Question 1 [40 marks] Answer all eight parts.

(i) Write a fragment of Java code that writes out the numbers from one to 100 inclusive with ten numbers per line. (5 marks)

```
String numbers = "";
int count = 0;

for(int i = 1; i < 101; i++) {
        numbers += " " + i;
        count++;

        if(count == 10) {
                System.out.println(numbers);
                count = 0;
                numbers = "";
        }
}
```

(ii) Give a series of Java statements that do the following: (a) declare a variable named stk to refer to a stack (ADT Stack) of integer values; (b) create a stack (ArrayBasedStack) and make stk refer to it; (c) populate the stack by adding the numbers from one to ten inclusive; (d) remove and print the top three numbers from the stack. (5 marks)

```
Stack<Integer> stk = new ArrayBasedStack<Integer>();

for(int i = 1; i < 11; i++) {
        stk.push(i);
}

for(int j = 1; j < 4; j++) {
        System.out.println(stk.pop());
}
```

(iii) Give a complete pseudocode algorithm that takes a list object (ADT List) named lst containing integer elements and that deletes all odd numbers from the list. Your algorithm may manipulate lst by means of the operations of ADT List only. (5 marks)

```
Algorithm removeOdds(list)
        for(int i = 0; i < list.size(); i++) {
                if(list.get(i) % 2 != 0) {
                        list.remove(i);
                }
```

}

(iv) Describe succinctly, using words and/or diagrams as appropriate, how an array might be used to represent an object of type ADT Queue. Indicate clearly how operations enqueue and dequeue can be implemented efficiently. (5 marks)

- Answered in Summer 2011.

(v) Consider the pseudocode shown below of a simplified array-based implementation of ADT Stack. Translate this code into a detailed, compilable Javacode. You may assume, for now, that the stack items are of type Integer, but you must include appropriate instance/ class variable declarations, a constructor and implementations of the methods push and pop. (5 marks)

```
Representation:
  S: array of length N = 100
  t: integer variable

Algorithm push(o):
  t ←t+1
  S[t] ←o

Algorithm pop():
  e ←S[t]
  t ←t−1
  return e
```

(vi) Indicate briefly the modifications required to Part (v) to make the implementation generic i.e. capable of accommodating stack items of any type. (5 marks)

- Answered in Summer 2010.

(vii) Given below is a flawed version of a non-recursive version of the well-known binary search algorithm that contains a number of logical errors. Modify the code so that BinarySearch(S, k) returns the index within S that houses k, if k is present and so that it returns −1, if k is not present. (5 marks)

```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size()
    while low ≥high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k > midKey then
            high ←mid − 1
        else
            return −1
```

- Answered in Summer 2010.

(viii) Explain the concept of a comparator and the role this concept plays in the implementation of ADTs. (5 marks)

- Answered in Summer 2011.

Question 2 [20 marks]
Give a Java implementation of ADT List. Include both an interface List.java and an implementation ArrayBasedList.java. For full marks your interface/implementation must

- **be as complete as possible (though you may ignore the iterator and listIterator operations completely);**
- **include implementations for operations size, isEmpty, get, set, add (both variants) and remove;**
- **be based on the concept of a left-justified array;**
- **be generic i.e. capable of accommodating any element type.**

Partial marks will be awarded for sensible, coherent answers that meet some but not all of the requirements listed.

```
public interface List<EltType> extends java.lang.Iterable<EltType> {
        public int size();
```

```
        public boolean isEmpty();
        public EltType get(int inx);
        public EltType set(int inx, EltType newElt);
        public void add(EltType newElt);
        public void add(int inx, EltType newElt);
        public EltType remove(int inx);
}

public class ArrayBasedList<EltType> implements List<EltType> {

/* default capacity of the deque */
private static final int  INIT_CAP = 10;

/* maximum current  capacity of the deque */
private int capacity;

/* the deque elemets */
protected EltType elements[];

/* num of elemnts in deque */
protected int numElts;

public ArrayBasedList(int initCapacity) {
        numElts = 0;
        capacity =  initCapacity;
        elements = (EltType[]) (new Object[capacity]);
}

public ArrayBasedList() {
        this(INIT_CAP);
}

public int size() {
        return numElts;
}

public boolean isEmpty() {
        return (numElts == 0);
}

public EltType get(int inx) {
        if ((inx < 0) || (numElts-1 < inx)) {
                flagError("index out of bounds");
         }
         return elements[inx];
}

public EltType set(int inx, EltType newElt) {
        if ((inx < 0) || (numElts-1 < inx)) {
```

```
                flagError("index out of bounds");
        }

        EltType oldElt = elements[inx];
        elements[inx] = newElt;
        return oldElt;
}

public void add(EltType newElt) {
        expandIfNecessary();
        elements[numElts] = newElt;
        numElts++;
}

public void add(int inx, EltType newElt) {
        checkIndex(inx, size() + 1);
        expandIfNecessary();

        for (int i = numElts-1; i >= inx ; i--) {
                elements[i+1] = elements[i];
        }
        elements[inx] = newElt;
        numElts++;
}

public EltType remove(int inx) {
        checkIndex(inx, size());
        EltType retElt = elements[inx];

        for (int i = inx; i < numElts -1 ; i++) {
                elements[i] = elements[i+1];
        }
        numElts--;
        return retElt;
  }

public Iterator<EltType> iterator() {
        return new ABLIterator<EltType>(this);
 }

public ListIterator<EltType>  listIterator() {
        return new ABLListIterator<EltType>(this);
}

// helper methods
private void checkIndex(int inx, int highInx) {
        if ((inx < 0) || (inx >= highInx)) {
                 flagError("index out of bounds");
        }
}
```

```
private void expandIfNecessary() {
        if (size() == capacity)   {
                EltType temp[] =  (EltType[])(new Object[2*capacity]);

                for (int i = 0; i < capacity; i++) {
                        temp[i] = elements[i];
                }
                elements = temp;
                capacity = 2*capacity;
        }
}

private void flagError(String errmsg) {
        System.out.println("ArrayBasedList: "+errmsg);
        System.exit(1);
}

}
```

Question 3 [20 marks]
(i) Give an algorithm in pseudocode that takes two lists L1 and L2 of elements with elements arranged in increasing order from front to back within the list, and that merges the two lists into a single list L (in increasing order). (4 marks)

- Answered in Summer 2011.

(ii) Give a pseudocode implementation of Merge-sort based on the above that takes a list of elements and that rearranges those elements in increasing order. Your algorithm may either be recursive or nonrecursive. (4 marks)

- Answered in Summer 2011.

(iii) Sketch a proof that your Merge-sort algorithm does actually sort the elements of any list given as input. (3 marks)

- Answered in Summer 2011.

(iv) Analyze the worst-case running time of your Merge algorithm. Assume the running time is captured by the number of comparisons performed. Comment on the efficiency of Merge-sort in relation to other well-known sorting algorithms. (3 marks)

- Answered in Summer 2011.

(v) Suppose that we have an application that requires the manipulation of a large number of student records. For each student we have his names (first, middle, last), id number, date of birth among other data. To forestall possible exam-time confusion, the Registrar's Office wants a list of groups of students with similar names. Indicate how efficiently to produce a list of names (first name, last name combinations) shared by more than one student and for each such "shared name", generate a list of the details of those students that bear that name. (6 marks)

- ADT List stores student's details in an ADT set
- Set stores all student information, owned by specific student
- Create empty map where key will be student "shared name" and value will be ADT set of each students information
- Iterate through set, for each student
    - get first and last name
    - Check map for key with "shared name" combination
    - If present, add student to maps value (set)
    - If not, create new set with students name and add to map