Q1.

In case of Binary Search the sub problem is finding the index of a search item on a sub array.

In case of finbonacci series the sub problem is finding the fib series of a smaller number.

Binary Search

```
Algorithm binSearchA, x, lower, upper)
Input. Already sorted array A of size n, value x to be searched for in array section
A[lower]..A[upper]
Output: true or false

if lower > upper then return false
mid <-- (upper + lower)/2
if x = A[mid] then return true
if x < A[mid] then
     return binSearch(A, x, lower, mid - 1)
else
    return binSearchA, x, mid + 1, upper)
```

Fib series

```
Algorithm fib(n)
Input: a natural number n
Output: F(n).
if (n = 0 || n = 1) then return n
return fib(n-1) + fib(n-2)
```

Lets do binary search on [3,4,5,6,7,8,8,9], find 8

If we want to perform a search for item 8. we divide the array into smaller sub arrays and depending on the mid value we decide whether to go with left sub problem or right sub problem. The sub problems never repeat themselves in binary search.

Mid = 7 < 8 so we go with right subproblem

Arr = [8,8,9]

Mid = 8 == 8. so the we find the value.

Lets calculate fib(5)

Fib(5) -> fib(4) + fib (3)

Fib(4) -> fib(3) + fib(2)

Fib(3) -> fib(2) + fib(1)

Here when we divide the fib(5) into sub problems fib(4) and fib(3), on further sub division, we can observe the calculations or subproblems repeating itself.

As we can see the subproblems do not repeat themselves in binary search but they do in case of fibonacci series. This is the overlapping of the subproblems.

Q2.

D	“”	“k”	“ka”	“kal”	“kale”
“”	0	1	2	3	4
“m”	1	1	2	3	4
“ma”	2	2	1	2	3
“map”	3	3	2	2	3
“mapl”	4	4	3	2	3
“maple”	5	5	4	3	2

Q3.

```
class Solution {
private int climbStairs(int[] table, int[] options, int target){
if(target==0){
return 1;
}
if(target < 0){
return 0;
}
if(table[target]==-3){
int count = 0;
for(int option:options){
count += climbStairs(table,options,target-option);
}
table[target] = count;
}
return table[target];
}
public int climbStairs(int n) {
int[] table= new int[n+1];
Arrays.fill(table,-3);
return climbStairs(table,new int[]{1,2},n);
}
}
```

Q4.

```
class Solution {
public static int longestCommonSubsequence(String text1, String text2) {
int[][] memo = new int[text1.length()+1][text2.length()+1];

for(int i=1;i<memo.length;i++){
for(int j=1;j<memo[i].length;j++){
if(text1.charAt(i-1) == text2.charAt(j-1)){
memo[i][j] = 1 + memo[i-1][j-1];
}else{
memo[i][j] = Math.max(memo[i-1][j],memo[i][j-1]);
}
}
}

return memo[text1.length()][text2.length()];
}
}
```