Day2

08 November 2022 20:51

2 month

Time & space Complexity

int i = 1; — O(1)

i = i + 1;

★ Simple statements → T.C - O(1) constant

if

Loop → varies → (Time Complexity)

P1 (sum of two number)

```
#include<iostream>
using namespace std;
int main(){
    int a=5, b=6;
    cout<<a+b<<endl;
}
```

T.C

O(1)

3 TC

if(a<b){

print().

```
if(a<b)
    print();
cin>> a;
```

Input | output

T.C - O(1)

Note

* Simple Statement ka Constant time

Loop

```
for(int i=n; i>0; i--){
    // some constant work.
}
```

L1

```
for(int i=0; i<n; i++){
    // some constant work
}
```

L2 → Big Oh order of

T.C - O(N)

L1 n=6

i = 6, 5, 4, 3, 2, 1

L2 n=10

i = 0, 1, 2, 3 --- 9, 10

** Time Complexity

* Input के हिसाब से program कितना Time ले रहा है वो T.C

Theory

Best	Average (Actual)	Worst
Omega (Ω)	Theta (Θ)	Big-Oh Notation
*	*	O(input) ✓
n=1	1 to n sum	n = 2,00,000
sum=1	2 to n	

Ex

```
for(int i=0; i<n; i=i+c){
    //Some Constant Work.
}
```

T.C

$\frac{n}{c}$

S -

Su -

Ex n = 10 c = 2

i = 0, 2, 4, 6, 8, 10 ✗

10/2 ⇒ 5

** Time Complexity में Constant को consider

** Time Complexity में Constant को consider नहीं करते ||

n → input

T.C – O(n/c)

** T.C हमेशा input value के Term पर निर्भर करती है ||

T.C – O(n)

Ex2

```
for(int i=n; i>0; i=i-c){
    // some Constant work.
}
```

T.C

UT – O(N)

i=n → i>0 i=i-c

c=2

n=10

i=10	10>0	O(n/c)
i=8	8>0	
i=6	6>0	
i=4	4>0	O(10/2)
i=2	2>0	
i=0	0>0 X	O(5)

T.C - $O(\frac{n}{c})$

$O(n)$

Ex 3

for(int $i=1$; $i<n$; $i=i \times c$) {

// some Constant work

}

$n = 32$ $c = 2$

UT - $\log_c n$

Sa

$i=1$ $1<32$ $i=2$

$i=2$ $2<32$ $i=4$

$i=4$ $4<32$ $i=8$

$i=8$ $8<32$ $i=16$

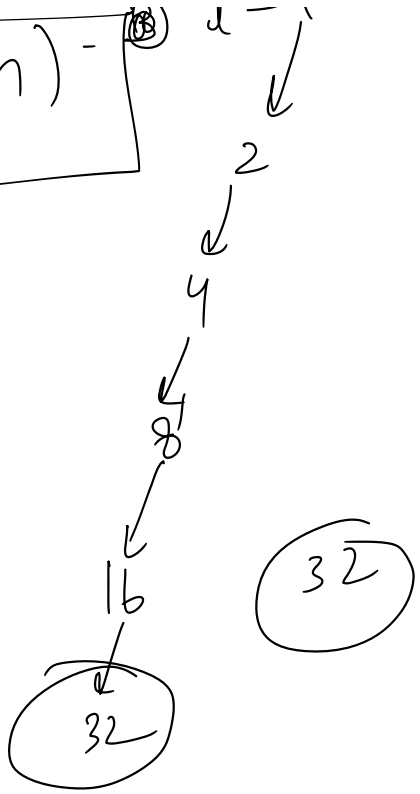
$i=16$ $16<32$ $i=32$

$32<32$ ✗

* Pattern Divide

$T.C - \log_2(n)$ ⑧ $i=1$ //

T.C – $\log_c(n)$



** जब भी हमारा Input किसी Constant के multiple में बढ़ता होता है या Divide होता होता है

T.C – $O(\log_c(n))$

Ex 4

```
for(int i=n; i>1; i=i/c){
    //Some constant work
}
```

T.C → $O(\log_c n)$

n = 32 c = 2

i = 32 32 > 1 i = 16

i = 8

i = 32

i = 16

i = 8

i = 4

i = 2

16 > 1 ✓

8 > 1

4 > 1

2 > 1

1 > 1 X

i = 8

i = 4

i = 2

i = 1

For

```
for(int i=2; i<n; i= pow(i,c))
    // some constant work.
```

T.C

c = 2

n = 32

2 < 32

4 < 32

16 < 32 ✓

256 < 32 X

$i = 2$

$i = 2^2$

$i = (2^2)^2$

$i = ((2^2)^2)^2$

Klima

$i = 2^2$

$i = (2^2)^2$

$i = ((2^2)^2)^2$

$2^0, 2^c, (2^c)^c, ((2^c)^c)^c \ldots$ K

$\Rightarrow \log 2^{c^k} < \log_2 n$

$\Rightarrow \log_2 2^c < \log_2 n$

$c^k < \log_2 n$

$k < \log_c \log_2 n$

$T.C - O(\log_c \log_2 n)$

$2^0, 2^{c^1}, 2^{c^2}, 2^{c^3}, 2^{c^4} - - 2^{c^k}$

$2^{c^k} < n$

$k \rightarrow \log_c \log_2 n$

$T.C - O(\log_c \log_2 n)$

Nested loop

Recursion

T.C ?

```
void fun(int n){
  int a=10;        O(1)
  print(a);        O(1)
  for(int i=0; i<n; i++){ }   O(n)
```

//some Constant_work

```
for(int i=1; i<n; i=i*2){
    // constant
}                                   } O(log2n)

for(int i=1; i<100; i++){           } O(1)
}
```

(worst case) O(n) + O(log2n) + O(1)

smaller

** we neglect the T.C.

T.C

O(1) < log log n < log n < n^1/3 < √n < n < n log n < n^2 < n^3 < n^4 < 2^n < n^n < n!

T.C - O(n) + O(log n) + O(1)

Neglect

T.C - O(n) + O(log n)

T.C - O(n) + O(log n)

T.C - O(N)

Time Complexity Chart

Ex:-

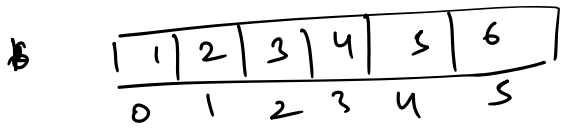
```
void fun(int n) {
    for(int i = 0; i < n; i++) {          — O(N)
        for(int j = 1; j < n; j = j * 2) { — O(log2 n)
            //some constant work
        }
    }
}
```

sa - n log n
ka - n log n

T.C → O(N * log2 n)

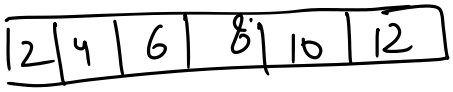
Space Complexity

Order of growth of memory space in terms of input size.



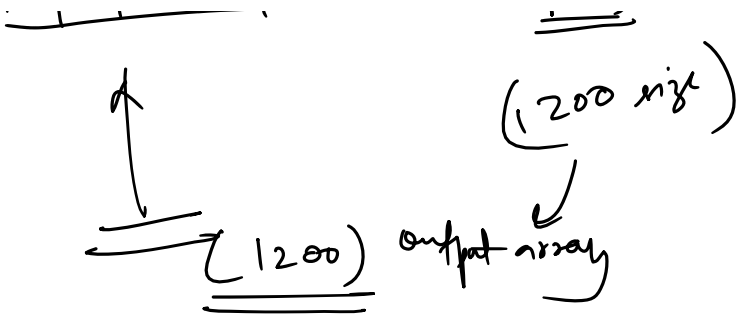
(6 size)

2X second array



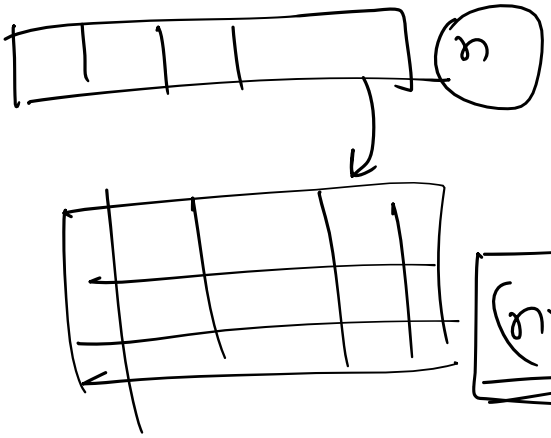
(output)

(2n size)

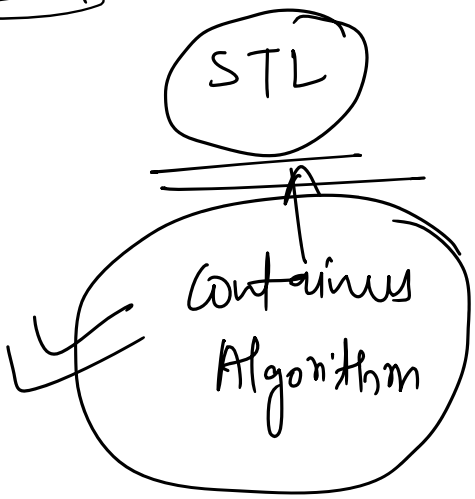


S.C - O(n)

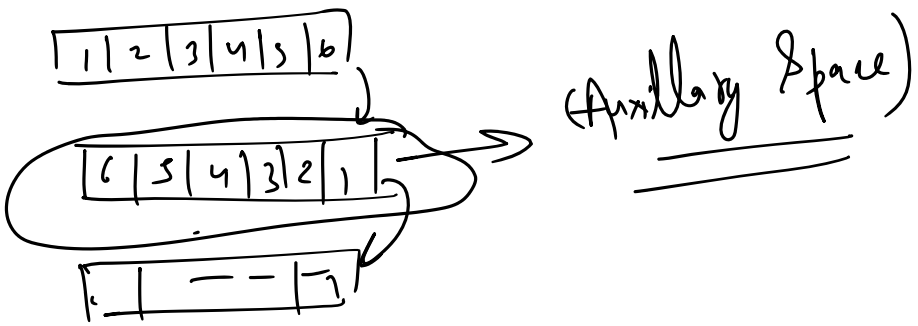
n = 6
n = 1200
n = 100,000



S.C - O(n x n)



Auxillary Space



Order of growth of extra space or temp memory

** Recursion T.C

Recursion

Sieve of Eratosthenes

* Prime Number — $O(\sqrt{n})$

(n) 1 to n (Prime Numbers)

return

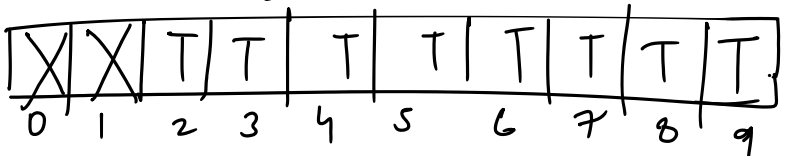
** for an input n. Find all prime numbers from 1 to n.

Algorithm

n = 9 → 1 to 9 → 2, 3, 5, 7

size = n+1

(1) boolean

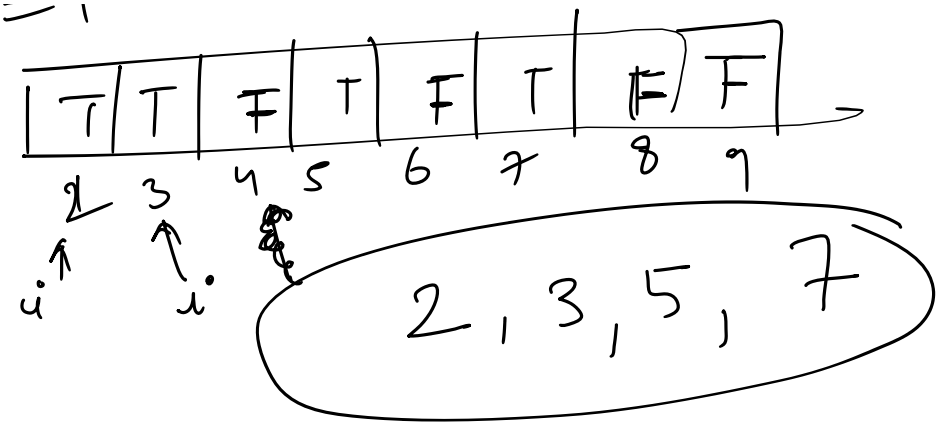


(Array of size = input+1)

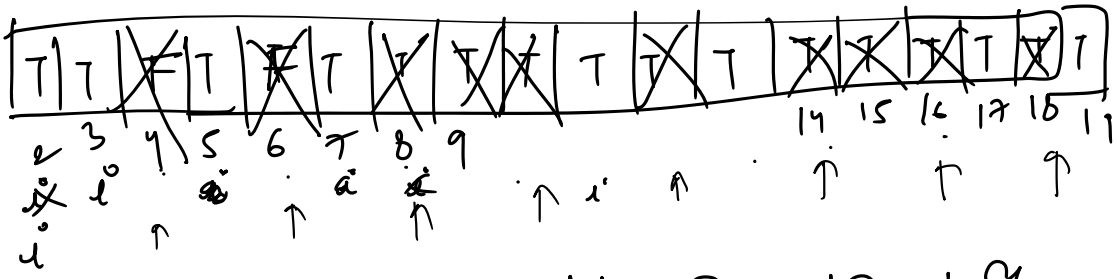
type → boolean

(2) Initialize True (Initial)

n = 9



n=19



Worst case

1 to n — O(n)

prime number — $O(\sqrt{n})$

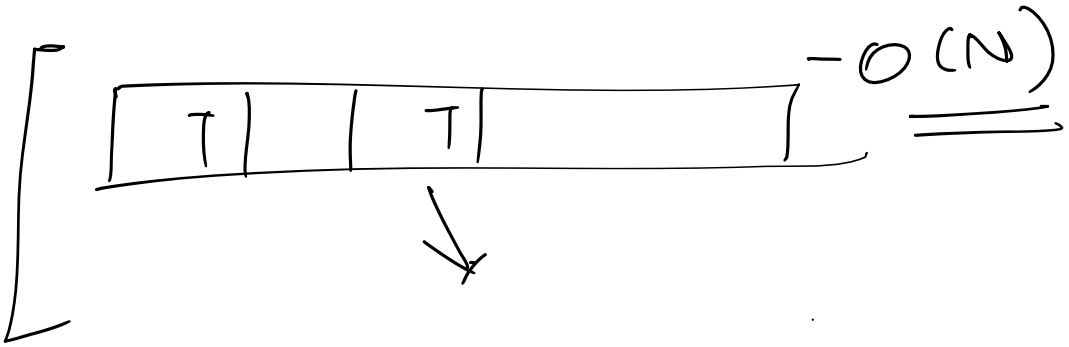
T.C — $O(n \times \sqrt{n})$

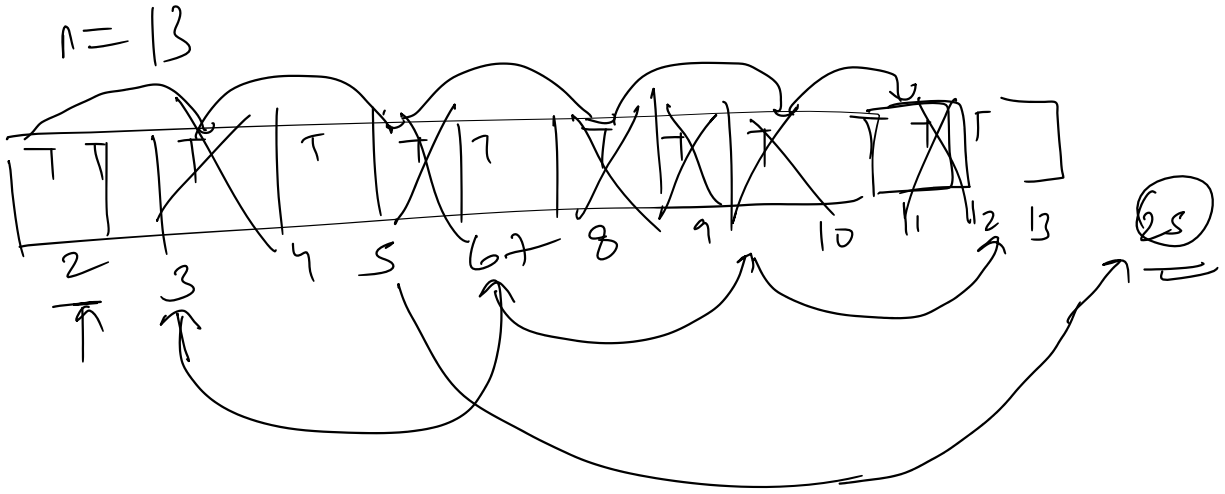
T.C — O(n)

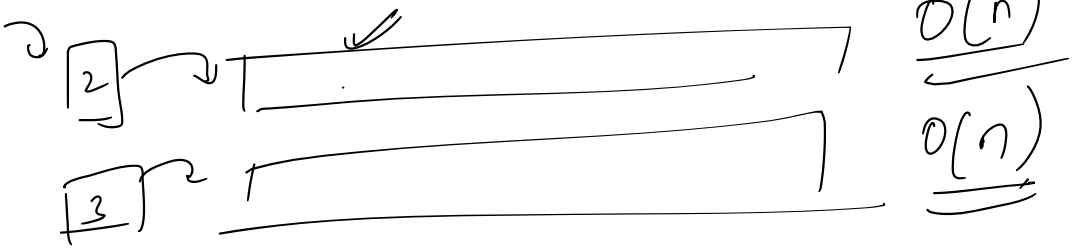
```
for(int i=2; i<=n; i++){                    O(n)
    [for(int j=i*i; j<=n; j=j+i){
        prime[j] = false;                   → O(log log n)
```

prime[j] = false; → O(log log n)

T.C – O(N × N)







Worst Case

$O(n \times \sqrt{n})$

$O(n \times \log\log n)$

multiply of 2

```
for(j=4; j<=n; j=j+2){
    prime[j] = false;
}
```
$O(N)$

```
for(j=9; j<=n; j=j+3){
    prime[j] = false;
}
```
$O(N)$

```
for(int i=5; i<=n; i++){
    for(int j=i*i; j<=n; j=j+i){
        prime[j] = false;
    }
}
```
$O(n \times \log\log n)$

```
for(int i=2; i<=n; i++){
    if(prime[i]==true){
        cout<<i<<endl;
    }
}
```

T.C- $O(n \log\log n)$

S.C- $O(n)$