1. Consider the following code:

```
public <T extends Comparable<T> > int mistery1 (T[] v)
{
   int cont=0 ;
   for (int i = 0; i < v.length; i++) {
      cont++ ;
      boolean sing = true ;
      for (int j = i+1; j < v.length; j++)
          if (v[i].compareTo(v[j]) == 0)
             sing = false ;
      if (!sing)
          cont-- ;
    }
    return cont ;
}
```

a) Explain the result of the code presented above and presents the result applied to the vector `a[8]={6,1,4,1,7,3,1,7}` and `mystery1(a)`.

b) Analyze temporal complexity following Big-Oh notation. Justify.

2. Consider the following method to process an **ordered sequence** of numbers:

```
public boolean mystery2 (int[] A, int value)
{
   boolean flag=false;
   for (int i = 0; i < (A.length-1); i++)
      for (int j = i+1; j < A.length; j++)
        if (A[i]+A[j] == value) {
           flag = true;
           System.out.println("pos "+ i +"->"+A[i]+", pos "+j+"->"+A[j]);
        }
   return flag;
}
```

a) Explain what the code presented above do and present the result applied to the vector `a[7]={1,13,17,18,22,33,35,38}` and `mystery2(a,35)`.

b) Validate if the mystery method is deterministic or non-deterministic and analyze temporal complexity following Big-Oh notation. Justify.

c) Propose a more efficient solution.

3. Consider the following code and validate if the method is deterministic or non-deterministic and analyse temporal complexity following Big-Oh notation. Justify.

```
public double power (double b, int e){

        if (e == 0)
         return 1;
        if (e == 1)
         return b;
        if (e % 2 == 0)
            return power (b*b, e/2);
        else
            return b*power(b*b, e/2) ;
}
```

Complementary Exercises

1. Consider the following code:

```
public static boolean mistery3(int[] data, int low, int high) {
    if (low >= high) return true;
    else if (!mistery3(data,low,high-1)) return false;
    else if (!mistery3(data, low+1, high)) return false;
    else return (data[low] != data[high]);
}
```

a) Explain what the method does.

b) Validate if mystery3 method is deterministic or non-deterministic and analyze its temporal complexity following Big-Oh notation. Justify.

c) Propose a more efficient solution.

2. Consider the following code

```
int binarySum (int[ ] data, int low, int high) {
    if (low > high)
       return 0;
    else if (low == high)
       return dat[low];
    else {
       int mid = (low + high) / 2;
       return binarySum(data, low, mid) + binarySum(data, mid+1, high);
    }
}
```

a) Validate if `binarySum()` method is deterministic or non-deterministic and analyse its temporal complexity following Big-Oh notation. Justify.