Date: / /
Page:

Day - 81

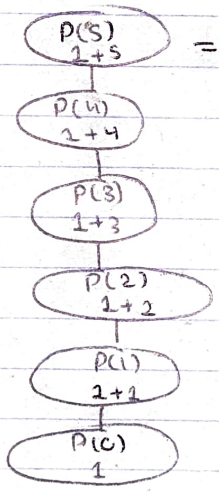
Recursion - 8

* **Time Complexity**

⇒ Total time taken by an algo to run as a function of i/p size.

Ex:

```
void print(int n){
    if(n==0)          ⟶ ① time for this
        return;
    cout << n << " ";  ⟶ ① time for this
    print(n-1);
}
```



= 6 = n+1

So, O(n)

⇒ So, Print(n) = cout << n , Print(n-1)

[T(n) = 1 + T(n-1)]

Now,
= 1 + 1 + T(n-2)
= 2 + T(n-2)
= 2 + 1 + T(n-3)
= 3 + T(n-3)

n - k = 0
when k = n

Date: / /
Page:

⇒ So,

$= n + T(n-n)$

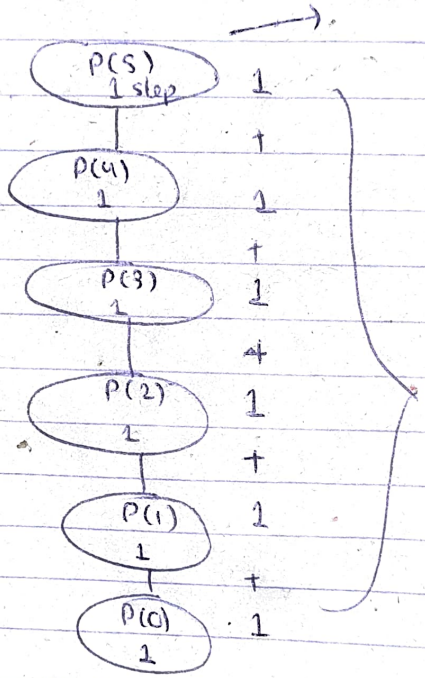
$= n + T(0) = n+1$

$= n$

$T(n) = O(n)$

⇒ This is the substitution method.

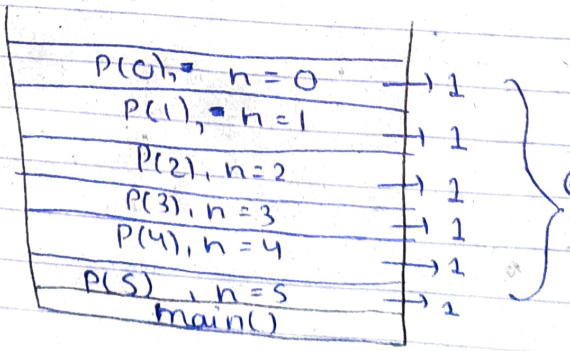
⇒ Another method is — Recursive Tree Method



$C = n+1$

$= O(n)$

Space Complexity



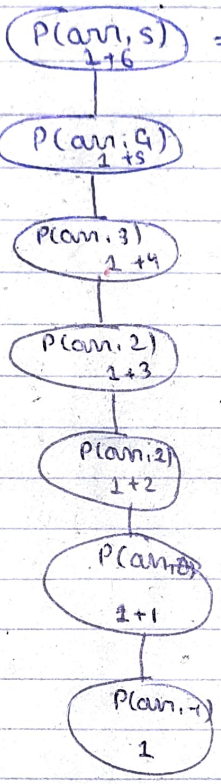
$C = n+1 = O(n)$

⇒ $O(n)$

Date: / /
Page:

Here, this is a pointer not an array.

```
void print(int arr[], int index){
    if(index == -1)
        return;
    cout << arr[index];
    print(arr, index-1);
}
```



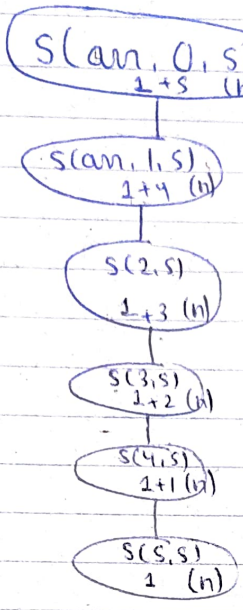
= 7 =

So, if 'n' is the length of the array then —
$T(n) = n+1 = O(n)$.

Also, S.C. is $\longrightarrow O(n)$.

n 1 1 → Space

```
int sum(vector<int> arr, int index, int n){
    if(index == n)
        return 0;
    return arr[index] + sum(arr, index+1, n);
}
```



$= 6 = n+1$

$T(n) = n+1$

$T.C. = O(n)$

So, Space is $6n$

=) So, space taken in one call is $n+2 \Rightarrow n$.

=) So, total space taken is $6n$.
(Here, we can't ignore 6 because 6 is dependent on n).

=) So, in terms of n —

$(n+1) * n$

$n^2 + n$ x

$O(n^2)$

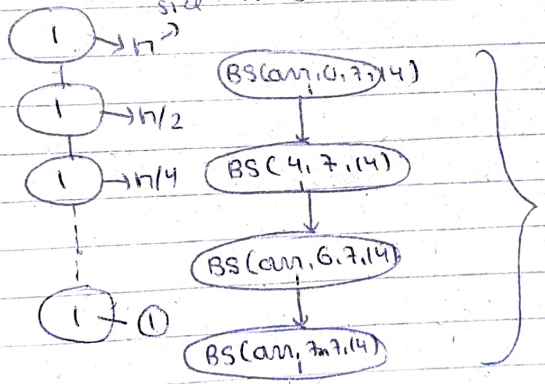
=) If we use reference variable then it will take constant space as (i.e. 4 or 8 bytes).

```
bool BinarySearch (int arr[], int start, int end,
                   int x) {
    if (start >= end)                          ①
        return 0;
    int mid = start + (end - start)/2;         ①
    if (arr[mid] == x)                         ①
        return 1;
    else if (arr[mid] < x)                     ①
        return BinarySearch(arr, mid+1, end, x);
    else                                       ①
        return BinarySearch(arr, start+1, mid-1, x);
```

3

←	0	1	2	3	4	5	6	7	→
start	1	2	4	6	8	10	12	14	end

$n = 8$, $x = 14$



We have to consider worst case.

So, $n \quad n/2 \quad n/4 \quad n/8 \quad \ldots \quad 1$

$1 + 1 + 1 + 1 \quad 1$

$\left(\frac{n}{2^0}\right) \left(\frac{n}{2^1}\right) \left(\frac{n}{2^2}\right) \left(\frac{n}{2^3}\right) \ldots \frac{n}{2^k}$

$1 \quad 2 \quad 3 \quad 4 \quad k+1$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k \log 2$

$k = \log_2 n$

So, Total time = k+1

$= \log_2 n + 1$

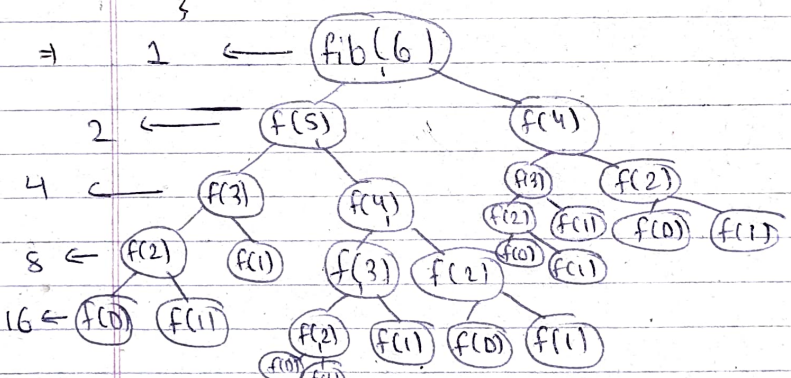
$= \log_2 n$

T.C. = $O(\log_2 n)$

⇒ Also, S.C. is $O(\log_2 n)$

*
```
int fib (int n){
    if (n <= 1)
        return n;
    return fib(n-1) + fib (n-2);
}
```

⇒



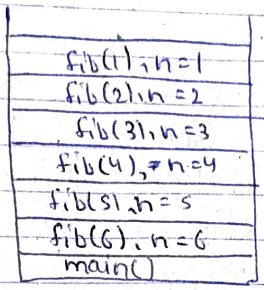
⇒ $1 + 2 + 4 + 8 + 16 + \ldots$

$(2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^{n-1})$

T.C. → $(2^n - 1)$ ⇒ $O(2^n)$

⇒ Space complexity is the ~~max~~ height of the tree.

⇒ So, S.C. = 6 = $O(n)$



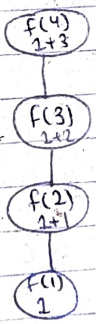
Maximum memory used is 6
i.e. n = 6

⇒ That is why space complexity is equal to the height of the tree.

Homework

1. int fact(int n){
① if(n<=1)
return 1;
return n*fact(n-1);
}

⇒



T.C. = O(n)
S.C. = O(n).

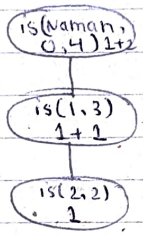
2. int power(int base, int exponent){
if(exponent == 0)
return 1;
return base * power(base, exponent - 1);
}

⇒ T.C. = O(exponent)
S.C. = O(exponent).

Date: / /
Page:

3.
```
bool isPalindrome (string str, int start, int end){
    if(start >=end){
        return true;
    return (str[start == str[end]) &&
            isPalindrome (str, start+1, end-1);
}
```

=1 T.C. = O(n)
S.C. = O(n)



4.
```
void reverseString (string& str, int start, int end){
    if (start < end){
        swap(str[start], str[end]);
        reverseString (str, start+1, end-1);
    }
}
```

=1 T.C. = O(n)
S.C. = O(n)

5.
```
bool isEven (int n){
    if(n==0)
        return true;
    return ! isEven (n-1);
}
```

T.C. = O(n)
S.C. = O(n)