1710440010
Dilara Karakoş

CSE 321 - HW 4

QUESTION 1:

The time complexity of brute force is O(n.m), which is sometimes written as O(n*m). So, if we were to search for a string of "n" characters in a string of "m" characters using brute force it would take us "n*m" tries.

Text : ooo ... o
n Lenght

Patten oo1o

1) ooooooo ... oo
oo1o. (3 comparison)
2) ooooooo ... oo
oo1o... (3 com)

3) ooooooo. ... oo
ooo1o (3 com)

n) ooo.... ooooooo
oo1o

Total Number of Comparisons : 3(N-3) = 3N-9
O(N)//

Worst case : oo1, Because, 3 comparison are maden when it find a matech.

QUESTION 2:

We need to find the one that has min all hamiltonian circuits and find cost.

AEDBCA : 27
AEDCBA : 22
AECBDA : 24
AECDBA : 21
AEBDCA : 18
AEBCDA : 16
ADBCEA : 24
ADBECA : 21
ADCBEA : 16
ADCEBA : 16
ADECBA : 25
ADEBCA : 22
ABDCEA : 21
ABDECA : 27
ABCDEA : 22
ABCEDA : 25
ABEDCA : 18

ABECDA : 16
ACDBEA : 18
ACDEBA : 18
ACEDBA : 27
ACEBDA : 21
ACBEDA : 22
ACBDEA : 27

PATHS: AEBCDA
ADCBEA
ADCEBA
ABECDA

Min Cost : 16

QUESTION 3 :

```
def log(n):
    if n==1
        return 0;
    else
        return log(n/2) +1;
```

$A(n) = A(n/2) + 1$

From Master Theorem;

$a=1 \quad a=b^d$
$b=2 \quad 1=2^0$
$d=0 \quad 1=1$

$\Big\} \; O(\log n)$ //

QUESTION 4:

I thought of solving this problem by iteration. So I mean, I checked each bottle one-by-one to see if the weight of the bottle was wrong.

```
procedure wrongbottle( bottles)
    while( 1 -> n bottles)
        if (bottles[i] incorrect)
            return bottles[i];
        end
    end
end
```

Best case : we can find the bottle in first iteration.

$B(n) = O(1)$

worst case : we can find the bottle in last iteration.

$W(n) = O(n)$

Average :

$A(n) = \sum_{i=1}^{n} i \cdot P(i)$ probably $\exists$ x = ith bottle is $\frac{p}{n}$

$A(n) = \sum_{i=1}^{n} i \cdot \frac{p}{n} = \frac{p}{n} \frac{n(n-1)}{2} = \frac{pn-p}{2} \Rightarrow O(n)$ //

QUESTION 5:

```
procedure find_kth ( arr1[0:n], arr2[0:m], k)
{
    if ( n==0)
        return  arr2[k];
    end if
    if (m==0)
        return  arr2[k];
    end if

    mid1 = n/2;
    mid2 = m/2;
    if ( mid1+mid2 <k)
        if ( arr1[mid1] > arr2[mid2])
            return  find_kth ( arr1, arr2[mid2+1 :]
                    , k-mid2-1);
        else
            return  find_kth ( arr1[mid1+1 :], arr2,
                    k-mid1-1);
        end if
    else
        if (arr1[mid1] > arr2[mid2])
            return  find_kth (arr1[:mid1], arr2, k)
        else
            return  find_kth ( arr1, arr2[:mid2], k)
        end if
    end if

end
}
```

I compare the middle elements of arrays array 1 and array 2. I said that these indices are mid 1 and mid 2. Assume that arr1[mid1] is equal to k, then clearly the elements after mid2 cannot be the required element. Then, set the last element of arr2 to be arr2[mid2].

Let us assume that arrays are A and B. The inputs are right. k's range is [0, len(A) + len(B)]. If lenght of one of the arrays is 0, the answer is k^{th} element of the second array. This is the base case of divide and conquer algorithm.

→ If mid element of A (mid index of B) is less than k

→ If mid element of A is greater than mid element of b, we can ignore the first half of b; else ignore the first half of A, adjust k

→ Else if k is less than sum of mid indices of A and B

→ If mid element of A is greater than mid element of B, we can safely ignore second half of A; else we can ignore second half of B.

Then the worst case is $\in O(\log m + \log n)$

m → A's length

n → B's length.