CSE 321 HW4 SOLUTIONS

171044098
Akif Kartal

1) Since, our pattern "0010" is length of 4 in each step, algorithm will do $\boxed{n-4+1 = n-3}$ comparison in worst case. Therefore total number of comparison will be (in terms of n);

$$\boxed{3.(n-3) = 3n-9}$$

* Here, in our pattern "0010" contains first 2 zero then one therefore 3 is coming from here.

- Worst case input of length 3 bits

In this case according to brute force algorithm worst case input will be $\boxed{001}$.

---

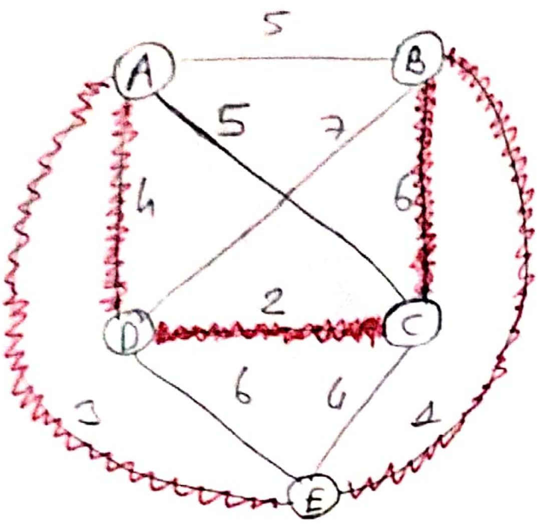
2) Brute-Force algorithm for the travelling salesman problem is following;

1) Make a list of all possible hamilton circuits.

2) Calculate the hamilton circuit by adding up the weights of its edges.

3) Choose the hamilton circuit with the smallest total weight.

* If we apply these steps following graph;



In this graph since there is a lot of hamilton circuits I will not be showing each of them instead I will be showing directly, hamilton circuit that has smallest total weight.

Brute Force Algorithm

Start Vertex = A

1. smallest path is A to E therefore select E = 3
2. smallest path is E to B therefore select B = 1
3. smallest path is B to C therefore select C = 6
4. // // // C to D // // D = 2
5. // // // D to A // // A = 4

Shortest hamilton circuit = A-E-B-C-D-A	Total = 16

3) This question require a decrease-by-half algorithm therefore to solve this question we need to find a relationship between $\log_2(n)$ and $\log_2(\frac{n}{2})$.

If we observe $\log_2(\frac{n}{2})$ statement we will see that.

by logarithm property; $\boxed{\log_a(\frac{b}{c}) = \log_a b - \log_a c}$

Thus, $\log_2(\frac{n}{2}) = \log_2 n - \log_2 2 = \boxed{\log_2 n - 1}$

Scanned with CamScanner

We found a relation which is;

$\log_2\left(\frac{n}{2}\right) = \log_2 n - 1$ therefore,

$\boxed{\log_2 n = \log\left(\frac{n}{2}\right) + 1}$ also $\boxed{\log_2 1 = 0}$

if we use this relations in our algorithm;

```
procedure computeLogBase2(n)
    if(n == 1)
T(1)=0 <—— return 0   // log_2 1 = 0
    else
T(n/2)+1 <—— return computeLogBase2(n/2) + 1
    end if
end
```

Recurrence relation from algorithm is;

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

by using master theorem our time efficiency is;

$f(n) = \Theta(1) \Rightarrow \Theta(n^0 \log^0 n) \Rightarrow \boxed{k=0}$

$a = 1$, $b = 2 \Rightarrow \log_b a = \boxed{\log_2 1 = 0}$

$\log_b a = k \Rightarrow 0 = 0$ therefore, (case 2)

$T(n) = \Theta(n^k \log^{p+1} n) = \Theta(n^0 \log^{0+1} n)$

$= \boxed{\Theta(\log_2 n)}$

Scanned with CamScanner

4) Question says "the weight of one of the bottles is set incorrectly" to find this bottle we will use the factory scale such that firstly, we will separate the bottles into two group. Then, by using factory scale we will scale them and which side is heavier or lighter then we will contiune with this side by applying same process. But the missing part here is what if the number of bottles is odd? In this case we will choose one of them and we will scale rest two group if their weights are equal then, the incorrect bottle is chosen bottle we don't need to scale anymore.

Analysis of algorithm.

When we observe this algorithm, we will see that this algorithm very similar to the binary search algorithm.

For example, if number of bottle is odd and incorrect bottle is in the middle we can directly find it in constant time as in binary search. Therefore from algorithm our recurrence relation is;

$$\boxed{T(n) = T(n/2) + 1 \quad , T(1) = 0}$$

We solved this relation in previous question our complexity will be $\boxed{T(n) = \Theta(\log_2 n)}$

Best Case $= T_{best}(n) = \Theta(1)$ // if bottle is in middle and number of bottles is odd.

Worst Case $= T_{worst}(n) = \Theta(\log_2 n)$

Average Case $= T_{avg}(n) = \Theta(\log_2 n)$

Scanned with CamScanner

5) In order to apply a divide and conquer algorithm we need to sorted arrays. Therefore firstly, we need to sort these two array. I will use merge sort to sort these array since it has a good performance on sorting. I will just use merge sort directly in my algorithm. (m and n are sizes of arrays)

```
procedure findxHelper(arr1, arr2, m, n, x)

    merge-sort(arr1, m) → O(mlogm)
    merge-sort(arr2, n) → O(nlogn)
    return findx(arr1, arr2, m, n, x) → O(logm+logn)
end
```

This algorithm first sorts the arrays then calls our main divide and conquer algorithm.

Divide and Conquer algorithm

In the next divide and conquer algorithm we will assume that x is valid such that x is between 1 and m+n which means that index of first element is 1.
In this algorithm first we will decide which array will contiune on to search according to their current size.
By this way we will be dividing our problem. After that, if array1 current index is 0 then we will return x^{th} element of array2. Lastly, if x is 1 we simply return minumum of the first element of array1 and array2.
After these controls we will call our function again by dividing problem size.

```
procedure findx(arr1, arr2, m, n, x)
    if (m > n) // decide which array will contiune
        return findx(arr2, arr1, n, m, x)
    end if
    if (m==0) // return xth element of array2
        return arr2[x-1]
    end if
    if (x ==1) // return minumum of first elements
        return min(arr1[0], arr2[0])
    end if     // set up new indexes
    i = min(m, x/2) // minumum of m and x/2
    j = min(n, x/2) // minumum of n and x/2

    if( arr1[i-1] > arr2[j-1]) // divide and conquer

        return findx(arr1, arr2[j:n], m, n-j, x-j)
    else
        return findx(arr1[i:m], arr2, m-i, n, x-i)
    end if
end
```

Worst Case analysis of algorithm

In this algorithm worst case happens if search operation doesn't terminate until last element of both array. In this case array1 will be divided logm times and array2 will be divided logn times.

Thus, $\boxed{T(m,n) = O(\log m + \log n)}$

Total time complexity of algorithm

In worst case total we have ;

$$\boxed{T(n,m) = O(m\log m) + O(n\log n) + O(\log m + \log n)}$$