Emirhan UZUN
171064019

1. Number of zeros: n
Number of searching character = m = 4

The string for search is 0010 and text is 0. 00

000000000 ... 0
001
001
001

In every time, there would be 3. search. Because the first 2 character is same and algorithm checks for third character. So total number of comparisons is:

# of comp = Search word size x Loop counter
= (M-1) x (n-M+1)
= 3 x (n-3)
= 3n-9 comparison

[(m-1) → Size is m but we compare m-1 character because of 00|1|0.]

[(n-m+1), for instance if text size is 4, and search size is 3, there would be (4-3+1) shifting.]

for input pattern length is 3, (m=3)

# of comparison is M(n-M+1)

$T_{worst}(n) = O(n.m) = O(n.3) = O(3n) = O(n)$

Emirhan UZUN
171044019

2 - There are 5 vertices. I pick the start point is A so we have to check $\frac{(5-1)!}{2} = 12$ circuit. Now I write all of them and then I'll find the minimum cost.

ABCDEA = 5+6+2+6+3 = 22
ABCEDA = 5+6+4+6+4 = 25
ABDCEA = 5+7+2+4+3 = 21
ABDECA = 5+7+6+4+5 = 27
ABECDA = 5+1+4+2+4 = 16
ABEDCA = 5+1+6+2+5 = 19
ACBDEA = 5+6+7+6+3 = 27
ACBEDA = 5+6+1+6+4 = 22
ACDBEA = 5+2+7+1+3 = 18
ACEBDA = 5+4+1+7+4 = 21
ADBCEA = 4+7+6+4+3 = 24
ADCBEA = 4+2+6+1+3 = 16

A → B : 5
A → C : 5
A → D : 4
A → E : 3
B → C : 6
B → D : 7
B → E : 1
C → D : 2
C → E : 4
D → E : 6

The minimum cost ways are
1 : A → B → E → C → D → A
2 : A → D → C → B → E → A

Both of them take 16 cost.

(I write costs for minimum ways. In homework file, the costs exist but it is more comfortable I think. Also 12 circuit is different. I check all of them and actually there are 24 circuit but half of them are same. So 12 ways remain)

Emirhan UZUN
171044019

3. Let's say $\log_2 n = x$ so $2^x = n$
If we want to $\log_2 n$, then we divide then n to 2 by recursively

My code is in file.

Pseudocode

```
calculate (n)
    if n=1
        return 0
    endif
    else
        return 1 + calculate (n/2)
end
```

If the $n = 1$ then $T(n)$ is 0.

So $$T(n) = \begin{cases} 0 & , n=1 \\ 1+T(\frac{n}{2}) & , n>1 \end{cases}$$

Using master theorem,

$$T(n) = T\left(\frac{n}{2}\right) + 1 \quad , \quad a=1 \quad b=2 \quad c=0$$

$$\log_b a = \log_2 1 = 0 = c$$

So $$T(n) = \Theta(n^c \cdot \log n) = \Theta(\log n)$$

Emirhan UZUN
171044019

4. This question is similar to fake coin problem in our notebook.
With a balance scale, we can compare any 3 sets of bottles

↳ Divide n bottles into 3 piles of $\lfloor \frac{n}{3} \rfloor$ bottles each, leaving 1 extra bottle aside if n is odd.

If n=1, the bottle is the incorrect. and return it as a incorrect.
If n=2, compare them and call the algorithm recursively on the not equal bottle
If n>2, and n is multiple of 3, then divide the bottles into 3 piles of $\frac{n}{3}$ bottles each

If n>2 and n mod 3 = 1 or n mod 3 = 2, then divide the bottles into the piles of sizes $\lfloor \frac{n}{3} \rfloor$, $\lfloor \frac{n}{3} \rfloor$ and $n-2\lfloor \frac{n}{3} \rfloor$ respectively

```
ALGORITHM FindIncorrectBottle (List, n),   (n=size)
    If n=1 then
        return List (which contains only incorrect bottle)
    If n=2 then
        Compare the two bottles and call this function on the incorrect bottle.
    Else
        Divide bottles into 3 piles of ⌊n/3⌋, ⌊n/3⌋ and n-2⌊n/3⌋ bottles
        If the first two piles have same weight
            Discard them and call this method recursively on the third pile
        Else
            Call the this method recursively on the lighter pile of the first two piles.
```

Best Case: If the size of bottles is 1, then function takes constant time which is $O(1)$

Worst & Average Case Size divided by 3 in every time. So the equation is $T(n) = T(n/3) + 1$ $T(1) = 1$. We can say $n = 3^k$ and $T(3^k) = k$ and $T(n) = \log_3 n$. I think worst and average case is equal because both of them continues recursively in else part. So worst and avg. is $O(\log_3 n)$

Emirhan UZUN
17/06/2019

5. Pseudocode for findElement (L1, L2, L1length, L2length, target, st1, st2)

```
if st1 is equal to L1 length do
    return L2[st2+target-1]
if st2 is equal to L2 length do
    return L1[st1+target-1]
if target is 0 or target is bigger than (L1length-st1)+(L2length-st2) do
    return -1 (not found)
if target is 1 do
    if L1[st1] is smaller than L2[st2] do
        return L1[st1]
    else return L2[st2]

temp = target / 2
if temp-1 is greater or equal than L1length - st1 do
    if L1[L1length-1] is smaller than L2[st2+temp-1] do
        return L2[st2 + (target-(L1length-st1)-1)]
    else return findElement(L1, L2, L1length, L2length, target-temp, st1, st2+temp)
if temp-1 is greater or equal than L2length-st2 do
    if L2[L2length-1] is smaller than L1[st1+temp-1] do
        return L1[st1 + (target-(L2length-st2)-1]
    else return findElement(L1, L2, L1length, L2length, target-temp, st1+temp, st2)
else
    if L1[temp+st1-1] is smaller than L2[temp+st2-1] do
        return findElement(L1, L2, L1length, L2length, target-temp, st1+temp, st2)
    else return findElement(L1, L2, L1length, L2length, target-temp, st1, st2+temp)
end
```

Pseudocode for helperFunction (L1, L2, target)

```
L1.sort()
L2.sort()
findElement(L1, L2, L1length, L2length, target)
```

Emirhan UZUN
171044019

In this algorithm I have 2 functions. The first one is helper function. This sorts the arrays before, then send to the other function The point which I sort arrays first is that merging arrays first and then find x^{th} element is forbidden. Now the time complexity of helper function is $O(n,m) = O(max(nlogn, mlogm))$. n means size of array 1 and m is size of array 2. Python sort' method takes nlogn time for size n. According to these algorithm I wrote nlogn or mlogm.

The other function is findElement This function takes arrays which are sorted and then find the x^{th} element if it exists Because x may be out of range. If it can't find, return -1. This function recursive and size halved in every call. So $T(n) = T(\frac{n}{2}) + 1$ and $T(n) = O(logn)$.

We take the maximum of nlogn, mlogm or logn. So the total time complexity of algorithm;

if $n > m$ $O(nlogn)$

if $m > n$ $O(mlogm)$

Because both of them is bigger than logn.