**Data Structures**

**First exam’s key**

**First semester 2016-2017**

**Q1:**

1.

 A) Unordered List B) Queue C) Heap D) Binary Search Tree

2.



**Q2:**

A)

 List<Integer> L1 = new List<Integer>();

 Scanner sc = new Scanner(System.in);

 for(int i=0;i<20;i++){

 L1.add(sc.nextInt());}

B)

Int total=0

Integer num = L2.first();

 while(num != null){

 total+=num;

 num = L2.next();}

System.out.println(total);

**Q3:**

Solution:

1) The **interface** of operations that is supported by a data structure is one factor to consider when choosing between several available data structures.

2) How much **space** does the data structure occupy?

3) What are the **running times** of the operation in its interface?

Example:

* + We are looking to “buy” the best implementation of a stack.
		- StackA. Does not provide a getSize operation.
			* i.e. there is not single operation that a client can use to get the number of entries in StackA.
		- StackB. Provides a getSize operation, implemented in the manner we discussed earlier, transferring entries back and forth between two stacks.
		- StackC. Provides a getSize operation, implemented as follows: a variable called size is maintained that is incremented every time an entry is pushed, and decremented every time an entry is popped.

Three situations:

* 1. Need to maintain a large number stacks, with no need to find the number of entries.
	2. Need to maintain only one stack, with frequent need to find the number of entries.
	3. Need to maintain a large number of stacks. With infrequent need to find the number of entries.

In situation a) StackA fits the bill.

Situation b) StackC .

Situation c) presents a choice between StackB and StackC.

**Q4:**

public void delete (Car myCar)

 {

 for (int i = 0; i<cars.numItems; i++)

 {

 if (myCar.brand.equals(cars.list[i].brand) && myCar.year==cars.list[i].year)

 {

 int loc=i;

 cars.list[loc]=cars.list[cars.numItems];

 cars.numItems--;

 }

 }

**Q5:**

Public boolean Search(T item)

{

 int loc = 0;

 boolean found = false;

 while(!found && loc < numItems)

 {

 if(list[loc] == item)

 found=true;

 else

 loc++

 }

 return found:

}