Question 1 [40%]

(i) Write a fragment of Java using Map / ArrayBasedMap that declares and creates a map object named nums, populates the map with the numbers from 1 to 100 inclusive (as keys) and then removes all those numbers divisible by 17 (i.e. 17, 34, 51, …). (8%)

Answer (Not Answer but the Full MapArray Implementation)

```
import helpers.Comparator;
import helpers.IntegerComparator;
import helpers.MapEntry;



public class MapArray<KeyType, ValueType> implements Map<KeyType, ValueType> {

		protected static final int INIT_CAPACITY = 100;			/* the initial capacity of the map
	*/
	protected int capacity ;						/* the current capacity
of the map */
	protected int numEntries;
	protected MapEntry<KeyType, ValueType>[] entries;
	protected Comparator<KeyType> comparator;			/* comparator that defines equality between
keys */
	protected static final int NO_SUCH_KEY = -1;			/* value denoting unsuccessful search */


	public MapArray(Comparator<KeyType> myComparator) {
		entries = new MapEntry[INIT_CAPACITY];
		capacity = INIT_CAPACITY;
		comparator = myComparator;
		numEntries = 0;
	}

	public int size() {
		return numEntries;
	}

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

	public ValueType get(KeyType k) {
		int indexWithKey = findEntry(k);

		if (indexWithKey != NO_SUCH_KEY) {
			return (entries[indexWithKey]).getValue();
		}
		return null;
	}

	public ValueType put(KeyType k, ValueType e)  {
		MapEntry<KeyType, ValueType> newEntry = new MapEntry<KeyType, ValueType>(k, e);
		int indexWithKey = findEntry(k);

		if (indexWithKey != NO_SUCH_KEY) {
			ValueType oldVal = entries[indexWithKey].getValue();
			entries[indexWithKey] = newEntry;
			return oldVal;
		}
		else {
			expandIfNecessary();
			entries[numEntries++] = newEntry;
		}
		return null;
		}
```

```
public ValueType remove(KeyType k) {
        int indexWithKey = findEntry(k);

        if (indexWithKey != NO_SUCH_KEY) {
                MapEntry<KeyType, ValueType> removedEntry =
    entries[indexWithKey];
                entries[indexWithKey] = entries[numEntries-1];
                numEntries--;
                return removedEntry.getValue();
        }
        return null;
    }



/************************
* Helper Methods
************************/
private int findEntry(KeyType key) {
    for (int i = 0; i < numEntries; i++) {
        if ( comparator.compare(key, entries[i].getKey()) == 0 ) {
            return i;
        }
    }
    return NO_SUCH_KEY;
    }


protected void expandIfNecessary() {
    if (numEntries == capacity) {
        int newCapacity = 2*capacity;
        MapEntry<KeyType, ValueType>[] temp =  new MapEntry[newCapacity];

        for (int i = 0; i < capacity; i++) {
            temp[i] = entries[i];
        }
        entries = temp;
        capacity = newCapacity;
    }
    }


}
```

REMOVE NUMBERS DIVISIBLE BY 17 ???

```
public remove() {
    capacity = 100;
    int number = 1;


    while (number <= capacity) {
        if (number % 17 == 0) {
            nums.remove(number);
        }
    }
}
```

(ii) Give a pseudocode fragment that takes the contents of a queue (ADT Queue) and reverses the order of the items contained within it. You may make use of an additional

ADT from {Stack, Queue, List, Map}, if you wish. (11%)

Answer

Algorithm ReverseQueue(Q)

Input: queue Q
Output: queue Q in reverse order

S is an empty stack
while(!Q.isEmpty()) do
    S.push(Q.dequeue())

(iii) The incomplete non-recursive binary search shown is intended to return the index within array S that contains the search key k (or "index" -1 if the search key is not present). Complete the algorithm by providing appropriate pseudocode for the placeholders labelled [a] etc. (8%)

Answer

```
Algorithm BinarySearch(S, k)
low <- 0
high <- S. size () - 1
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
        low mid+1
return - 1
```

(iv) Consider an implementation of ADT List using a left-justified array represenattion based on the following instance variable declarations, where EltType stands for the element type of the items in the list.

```
private EltType elements[];
private int numElts;
```

Give a complete implementation for the operation remove. *(8%)*

Answer

```
public EltType remove(int numElts) {
        checkIndex(numElts);
        EltType retElt = entries[numElts];

        entries[numElts] = entries[numEntries-1];

        numEntries--;
        return retElt;
}
```

(v) State the number of comparisons completed during the execution of the following algorithm when applied to an array X of length n. Justify your reasoning carefully. (8%)

```
Algorithm BS(X, n):
  s ←1
  while s < n do
    curr = 0
    while curr < n−1 do
      next = curr + 1
      currElt = X[curr]
      nextElt = X[next]
      if  currElt > nextElt then
        X[curr] = nextElt
        X[next] = currElt
      curr ←curr + 1
    s ←s +1
```

Answer

Question 2 [30%]

Give a complete Java implementation from scratch for an enhanced version of the traditional queue ADT that includes all the usual ADT operations as well as the following:

delete(val): Remove from the queue all elements that equal the specified value; return the number of deletions made. Input: EltType; Output: int

Your implementation must respect the following conditions:

1. It must be based on the concept of a doubly-linked list and
2. It must be capable of housing elements of any (comparable) data type.

You do not need to provide code for an interface nor for a node class.

Answer

```
import helpers.LLNode;


public class QueueDLinkedList<EltType> implements Queue<EltType> {

   // Declarations
   private int size;
   private LLNode<EltType> head;
   private LLNode<EltType> tail;

   public QueueDLinkedList() {
            size = 0;
            head = new LLNode<EltType>(null, null, null);
            tail = new LLNode<EltType>(head, null, null);
            head.setNext(tail);
   }

   public int size() {
            return size;
   }

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

   public static void main(String args[]) {

                    QueueDLinkedList<Integer> map = new QueueDLinkedList<Integer>();

                    map.enqueue(1);
                    map.enqueue(2);
                    map.enqueue(3);
                    map.enqueue(4);
                    System.out.println("Front: " + map.front());
                    System.out.println(map.dequeue());

                    map.showMap();
            }

   public EltType front() {

            if (size == 0) {
                    flagError("illegal queue op");
            }

            LLNode<EltType> firstNode = head.getNext();
            return firstNode.getElement();
   }
```

```
  public void enqueue(EltType obj) {
        LLNode<EltType> oldLast = tail.getPrev();
        LLNode<EltType> newLast = new LLNode<EltType>(oldLast, tail, obj);

        oldLast.setNext(newLast);
        tail.setPrev(newLast);

        size++;
  }

  public EltType dequeue() {

        if (size == 0) {
              flagError("illegal queue op");
        }

        LLNode<EltType> oldFirst = head.getNext();
        //head = head.getNext();
        head.setNext(oldFirst.getNext());

        size--;
        return oldFirst.getElement();
  }

        /************************
          * Helper Methods
          ************************/
        public void showMap() {
              System.out.println("\n****Start Map Structure****");
              System.out.println("Headnext: " + head.getNext().getElement());

              // get starting node
              LLNode<EltType> currentElement = head.getNext();


              // loop through map
              for(int i = 0; i < size; i++) {

                    System.out.println("element at position " + i + ": value = " +
currentElement.getElement());

                    // go to next node
                    currentElement = currentElement.getNext();

              }

              System.out.println("Tailprev: " + tail.getPrev().getElement());
              System.out.println("****End Map Structure****");
        }


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


}
```

Question 3 [30%]

(i) The following recursive algorithm segregates the contents of A[f…r] (the segment of array A between indices f and r inclusive) so that all the values less than or equal to x appear to the left of those greater than x.

```
Algorithm Split(A, x,  f,  r):
  if r < f then
    return
  else
  if A[f] ≤x then
    Split(A, x, f+1, r)
  else
  if A[r] > x then
    Split(A, x, f, r−1)
  else
     temp = A[f]
     A[f] = A[r]
     A[r]= temp
     Split(A, x, f+1, r−1)
```

Draw a recursion tree to show the execution of Split (X, 5, 0, 7) where the array contains [3, 8, 4, 1, 6, 5, 2, 7] initially. Show the state of the array and the values of f and r at each stage. (6%)

Answer

(ii) Argue that for any array A of length n, the number of calls to Split arising from Split(A, x, 0, n-1) for any x is at most n + 1 (6%)

Answer

(iii) Show how the algorithm may be modified to produce a variant Split2 so that Split2(A, x, f, r) not only partitions the elements in the manner of Split but also returns the number of values in A[f…r] that are less than or equal to x. (9%)

Answer (Worth 3 / 9 as is)

```
Algorithm Split2(A,x, f, r)
    count <- 0
    if r < f then
        return
    else
    if A[f] <= x then
        Split (A, x, f+1, r)
        count++
    else
    if A[r] > x then
        Split (A, x, f, r-1)
    else
        temp = A[f]
        A[f] = A[r]
        Split (A, x, f+1, r-1)
```

(iv) Based on the Split2 algorithm, write a recursive sorting algorithm (in pseudocode) that takes an interval of an array A[f…r] and that re-arranges the contents so that they appear in increasing order left to right. (9%)

Answer