, (scan - 1));//Exchange the 2 data
         }
         scan--; //Keep traversing
      }
      current++; //Keep the loop going if sorted is false
   }
}
```

EX 14.1 Hand trace a queue X through the following operations:

```
X.enqueue( new Integer(4) );
X.enqueue( new Integer(1) );
Object Y = X.dequeue();
X.enqueue( new Integer(8) );
X.enqueue( new Integer(2) );
X.enqueue( new Integer(5) );
X.enqueue( new Integer(3) );
Object Y = X.dequeue();
X.enqueue( new Integer(4) );
X.enqueue( new Integer(9) );
```

One possible solution:

X.enqueue(new Integer(4));		4					
X.enqueue(new Integer(1));		1	4				
Object Y= X.dequeue();			1				
X.enqueue(new Integer(8));		8	1				
X.enqueue(new Integer(8));		2	8	1			
X.enqueue(new Integer(5));		5	2	8	1		
X.enqueue(new Integer(3));		3	5	2	8	1	
Object Y= X.dequeue();			3	5	2	8	
X.enqueue(new Integer(4));		4	3	5	2	8	
X.enqueue(new Integer(9));		9	4	3	5	2	8
		Rear					Front

EX 14.2 Given the queue X that results from Exercise 14.1, what would be the result of each of the following?

a) X.first();
b) Y = X.dequeue();
   X.first();
c) Y = X.dequeue();
d) X.first();

One possible solution:

X.first();	8
Y=X.dequeue();	8
X.first();	2
Y=X.dequeue();	2
X.first();	5

EX 14.9 Explain why the array implementation of a stack does not require elemnts to be shifted but the non circular array implementation of a queue does.

One possible solution:

Because stack is a data structure in which the first element in is the last element out, or first-in-last-out operation. Due to this, the push and pop operations only care about current position, or one end, of the element in the stack while the other end, index 0, is the last location of the operation

before the stack becomes empty. In contrast, non-circular array of implementation of a queue requires elements to be shifted to index 0 of the stack because the queue is a data structure in which the first element enters is the first element exits. The enqueuer operation pushes the element from the rear index while the dequeue operation pops the element from index 0 only. If no shifting of elements is performed, then the dequeuer operation can only be done once and the subsequent dequeuer will yield no value due to the empty content at index 0.

PP 14.9 Create a system using a stack and a queue to test whether a given string is a palindrome (that is, whether the characters read the same both forwards and backwards).

One possible approach:

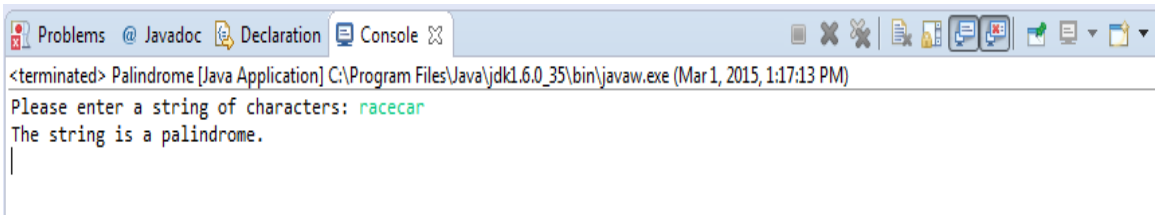
```
import java.util.*;
import java.io.*;

/**
 * Palindrome demonstrates the use of queue and stack to
 * determine whether a given string of characters is a palindrome.
 * CST200, Spring 2015
 * @author Hieu Pham
 * @version 4.0
 */
public class Palindrome
{
		@SuppressWarnings("unchecked")
		public static void main(String[] args)
		{
			Scanner iCh = new Scanner(System.in);//Keyboard input
			String ICH = "";
			char TestChar; //Use this to equate input string
			//char TC=0; //Use this to equate input string
			Scanner again = new Scanner(System.in);//Need this for
			messages and prompting
			String Prompting=""; //Need this for messages and
			prompting
			boolean confirmed = false; //Need to confirm palindrome
			boolean longstring = false;
			int a=0,b=0,c=0; //Need for computing in various places

			@SuppressWarnings("rawtypes")
			Stack CharStack = new Stack();
			@SuppressWarnings("rawtypes")
			Queue CharQueue = new LinkedList();
			//Declaring the stack and the queue here will force
			//them to be empty each pass through
```

```
System.out.print("Please enter a string of characters: ");
ICH=iCh.nextLine();//User enters a \n terminated string
here
b = ICH.length(); //Determine the length of user's string

if(b>16)
{
	System.out.println("string too long");
	longstring = true;
}

while(c < b) //Read the input string one character at a
time.
{
	TestChar=ICH.charAt(c); //Process one char at a time
	CharStack.push(TestChar); //Load the stack
	CharQueue.add(TestChar);  //Load the queue
	//A copy of ICH goes at the top of the stack and at
	the rear end
	//of the queue.
	c++;
}// end while
confirmed = true;
a = 1;
//Some attempt to guard the stack to avoid
//the emptystack situation while in processing
try
{
	//Repetitively compare the top element of the stack
	//and the front element of the queue
	if(CharStack.peek() == CharQueue.peek())
	{
		CharStack.pop();
		CharQueue.remove();
		a++;
	}
	else
	{
		//Change confirmed flag to false when
		//the comparison failed
		confirmed = false;
	}
}
catch(EmptyStackException e)
{
	System.out.println("empty structure");
}

//Processing the result...
if((confirmed == true)&&(!longstring))
{
	System.out.println("The string is a palindrome.");
}
else
{
```

```
                        System.out.println("The string is a Not
                        palindrome.");
                        if(!longstring)
                        {
                                System.out.println("The string is too
                                long.");
                        }
                }
        }
}
```

Output:

confirmed is true



confirmed is false