Akbota Toxanbayeva
at3017

1.

```
import java.awt.List;
 Public static void printLots (List<Integer>L, List<Integer>P)
         Public void add(int a , int b) {
             Print a+b;
}
           for (Integer a : P) // Loop over the list
           {
               System.out.println(L.get(a));
            }

        }

}
```

2.

```
List intersect(list L1, list L2){
    List result;
    Position L1pos= 0, L2pos= 0, resultPos = 0;
    while( L1pos <L1.size() && L2pos <L2.size()){

          if( L1.get(L1pos)< L2.get(L2pos))
            L1pos = L1 pos++;

          else if(L1.get(L1pos) > L2.get(L2pos) )
            L2pos = L2pos++;

          else{
        result.insert(resultPos, (L1.get(L1pos))
                 L1pos ++;
                L2pos ++;
                resultPos ++;
           }
   }
 return result;
}
```

3.Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used. Provide java-like pseudocode

```
     public Class Stack {
   private static object [] array = null;
   private static int stack_number = 0;
   private int stack_id =0
   private int s1;
   private int s2;
public Stack(int size) {
            s1 =0;
            s2= size-1;
  stack_number++;
  stack_id =stack_number;
            if (array==null) {
    array = object[size];
  }
}
  public void push (object element)throws Exception {

     if (this.stack_id ==1) {
         array[s1] =element;
         s1++;

     }
     else{
        array[s2] =element;
         s2--;
     }
     if (s1 == s2)
     {
        throw new Exception("Both stacks are full" );
     }
  }
  public object pop () throws Exception{
     Object element = null;
     if (this.stack_id ==1) {

         element =array[s1] ;
         s1--;
     }
     else{
        element =array[s2] ;
         s2++;
     }
     if (s1 <0)
     {
      throw new Exception("No element in stack 1" );
    }
    if (s2 >=size)
    {
      throw new Exception("No element in stack 2" );
    }
  }
}
```

Algorithm

```
1. Allocate array
2. Two indexes
        s1 = 0
        s2 = end of array
3. Push s1 -> s1 + 1
4. Push s2 -> s2 - 1
5. Check for outflow
         s1 = s1 +1
         s1 - s2 ?
        If equal don't write throw stack overflow
6. If s1 < 0 or s2 -> length stack underflow
```

4.

a)

1. Push 4 to to holding track S1
2. Push 3 to to holding track S1
3. Enqueue 1 to the output track
4. Push 8 to holding track S2
5. Enqueue 2 to the output track
6. Pop 3 to the output track from holding track S1
7. Pop 4 to the output track from holding track S1
8. Push 7 to holding track S2
9. Push 6 to holding track S2
10. Push 9 to holding track S3
11. Enqueue 5 to the output track
12. Pop 6 to the output track from holding track S2
13. Pop 7 to the output track from holding track S2
14. Pop 8 to the output track from holding track S2
15. Pop 9 to the output track from holding track S3

b)

1 9 6 7 2 8 5 3 4