# Prob1:

**BinarySearch**:

* Subproblems: BinarySearch(A, x, lower, mid-1) & BinarySearch(A, x, mid+1, upper)
* Input array is divided into 2 separate hafts, each subproblem handle each haft

**Fibonacci**:

* Subproblems: fib(n-1), fib(n-2)
* fib(n-1) will have 2 subproblems fib(n-2) & fib(n-3), fib(n-2) also have fib(n-3)

# Prob2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | 3 |

# Prob3:

//Recursive

static int sub(String s1, String s2) {

if (s1.length() == 0 || s2.length() == 0)

return 0;

if (s1.charAt(0) == s2.charAt(0))

return 1 + sub(s1.substring(1), s2.substring(1));

return Math.max(sub(s1.substring(1), s2), sub(s1, s2.substring(1)));

}

//Dynamic Programming Solution

static int subD(String s1, String s2) {

int l1 = s1.length(), l2 = s2.length();

int[][] D = new int[l1 + 1][l2 + 1];

for (int i = 1; i <= l1; i++) {

for (int j = 1; j <= l2; j++) {

if (s1.charAt(i - 1) == s2.charAt(j - 1))

D[i][j] = 1 + D[i - 1][j - 1];

else

D[i][j] = Math.max(D[i - 1][j], D[i][j - 1]);

}

}

return D[l1][l2];

}