PRE-EXAMEN

Camilo Villalba
Oscar Díaz

1.

```
public static <T> void printIndexes(ArrayList<T> P, ArrayList<Integer> L){
      for( int i = 0; i < L.size(); i++){
            if(L.get(i) <= P.size()){
                  System.out.println(P.get(L.get(i)).toString());
            }
            else{
                  throw new IndexOutOfBoundsException( “out of bounds”);
            }
      }
}
```

b)complejidad n

2.

```
public static ArrayList<Integer> interseccion(ArrayList<Integer> L1, ArrayList<Integer> L2){
            ArrayList<Integer> inter = new ArrayList<Integer>();
            boolean cont = false;

            for(int i =0;i<L1.size();i++){
                  if(inter.size()==0){
                        cont=false;
                  }else{
                        for(int c=0;c<inter.size();c++){
                              if(inter.get(c)==L1.get(i)){
```

```
                        cont=true;
                    }else{}
                }
            }
            if(cont==false){
                inter.add(L1.get(i));
            }
        }
        cont=false;
        for(int i =0;i<L2.size();i++){
            if(inter.size()==0){
                cont=false;
            }else{
                for(int c=0;c<inter.size();c++){
                    if(inter.get(c)==L2.get(i)){
                        cont=true;
                    }
                    else{}
                }
            }
            if(cont==false){
                inter.add(L2.get(i));
            }
        }
        return inter;
}
```

3.

```
public static ArrayList<Integer> union(ArrayList<Integer> L1, ArrayList<Integer> L2){
	boolean cont = false;
	ArrayList<Integer> un = new ArrayList<Integer>();
	for(int i =0;i<L1.size();i++){
		if(un.size()==0){
			cont=false;
		}else{
			for(int c=0;c<un.size();c++){
				if(un.get(c)==L1.get(i)){
					cont=true;
				}else{}
			}
		}
		if(cont==false){
			un.add(L1.get(i));
		}
	}
	cont=false;
	for(int i =0;i<L2.size();i++){
		if(un.size()==0){
			cont=false;
		}else{
			for(int c=0;c<un.size();c++){
				if(un.get(c)==L2.get(i)){
					cont=true;
				}else{}
			}
		}
		if(cont==false){
			un.add(L2.get(i));
		}
	}
```

```
            return un;
      }
```

4. a) 1258346
    b)64521588
    c)

```
public static void dfs(int [][] G, int n){
            int size = G.length;
            int[] vL = new int[size]; //visitedList
            int[] pL = new int[size]; //predecessorsList
            for( int  i = 0; i < size; i++){
                  vL[i] = 0;
                  pL[i] = -1;
            }
            System.out.println(n);
            n = n-1;
            pL[n] = -2; // predecesor del nodo inicial para distinguirlo del resto
            dfs(G, n, vL, pL);
      }

      public static void dfs(int[][] G, int n, int[] vL, int[] pL) {
                  int i = 0;
                  int size = G.length-1;
                  // revisa si ya recorrio todo el grafo y esta nuevamente en lo posición inicial
                  if(pL[n] == -2 && visitedAllNodes(G,n,vL) == true){
                        return;
                  }
```

```
            // busca la primera conexión a un nodo no conocido
            while( (G[n][i] == 0 || (G[n][i] == 1 && vL[i] == 1)) && i < pL.length -1 ){
                  i++;
            }

            // revisa si el nodo no tiene mas conexiones con nodos desconocidos para regresar al nodo
            //predecesor
            if((G[n][size] == 0 || (G[n][size] == 1 && vL[size] == 1))&& i == pL.length-1){
                  vL[n] = 1;
                  dfs(G,pL[n],vL,pL);
            }else{
                  vL[n] = 1; // marca el nodo como visitado
                  pL[i] = n; // marca el preedcesor del nodo al q se dirige
                  System.out.println(i+1);
                  n = i;      //n sera el nuevo para realizar la busqueda
                  dfs(G,n,vL,pL);
            }
}

//retorna verdadero si todos los nodos adyacentes a "n" an sido visitados
public static boolean visitedAllNodes(int[][] G, int n, int[] vL)
{
      for(int i = 0; i < G.length; i++){
            if( G[n][i]  == 1 && G[n][i] != vL[i])
                  return false;
      }
      return true;
}
```

5.
Ventajas: no tiene que cambiar los apuntadores de los elementos cada vez que se elimina un nodo

```
class ChainNode<T>
      {
         // package visible fields
         T element;
         ChainNode<T> next;
         boolean isDeleted = false; // se añade un campo para decir si el nodo a sido borrado

         //
         public boolean isDeleted() {
                 return isDeleted;
         }

         public void setDeleted(boolean isDeleted) {
               this.isDeleted = isDeleted;
         }
         // package visible constructors
         ChainNode( )
         {
            this( null, null );
         }

         ChainNode( T element )
         {
            this( element, null );
         }

         ChainNode( T element, ChainNode<T> next )
         {
            this.element = element;
            this.next = next;
         }
```

```
	}

public class Chain<T> implements LinearList<T>, Iterable<T>
{
	// fields
	protected ChainNode<T> firstNode;
	protected int size;
	protected int deletedSize; // se añade un campo para contar la cantidad de elementos borrados


	// verifica si la cantidad de elementos borrados es igual a la de no borrados y en ese caso elimina
	//los no borrados
	void checkDeletedSize(){
			T removedElement;
			if(deletedSize == size/2){
				ChainNode<T> p = firstNode;
				for( int i  = 0; i < size; i++){
					if(p.isDeleted == true){
						if( i == 0 ) // remove first node
						{
							firstNode = firstNode.next;
							size--;
							i--;
						}else{
							ChainNode<T> q = firstNode;
						    for( int j = 0; j < i - 1; j++ )
							  q = q.next;

						    q.next = q.next.next; // remove desired node
						    size--;
						}
					}
				}
```

```
                    deletedSize = 0;
                }
}

public T remove( int index ){
            checkIndex( index );
            T removedElement;
            if( index == 0 ) // remove first node
            {
               removedElement = firstNode.element;
             firstNode.setDeleted(true); //marca el elemento como borrado
            }
            else
            {  // use q to get to predecessor of desired node
               ChainNode<T> q = firstNode;
               for( int i = 0; i < index - 1; i++ )
                  q = q.next;

               q.setDeleted(true);
               removedElement = q.next.element;
            }
            deletedSize++;
            checkDeletedSize();
            return removedElement;
}

public void add( int index, T theElement ) {
            if( index < 0 || index > size )
               // invalid list position
               throw new IndexOutOfBoundsException
                  ( "index = " + index + "  size = " + size );

            if( index == 0 )
```

```
                    // insert at front
                    firstNode = new ChainNode<T>( theElement, firstNode );
                 else
                 {   // find predecessor of new element
                    ChainNode<T> p = firstNode;
                    for( int i = 0; i < index - 1; i++ )
                       p = p.next;

                    // insert after p
                    p.next = new ChainNode<T>( theElement, p.next );
                 }
                 size++;
                 checkDeletedSize();
}
```

7.

```
public class LaptopList<T super Laptop extends Comparable >extends ArrayList<T> implements Iterable<T>
{
            public void sort(int start, int end, Comparator c){
                  int t = this.size();
                  int size = this.size();
                  T tmp;

                  for(int j = 0;j*2 < size;j++){
                        for(int i = 0;i < size; i++){
                              if(c.compare(this.get(i),this.get(i+1)) > 0){
                                    tmp= this.get(i);
                                    this.set(i, this.get(i+1));
                                    this.set(i+1, tmp);
                              }
                              if(c.compare(this.get(t),this.get(t-1)) < 0){
                                    tmp= this.get(t);
```

```
                                this.set(t, this.get(t-1));
                                this.set(t-1, tmp);
                           }
                           t--;
                     }
                     t = size;
               }
          }

          public Iterator<T> iterator (){
               return new latopListIterator( );
          }
     }


     Comparator c = new Comparator<Copmuter>() {
       public int compare(Comuter o1, Computer o2) {
           return o1.getRam() - o2.getRam();
       }
```

8.

```
public static void buscar(int entero1, int entero2){

             int operador=((diccionario.size()/entero2)+((diccionario.size()*entero1)/(entero2/2)));

             if (diccionario.get(operador)==palabra){
              System.out.println(diccionario.get(operador));
             }
             if(diccionario.get(operador)<palabra){
              buscar((entero1*2)+1,entero2*2);
             }
             if(diccionario.get(operador)>palabra){
```

```
                buscar((entero1*2),entero2*2);
               }
}
complejidad = (log n)3+1
```