1. Consider the following method to process an **ordered sequence** of numbers **not repeated**:

```
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 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.

2. Consider the following code:

```
public Map<Pessoa, Set<CartaoCredito>> anComplex (HashMap<CartaoCredito, Pessoa> mc){

    Map<Pessoa, Set<CartaoCredito>> mp = new TreeMap<>();

    for (Map.Entry<CartaoCredito, Pessoa> mccp : mc.entrySet()) {
        CartaoCredito cc = mccp.getKey();
        Pessoa p = mccp.getValue();
        Set<CartaoCredito> scc = mp.get(p);
        if (scc == null) {
            scc = new HashSet<>();
            mp.put(p,scc);
        }
        scc.add(cc);
    }
    return mp;
}
```

a) Explain what the method does.

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

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 recursive function:

```
public static void exemplo (Integer[] a, int n) {
    int x = n/2;
    if (n > 0){
       exemplo(a,x);
       for (int i=0; i<n; i++)
          a[i] *=2;
    }
}
```

a) Analyse the example method and present the result of the method applied to the vector a: 1, 1, 1, 1, 1, 1 with the following example invocation (a, a.length).

b) Analyse the algorithm for its temporal complexity, using the Big-Oh notation. Justify.

2. Consider the following code

```
int binarySum (int[ ] data, int low, int high) {
    if (low > high)
       return 0;
    else if (low == high)
       return data[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.