CSE 321 Homework-4

Muhammed Yasir
Fidan
161044056

1-) Because of our text contains n zeros, pattern 0010 always done 3 comparisons, because third bit(1) will fail after comprison and bruteforce algorithm don't check last 0 bit. And this operation continues for n-3 times

For example if n=5 and we check 0010 pattern

n=5 → 00000 00000 Total 6 times compare
0010 0010
3 time compare 3 time compare

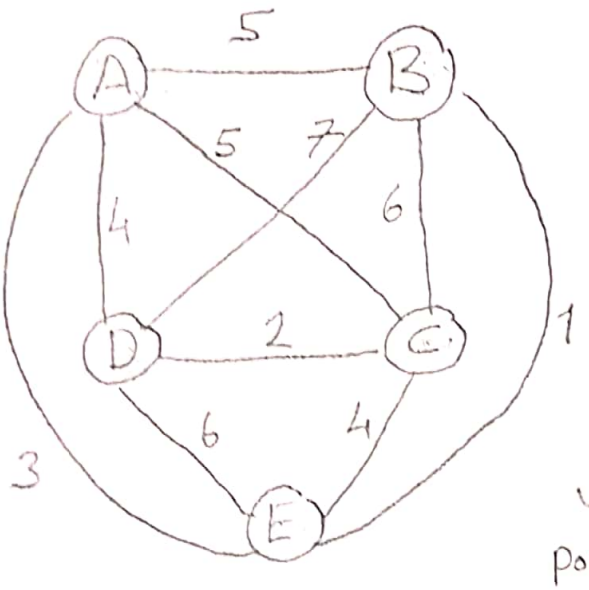
So, we compare 3 bit each n-3 times, general formula

$3.(n-3)$ compare

what is the worst case input pattern of length 3(3bits) for BF?

For brute force algorithm, worst case 3 bit pattern is "001" because its always do 3 comprison and last comparison will fail because of 1.

2-) Apply brute-force algorithm for the travelling salesmen problem.



In travelling salesmen problem, we have to find minimum cost travelling route that visit every node once and return starting one.

by using brute-force algorithm we can do this by checking every possible route and found min. one.

Brute force all possible routes: Lets start from A because it doesn't matter

We don't need write reverse routes too, because they have same cost.

A B C D E A → 22

A C B D E A → 27

A D B C E A → 25

A B D C E A → 21

A B C E D A → 25

A B E C D A → 16 ←

A B E D C A → 19

A B D E C A → 25

A C B E D A → 22

A C E B D A → 21

A C D B E A → 18

A D C B E A → 16 ←

So, minimum route has 16 cost

3) Design a decrease-by-half algorithm for computing $\log n$ (base 2) Calculate its time efficiency.

```
function log(n):
    if (n < 2):
        return 0;
    else
        return 1 + log(⌊n/2⌋);
```

Because we divide half every iteration, this algorithm has $O(\log n)$ time complexity.

4-) This is same as fake coin problem that we saw in class. for solving this by a decrease-and-conquer algorithm, we can divide bottles 2 part and check their weight. If one of them weight is wrong, it means incorrect bottle in these part. If bottle count is an odd number just select a bottle and divide other bottles 2 part again. If those 2 part have same weight, it means selected bottle is wrong one. Otherwise wrong bottle is in part that has incorrect weight.

In best case, if bottle number is odd and after selecting one bottle and divide other bottles 2 part and those part balanced, its mean selecting bottle is wrong one. This take $O(1)$ time complexity.

But, in worst case and average case because we every time divide 2 part this algorithm take $O(\log n)$ time

5-) Merging arrays first is forbidden, so we can sort both array firstly with quick sort or merge sort algorith. After sorting 2 array we can use this algorithm for finding x^{th} element.

```
def Algorithm(arr1, arr2, x)
    if(len(arr1) == 0): return arr2[k]
    if(len(arr2) == 0): return arr1[k]
    mid_1 = len(arr1) /2
    mid_2 = len(arr2) /2
    if((mid_1 + mid_2) < x):
        if(arr1[mid_1] > arr2[mid_2]):
            return Algorithm(arr1, arr2[mid_2+1:], k-mid_2-1)
        else:
            return Algorithm(arr1[mid_1+1:], arr2, k-mid_1-1)
    else:
        if arr1[mid_1] > arr2[mid_2]:
            return Algorithm(arr1[:mid_1], arr2, k)
        else:
            return Algorithm(arr1, arr2[:mid_2], k)
end
```

In this algorithm, I take middle indices of arr1 and arr2, Lets assume arr1[mid_1] x, then clearly the elements after mid_2 cannot be the required element. Then, we set the last element of arr2 to arr2[mid_2] In this way we reduce the problem size to half of the arrays so complexity is $O(\log(m)+\log(n))$ in worst case. But sorting part Complexity $O(n \log n)$