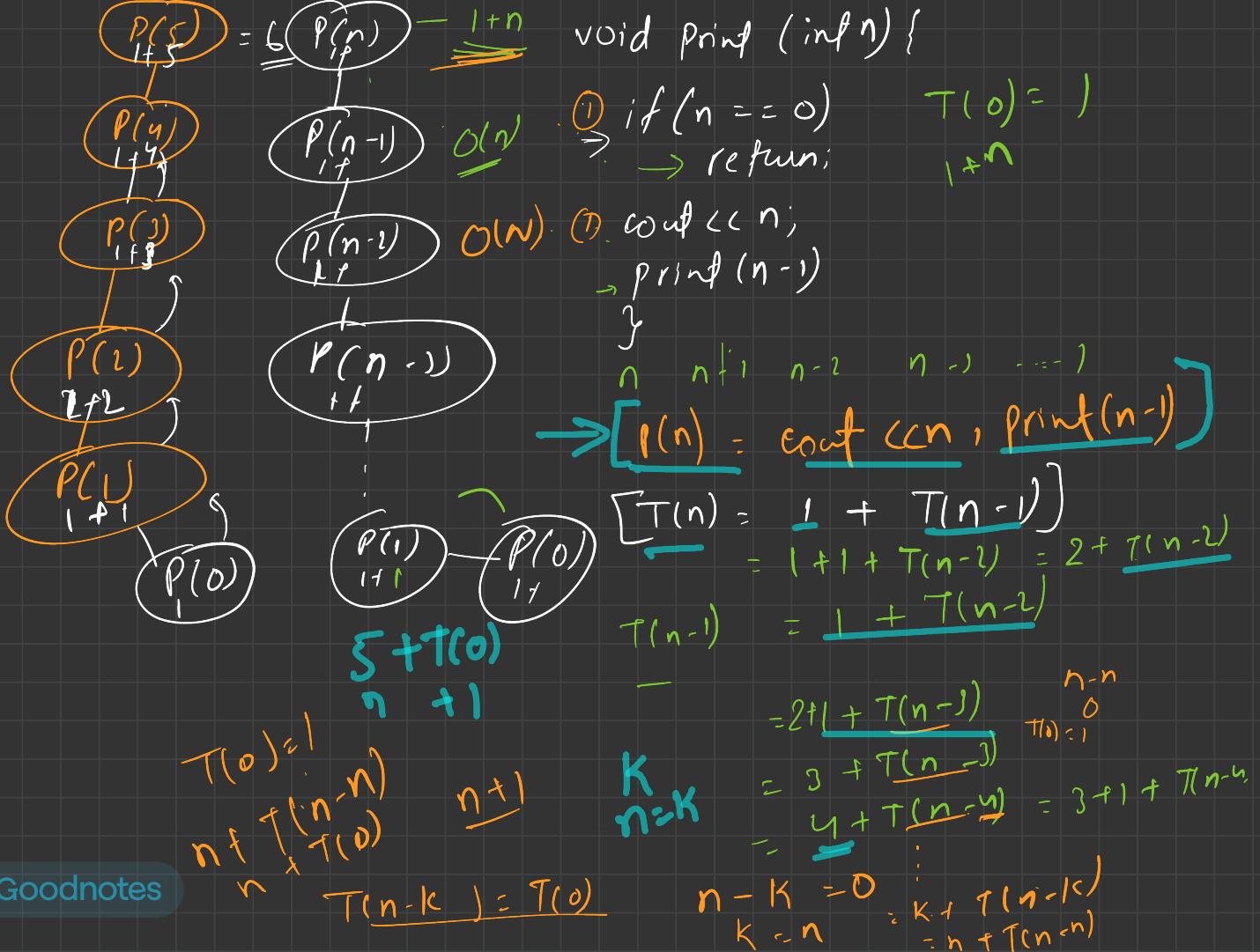
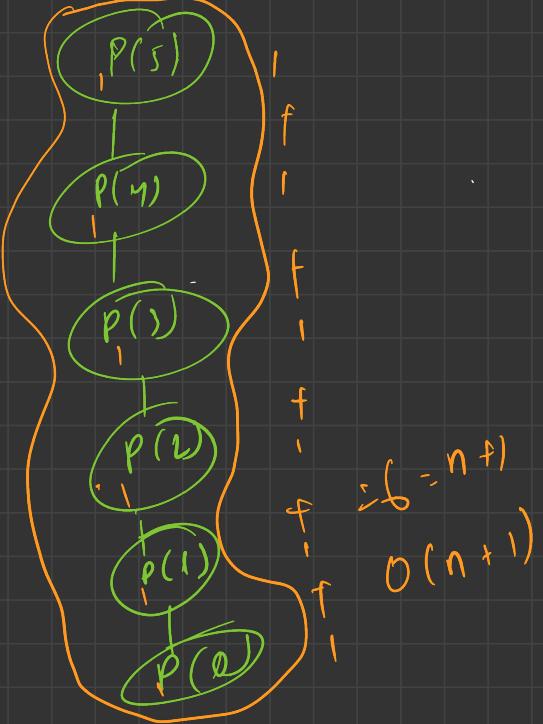


## Time Complexity

\* Total time taken by an algorithm to run as a function of its input size

for (i=0 ; i < n ; if f) {  
    cout << i;  
}  
  
 $\lambda = O(1)$   
 $|+| + |+| + |+| + |+|$   
  
 $\xrightarrow{\text{if } f} 3n$        $T = O(n)$   
  
 $n=5$   
 $O(n)$





```

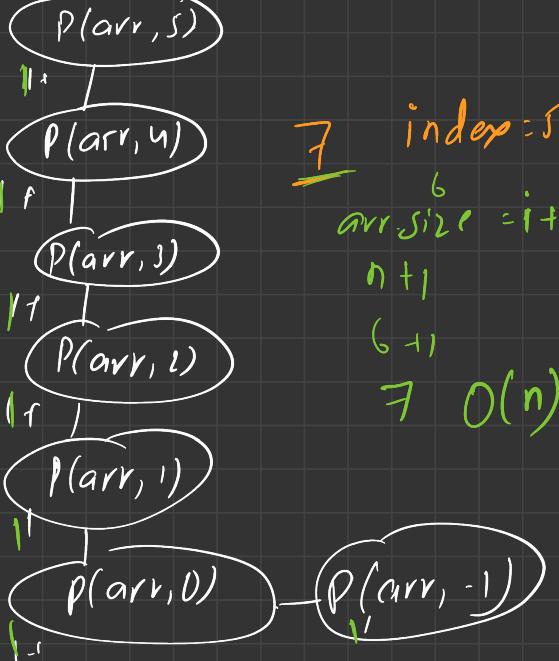
void P(int n)
→ if(n == 0)
   ↗ return
→ cout << n;
→ P(n - 1);
    
```

```

int main() {
    → P(5)
    } 
```

$\inf \text{arr}[]$   
 $\text{int } *\text{arr}$

~~4 byte~~  
~~80~~  $\mu$   $n \leq 11$   
 $i = 6$   
 $i = \text{arr.size} - 1$   
 $5 \leftarrow \text{arr.size}$   
 $6 \leftarrow n$



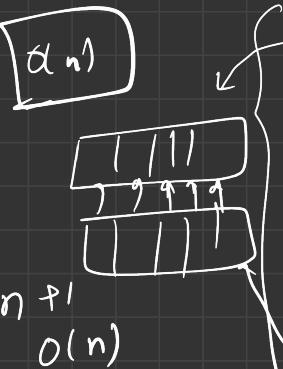
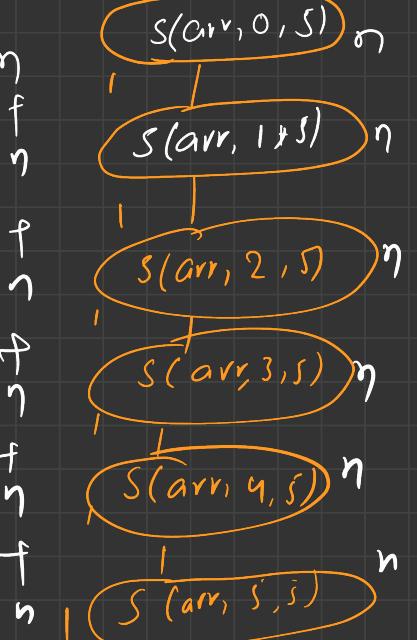
$\exists \quad \text{index} = 5$   
 $\text{arr.size} = 6$   
 $n + 1$   
 $(6 - 1)$   
 $\exists \quad O(n)$

`void P(int arr[], int i){`  
`if (i == -1)`  
`return;`  
`cout << arr[i];`  
`P(arr, i - 1);`

↑  $n$

`}`  
`index`  
`[1 | 2 | 3 | 4 | 5 | 6]`  
`int main(){`  
`int arr[] = {1, 2, 3, 4, 5};`  
`P(arr[], 5);`  
`}`

$O(N)$   
 $(n^2) \times n$   
 $\frac{n^2 \times n}{n} = O(n^2)$

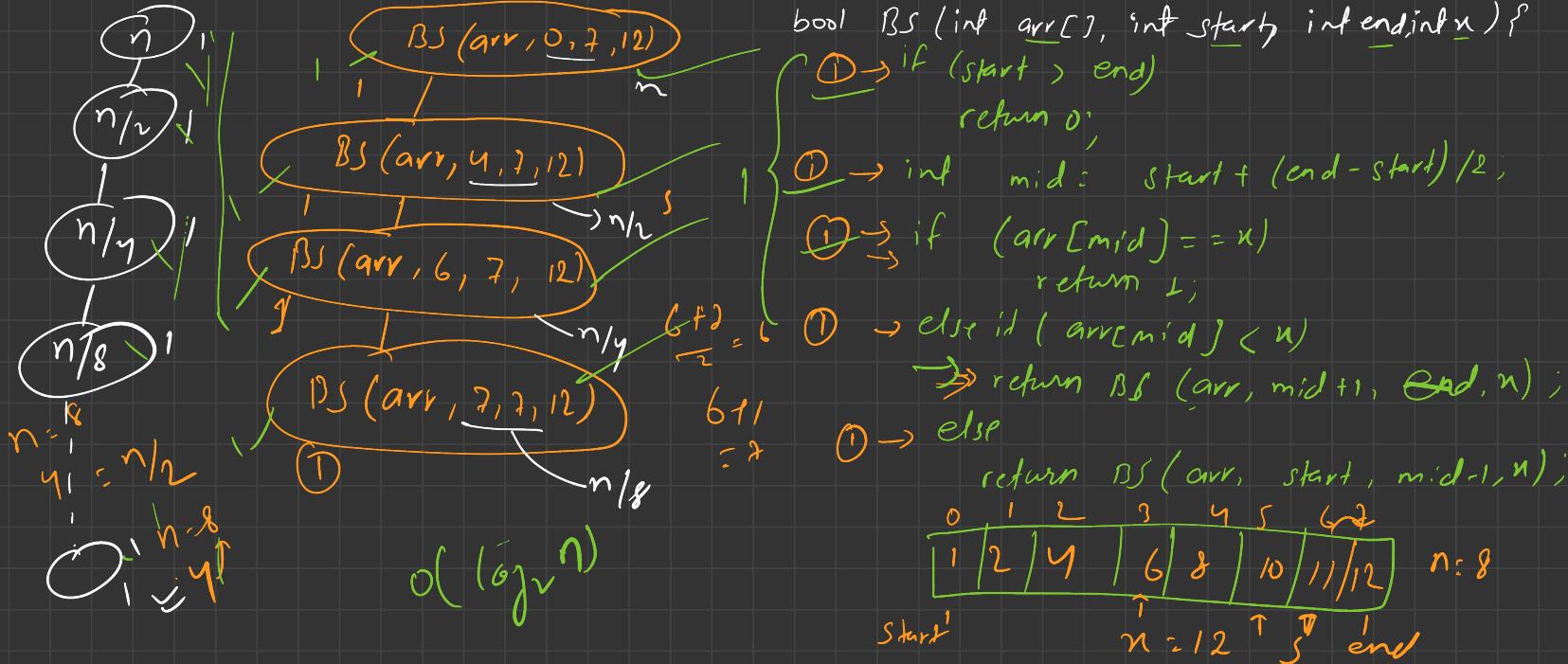


```

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

int main() {
    int n;
    cin >> n;
    vector<int> v(n);
    cout << sum (arr, 0, n);
}
  
```

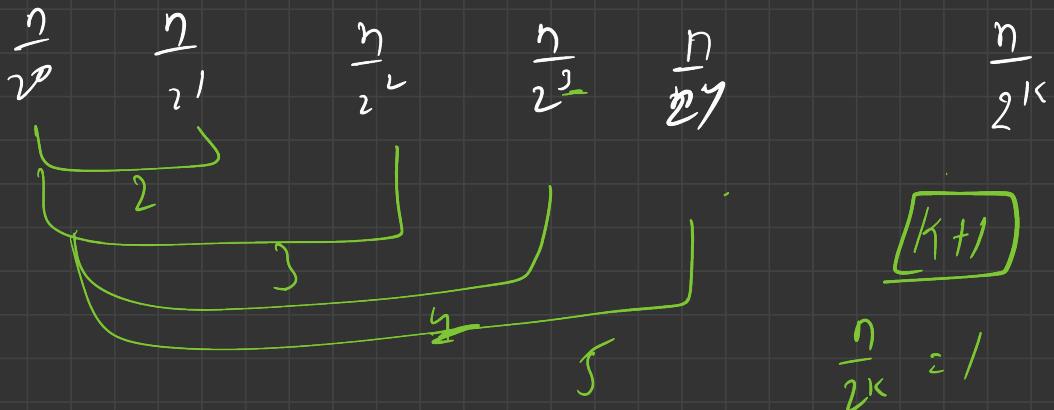
Diagram illustrating the recursive calls for the  $sum$  function. The code defines a recursive function  $sum$  that calculates the sum of elements in a vector  $arr$  from index  $i$  to  $n$ . If  $i$  equals  $n$ , it returns 0. Otherwise, it returns the current element  $arr[i]$  plus the sum of the remaining elements. The main function reads an integer  $n$  and initializes a vector  $v$  of size  $n$ . It then prints the result of calling  $sum(v, 0, n)$ .



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

Sub +

$$\begin{matrix} n \\ / & \backslash \\ f & + & f & + & f & + & f & + & \dots & f \\ \frac{n}{2^0} & \frac{n}{2^1} & \frac{n}{2^2} & \frac{n}{2^3} & \frac{n}{2^4} & \frac{n}{2^5} & \frac{n}{2^6} & \frac{n}{2^7} & \dots & \frac{n}{2^k} \end{matrix}$$

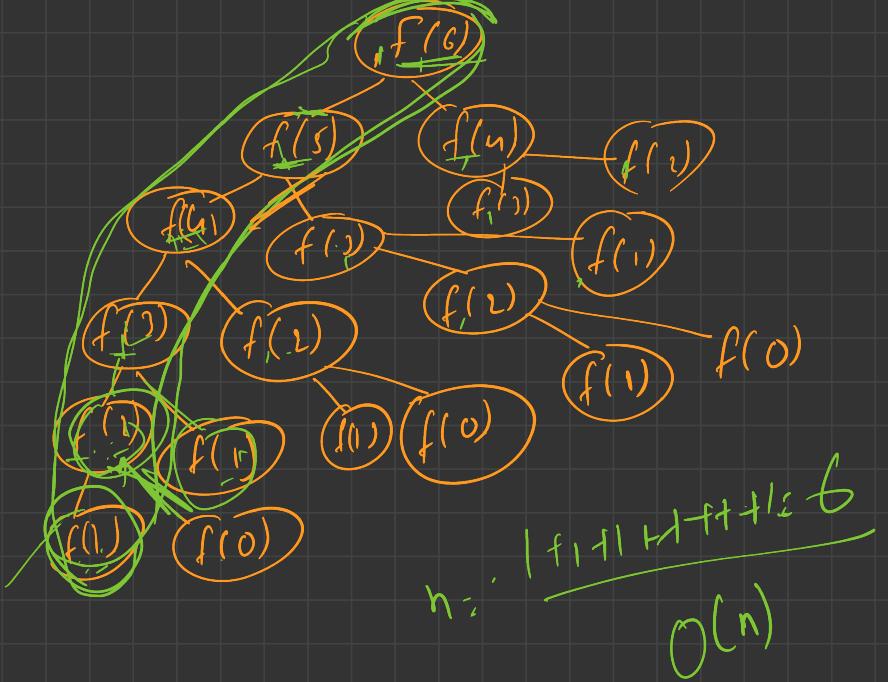


$$n = 2^k$$

$$\log n = k \log 2$$

$$k = \frac{\log n}{\log 2} \quad k = \log_2 n$$

$$\boxed{k+1} \quad O(\log n)$$

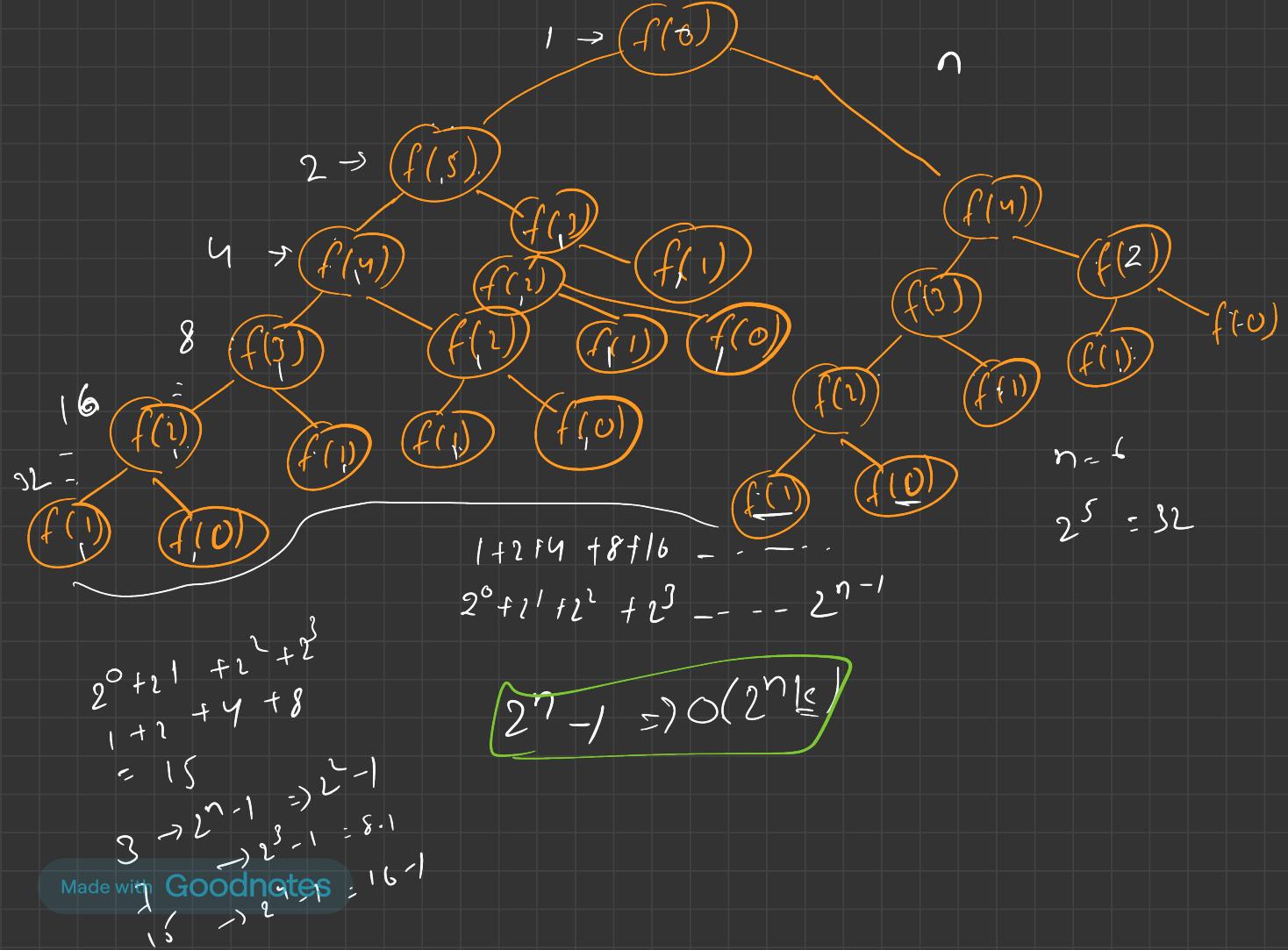


$n = \underbrace{f_1 + f_1 + f_1 + f_1 + f_1 + f_1}_{O(n)}$

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

 $T(n) = T(n-1) + T(n-2) + 6$ 
 $\downarrow$ 
 $2^{n-1}$ 

```



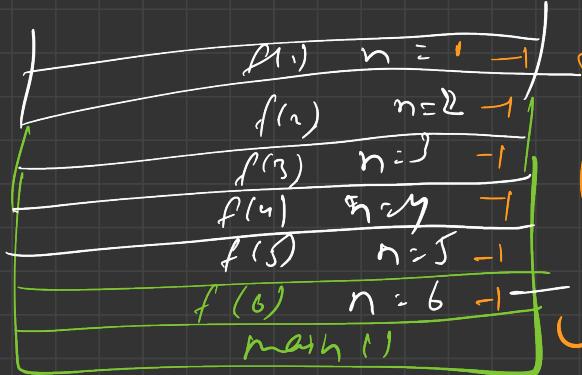
```
int fib (int n) {
```

```
    if (n <= 1) {  
        return n; }
```

```
    return fib(n-1) + fib(n-2);
```

Stack

Memory  $f(6)$



6

$O(n)$

Space

