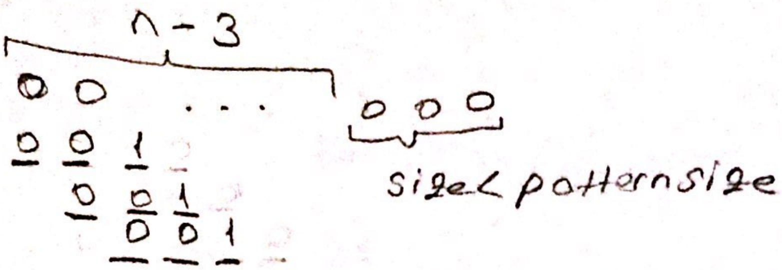
1)



There are three comprasions every time in this algorithm untill text's size is n-3

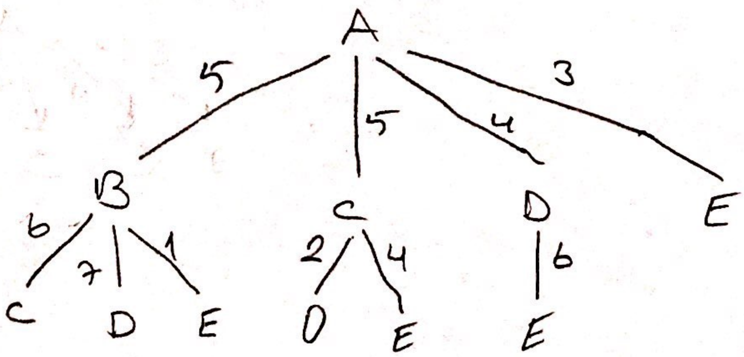
$3 \times (n-3) = 3n-9$ comprasions.

worst case $\Rightarrow m(n-m+1) \rightarrow 3(n-3+1)$

$= 3(n-2)$

$\boxed{\Rightarrow \Theta(n)}$

2)



→ Brute force algorithm computes the path length for every possibilities find the shortest path.

$\frac{(5-1)!}{2} = 12$ possibilities.

shortest

A-B-C-D-E-A	22	A-B-E-C-D-A	16	A-C-D-B-E-A	18
A-B-C-E-D-A	25	A-B-E-D-C-A	19	A-C-E-B-D-A	21
A-B-D-C-E-A	21	A-C-B-D-E-A	27	A-D-B-C-E-A	24
A-B-D-E-C-A	27	A-C-B-E-D-A	22	A-D-C-B-E-A	16

3)

```
function fun(n):
    if (n <= 1)
        return 0
    else
        return fun(n/2)+1
```

The recurrence relation for the number of additions.

$$T(n) = T(n/2) + 1, \quad n > 1, \quad T(1) = 0$$

$$T(n) \in \Theta(\log_2 n)$$

4)

Among n identical looking coins one is fake with a balance scale, we can compare any 2 sets of bottles.

→ Divide n bottles to 2 piles of $n/2$ bottles each leaving 1 extra bottle aside if n is odd.

→ continue dividing by 2

Resulting recursion formula $w(n) = w(n/2) + 1$ for $n > 1$

so, $\boxed{w(n) = \log n}$ $\quad w(1) = 0$

⊕ This recursion is identical to the one for the worst case number of comparisons in binary search.

The interesting point here is that this algorithm is not the most efficient solution. It would be more efficient to divide the bottles into 3 piles of about $n/3$ bottles each.

5) We compare the middle elements of arrays arr1 and arr2 let us call these indices mid1 and mid2 respectively.

Let us assume arr1[mid1] k, then clearly the elements after mid2 cannot be the required element. We then set the last element of arr2 to be arr2[mid2].

In this way, we define a new subproblem with half the size of one of the arrays.

```
def fun (arr1, arr2, end1, end2, k):
    if (arr1 == end1)
        return arr2[k]
    if (arr2 == end2)
        return arr1[k]

    mid1 = (end1 - arr1)/2
    mid2 = (end2 - arr2)/2

    if (mid1 + mid2 < k)
        if (arr1[mid1] > arr2[mid2])
            return fun (arr1, arr2 + mid2 + 1, end1, end2,
                        k - mid2 - 1)
        else
            return fun(arr1 + mid1 + 1, arr2, end1, end2,
                        k - mid1 - 1)
    if (arr1[mid1] > arr2[mid2])
        return fun (arr1, arr2, arr1 + mid1, end2, k)
    else
        return fun (arr1, arr2, end1, arr2 + mid2, k)
```

CamScanner ile tarandı

Note that

In the above code, k is 0 indexed which means if we want a k that's 1 indexed, we have to subtract 1 when passing it to the function.

Worst case time complexity

$O(\log n + \log m)$