Hands-on-4

DAA

1002219407
Praveen Anand

Problem 0:

Implement the fibonacci sequence

```
x= fib(n)
  if n==0
    return 0
  if n== 1
    return 1
  return fib(n-1) + fib(n-2)
```

* To get fib(5)

The first recursive call is fib(4)

The second recursive call is fib(3)

* For fib(4) which is recieved from fib(5)

The first rec. call is fib(3)

The second rec. call is fib(2)

* fib(3).

The first rec. call is fib(2)

The second rec. call is fib(1)

* fib(2)

The first rec. call is ~~fib(1)~~ fib(1)

The second call is fib(0)

* for fib(1) - It is taken from fib(2) -

fib(1) returns 1.

* fib(0) also derived from fib(2)

fib(0) will return 0.

Hence, fib(1) returns 1

fib(0) returns 0.

fib(2) = fib(0) + fib(1)

fib(2) = 0+1 = 1

⇒ fib(3) = fib(2) + fib(1)

= 1+1 = 2

fib(4) = fib(3) + fib(2)

= 2 + 1 = 3

fib(5) = fib(4) + fib(3) = 3+2 = 5.

Time Complexity of fibonnacci:

The time complexity is $O(2^n)$ and it is exponential. This is exponential because it recursively generates and calls two or more recursive calls. Since it is exponential it is inefficient for larger values of n.

Ways to improving fibonacci:-

Memoization:-

A method to optimize the algorithm is by using Memoization. which remembers already calculated values. and this avoids redundant computations and reduces time complexity to $O(n)$.

Iterative Approach:-

In this method fibonacci numbers are computed using a loop, than recursion. This gives $O(n)$ time complexity with $O(1)$ space complexity.

Problem-1

Time Complexity:-

To take elements and store it in a single array, we can assume time complexity of $O(N*k)$, then have sorted the array based on merge sort. Total over all time complexity is $O(NK) + O(Nk\log n)$ $= O(Nk\log(Nk))$.

Improved Approach:

Since each array is sorted could have used a min heap that would return smallest element in constat time.

Since the arrays is sorted we can take one element from each arrays and compare the first element of the array. After finding the smallest element. we must take next elemet from the same array & then compare again.

Worst case time complexity is $O(Nk(\log k))$.

Problem - 2

Time Complexity:-

Array is traversal is done once so time complexity is $O(n)$.

Improved Approach:

L) This itself is the efficient approach as both time & space complexity is efficient.