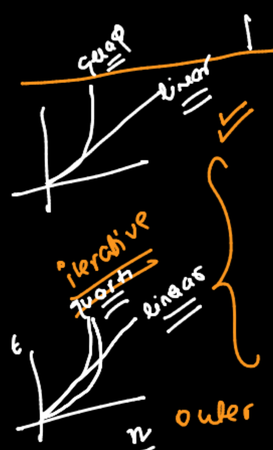
eg:1 f(int n) { approximation

loop [for (int i=1 ; i≤n ; i++) { cout << i << endl; }] ← O(1)

$10^9 \cdot cos \approx 10^9$ times, const work

$= n \cdot const \approx n \Rightarrow O(n)$



f(int n) {

for (int i=1 ; i≤n ; i++) { → n^2

for (int j=1 ; j≤n ; j++) { }] n times

cout << i << j ; ← const

$\Theta(n)$ ←

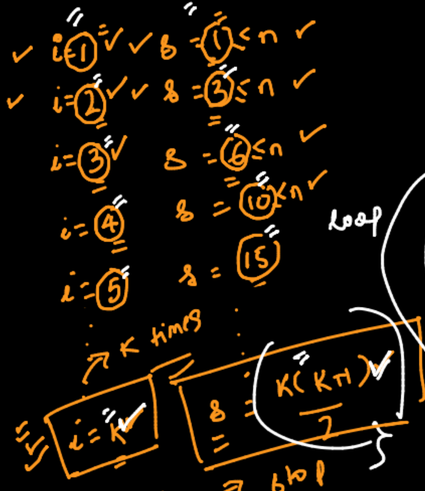
? ←

outer loop ⇒ n times

10^3 $c=2$ $2 \cdot 10^3 \approx 10^3$ $n \cdot \Theta(n) \sim O(n^2)$ → quadratic

$1+2+3+4+\dots+K = \frac{K(K+1)}{2}$

eg:3 f(int n) {



int i=1;

int s=1;

while (s ≤ n) { $O(\sqrt{n})$

loop

const → i++;

s += i;

$\frac{K(K+1)}{2} \approx n$ ⇒ $\frac{K^2}{2} + \frac{K}{2} = n$ ← dominant term

$\approx K^2 = n$

$\approx K = (\sqrt{n})$

eg:4 f(int n) {

i=1 ; i*i≤n ; i++

i*i≤n

$i \le \sqrt{n}$

$\sqrt{n}$ times

$O(\sqrt{n})$

for (int i=1 ; i*i≤n ; i++) {

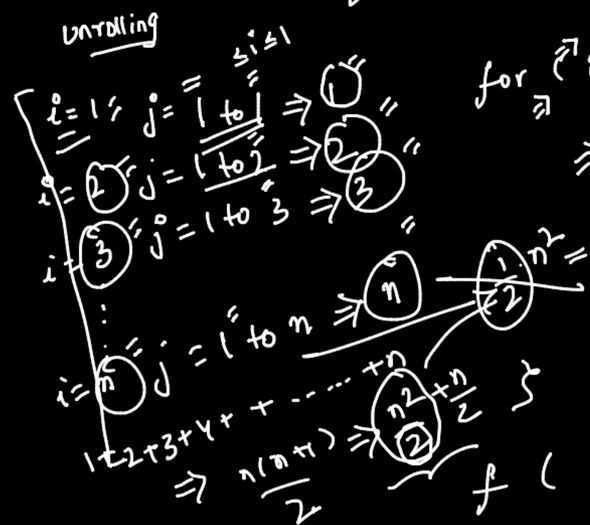
= O(1) ⇒ cout << i ;

}

}

eg:5 f(int n) {

unrolling



for (int i=1 ; i<=n ; i++) {

for (int j=1 ; j<=i ; j++) {

cout << i << j ;

}

}

$1^2+2^2+3^2+\dots+n^2$

$\Rightarrow \frac{n(n+1)}{2}\frac{(2n+1)}{6}$ ... const

$\Rightarrow O(n^3)$

unroll i=1 j=1 to 1 ⇒ 1

i=2 j=1 to 4 ⇒ 4

i=3 j=1 to 9 ⇒ 9

i=n j=1 to n^2 ⇒ n^2

for(int i=1 ; i<=n ; i++){

[for (int j=1 ; j<=i*i ; j++) {

cout << i << j ;

}

eg: 7

```
int f(int n) {
    for (int i=1; i<n; i*=2) {
        cout << i;
    }
}
```

$i = 1 \Rightarrow 2^0 < n$
$\downarrow *2$
$i = 2 \Rightarrow 2^1 < n$
$\downarrow *2$
$i = 4 \Rightarrow 2^2 < n$
$\downarrow *2$
$i = 8 \Rightarrow 2^3 < n$
$\downarrow *2$
$i = 16 \Rightarrow 2^4 < n$
$\vdots$
$\downarrow *2$
$i = 2^k \rightarrow$ stop

$2^k = n$
$k = \log_2 n$
$\Rightarrow O(\log_2 n)$

eg: 8

```
f(int n) {
    for (int i=n/2; i<=n; i++) {
        for (int j=1; j<=n/2; j++) {
            for (int k=1; k<n; k*=2) {
                cout << i << j << k;
            }
        }
    }
}
```

$n/2 \quad \log n \quad \log n$

$O(n(\log n)^2)$

eg: 9

```
f(int n) {
    for (int i=n/2; i<=n; i++) {
        for (int j=1; j<n; j*=2) {
            for (int k=1; k<n; k*=2) {
                cout << i << j << k;
            }
        }
    }
}
```

eg: 10

```
f(int n) {
    while (n>1) {
        n /= 2;
    }
}
```

$n \rightarrow n/2^0 > 1$
$n/2 = n/2^1 > 1$
$n/4 = n/2^2 > 1$
$n/8 = n/2^3 > 1$
$\vdots$
$n/2^k = 1 \Rightarrow n = 2^k$
$k = \log_2 n$

$O(\log_2 n)$

eg: 11

```
f(int n) {
  for(int i=1; i<=n; i++) {
    for(int j=1; j<=n; j+=i) {
      cout << i << j;
    }
  }
}
```

i=1 j=1 to n ⇒ n
i=2 j=1 to n ⇒ n/2
i=3 j=1 to n ⇒ n/3
⋮
i=n j=1 to n ⇒ n/n

n + n/2 + n/3 + ... + n/n
n(1/1 + 1/2 + 1/3 + 1/4 + ... + 1/n) ⇒ O(n log n)
log n

eg: 12

```
BinarySearch(int A[], int n, int key) {
  int s = 0;
  int e = n-1;
  while(s <= e) {
    mid = (s+e)/2;
    if(A[mid] == key) {
      return mid;
    } else if(A[mid] > key) {
      e = mid-1
    } else {
      s = mid+1
    }
  }
  return -1;
}
```

n → s=0, e=n-1
s=mid+1 → s=0, e=n/2-1
s=mid+1 → s=0, e=n/4-1
s=mid+1 → s=0, e=n/8-1
⋮
s=mid+1 → s=0, e=n/2^K-1

K = log2 n ⇒ O(log n)
n/2^K = 1
n = 2^K
K = log2 n
log2 n + 1 ≈ log2 n

n ⇒ sort(a, a+n) ⇒ also : n log n

eg: 13

```
Bubble Sort(int A[], int n) {
  for(int i=0; i<n-1; i++) {
    for(int j=0; j<n-i-1; j++) {
      if(A[j] > A[j+1]) {
        swap(A[j], A[j+1])
      }
    }
  }
}
```

i=0 j=0 to <n-1 ⇒ n-1
i=1 j=0 to <n-2 ⇒ n-2
i=2 j=0 to <n-3 ⇒ n-3
⋮
n ⇒ 10^6 O(n^2) ⇒ 10^12 > 10^9 ⇒ TLE
i=n-2 j=0 to 1 ⇒ 1

1 + 2 + 3 + ... + n-3 + n-2 + n-1 + n - n
n(n-1)/2 = n^2/2 - n/2
n(n-1)/2 - n ⇒ O(n^2)

Time Limit Exceeded

eg: 14

```
f(int n) {
  // O(1) ⇒
  cout << n * 10;
}
```

$\Rightarrow$ Back substitution

$\Rightarrow f(n)$ { $1+1+\cdots+1$ (n times) $\Rightarrow O(n)$

$\Rightarrow T(n)$

$\Rightarrow T(n) = \begin{cases} \text{if}(n==1)\ \{ \text{base} \\ \quad \text{return}; \\ \} \end{cases}$ [1] [C] = const

$= \begin{cases} \text{return } f(n-1) \\ T(n-1) \end{cases}$ = recursive

}

$\Rightarrow T(n) = T(n-1) + C$ [RC]

$T(1) = C$ $n=1$ [BC]

$T(n) = T(n-1) + C$ — (1)

$T(n-1) = T(n-2) + C$ — (2)

$T(n-2) = T(n-3) + C$ — (3)

$T(n) = T(n-2) + 2C$

$T(n) = T(n-3) + 3C$

$\vdots$

$T(n) = T(n-K) + KC$

$n-K=1 \Rightarrow K = n-1$

$= T(n-(n-1)) + (n-1)C$

$= T(n-n+1) + (n-1)C$

$= T(1) + C(n-1)$

$= C + (n-1)C$

$= nC$

$\sim O(n)$

$T(n) \rightarrow f(n)$ {

Base & $\begin{cases} \text{if}(n==1)\ \{ \\ \quad \text{return}; \\ \} \end{cases} + 1 \Rightarrow \frac{n(n+1)}{2} \simeq O(n^2)$

$n + n-1 + n-2 + \cdots$

$\begin{cases} \text{for}(\text{int } i=1; i \le n; i++)\ \{ \\ \} \end{cases}$ n times

$\checkmark$ = return $f(n-1)$;

}

[C] [1] $T(n) = T(n-1) + n + C$ $\Rightarrow$ log $\rightarrow T(n-1)$

$T(n) = T(n-1) + n$ — (1)

[C] [1] $T(n-1) = T(n-2) + n-1$ — (2)

$T(n-2) = T(n-3) + n-2$ — (3)

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

$= T(n-3) + n-2 + n-1 + n$

$\vdots$

$= T(n-K) + \cdots + n-2 + n-1 + n$

$n-K=1$

$K = n-1$

$\Rightarrow T(1) + \cdots + 1+2+3 \cdots n \Rightarrow \frac{n(n+1)}{2} = O(n^2)$

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

T(n) = c (Basic)

```
return f(n/2) + f(n/2);
```

T(n/2) + T(n/2) (Recursive)

}

$T(n) = 2T(n/2) + c$ — ①

$T(n/2) = 2T(n/4) + c$ — ②

$T(n/4) = 2T(n/8) + c$ — ③

$T(n) = 2(2T(n/4) + c) + c$

$= 2^2 T(n/4) + 2c + c$

$= 2^2 (2T(n/8) + c) + 2c + c$

$= 2^3 T(n/8) + 2^2 c + 2c + c$

$= 2^k T(n/2^k) + \dots + 2^2 c + 2c + c$

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

$n = 2^k$

$k = \log_2 n$

$= 2^{\log_2 n} T(1) + \dots + 2^2 c + 2c + c$

$= nc + \dots + 2^2 c + 2c + c$

$n/2^k = 1 \Rightarrow k = \log_2 n$

$T(n) = 2^k T(n/2^k) + \dots + 2^2 c + 2c + 2^0 c$

$= 2^k T(1) + 2^{k-1} c + 2^{k-2} c + \dots 2^2 c + 2c + c$

$= 2^k c + 2^{k-1} c + 2^{k-2} c + \dots 2^2 c + 2c + c$

$= c(2^k + 2^{k-1} + \dots 2^2 + 2^1 + 2^0)$

GP: $2^0 \left(\frac{2^{k+1} - 1}{2 - 1}\right)$, $r > 1$

$c \cdot \frac{1(2^{k+1} - 1)}{2 - 1 \Rightarrow 1} \Leftarrow 1\left(\frac{r^n - 1}{r - 1}\right)$

$= c(2^{k+1} - 1)$

$= 2^{k+1} c - c$

$= 2^{(k+1)} c$

$= 2^{\log n + 1} \cdot c$

$= 2^{\log_2 n} \cdot 2^1 \cdot c$

$= 2^{\log_2 n} \cdot c$

$\log_b a = c \Rightarrow a = b^c$

$y = 2^{\log_2 n}$

$\log_2 y = \log_2^n \frac{\log_2^2}{1}$

$\log_2 y = \log_2^n$

$\log_2 y - \log_2^n = 0$

$\log\left(\frac{y}{n}\right) = 0$

$\frac{y}{n} = 1$

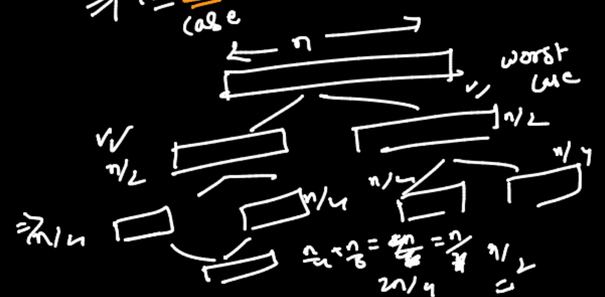
$y = n$

```
f(n) {
  if (n > 1) {
    f(n/2)
    f(n/2)
    g(....)
  }
}
```

$T(n)$ → $T(n/2)$, $T(n/2)$, $O(n)$ → merge

merge sort

⇒ assume it takes $O(n)$ in worst case



$T(n) = 2T(n/2) + n$ — ①

$T(n/2) = 2T(n/4) + n/2$ — ②

$T(n/4) = 2T(n/8) + n/4$ — ③

$T(n) = 2(2T(n/4) + n/2) + n$

$= 2^2 T(n/4) + 2n/2 + n$

$= 2^2 T(n/4) + 2n$

$T(n) = 2^2(2T(n/8) + n/4) + 2n$

$= 2^3 T(n/8) + 2^2 n/4 + 2n$

$= 2^3 T(n/8) + 3n$

$= \dots$

$= 2^k T(n/2^k) + kn$ ⟸

$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$T(n) = 2^k T(n/2^k) + k \cdot n$

$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$T(n) = 2^{\log_2 n} T(1) + \log_2 n \cdot n$

$= n \cdot 1 + n \log_2 n$

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

$n + 3n$

$4n$

```
f(n) {
T(n)  if (n == 0 || n == 1) { return } c
      }
      return f(n-1) + f(n-2)
             T(n-1)   T(n-2)
      ≤ 2cs
}
```

$T(n) = T(n-1) + T(n-2) + c$

$T(n) \approx 2T(n-1) + c$ ← ①

$T(n-1) = 2T(n-2) + c$ — ②

$T(n-2) = 2T(n-3) + c$ — ③

$T(n) = 2(2T(n-2)+c) + c$

$= 2^2 T(n-2) + 2c + c$

$= 2^2(2T(n-3)+c) + 2c + c$

$= 2^3 T(n-3) + 2^2 c + 2c + c$

$\vdots$

$= 2^k T(n-k) + \cdots + 2^2 c + 2c + c$

$n = 0$ or $n - k = 0$, $k = n$

$= 2^k T(0) + \cdots + 2^2 c + 2c + c$

$= 2^k c + 2^{k-1} c + \cdots + 2^2 c + 2c + c$

$= c(2^k + 2^{k-1} + \cdots + 2^2 + 2 + 1)$

$\Rightarrow c\left(\frac{2^{k+1}-1}{2-1}\right)$

$\Rightarrow 2^{k+1} c \Rightarrow 2^{k+1}$ ← exponential

$\Rightarrow 2^{(k)+1}$

$\Rightarrow 2^{n+1} \sim O(2^n)$

Best

$O(1) < O(\log\log n) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) \cdots$ worst $< O(n^n)$

$n! \Leftarrow n^n$

$\Rightarrow n(n-1)(n-2)(\cdots)(\cdots)$ ← worst of worst

$n \Rightarrow 2^{2^{13}}$

$\log_2 2^{2^{13}} \Rightarrow 2^{13}$

$\log\log n \Rightarrow 2^{13}$

13

$2^{10} = 1024$

$2^{11} = 2048$

$2^{12} = 4096$

2^{13}

409자

×2

8192

Space complexity → iterative = recursive =

↓

extra space apart from i/p

space = O(1)

```
int fact(int n)
{
    int ans = 1;
    for(int i = n; i >= 1; i--) {
        ans *= i;
    }
    return ans;
}
```

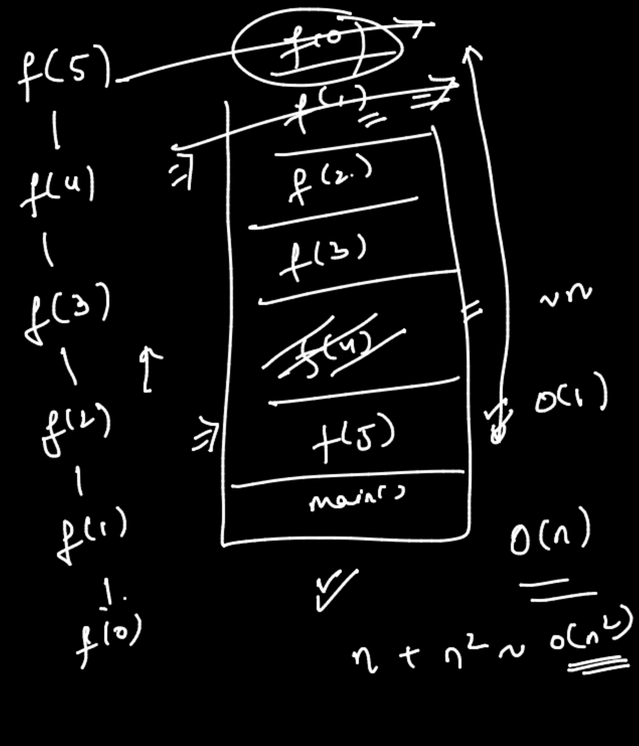
time O(n)

n × n ⇒ n²

pass by ref

x ... n times

n = 5



time O(n)

```
int factRec(int n) {
    if(n == 0) {
        return 1
    }
    return n * factRec(n-1);
}
```

```
int fibo(int n) {
    if (n==0 || n==1)
        return n;
    int a = 0;
    int b = 1;
    for (int i=2; i<=n; i++) {
        int c = a+b;
        a = b;
        b = c;
    }
    return b
}
```

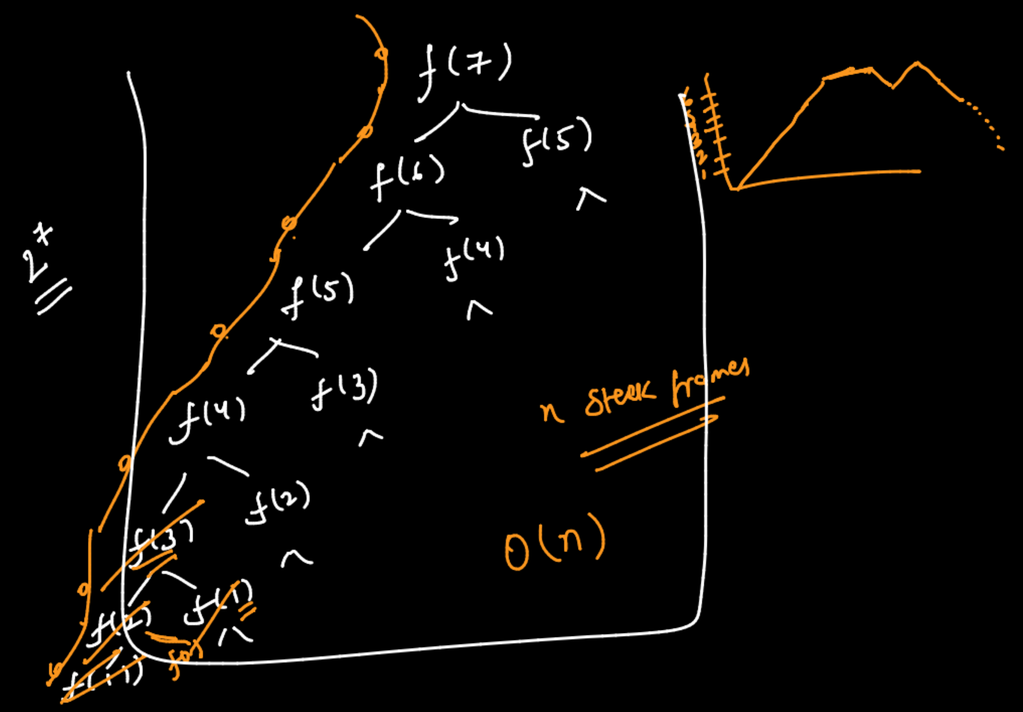
time = n-1 ~ n = O(n)

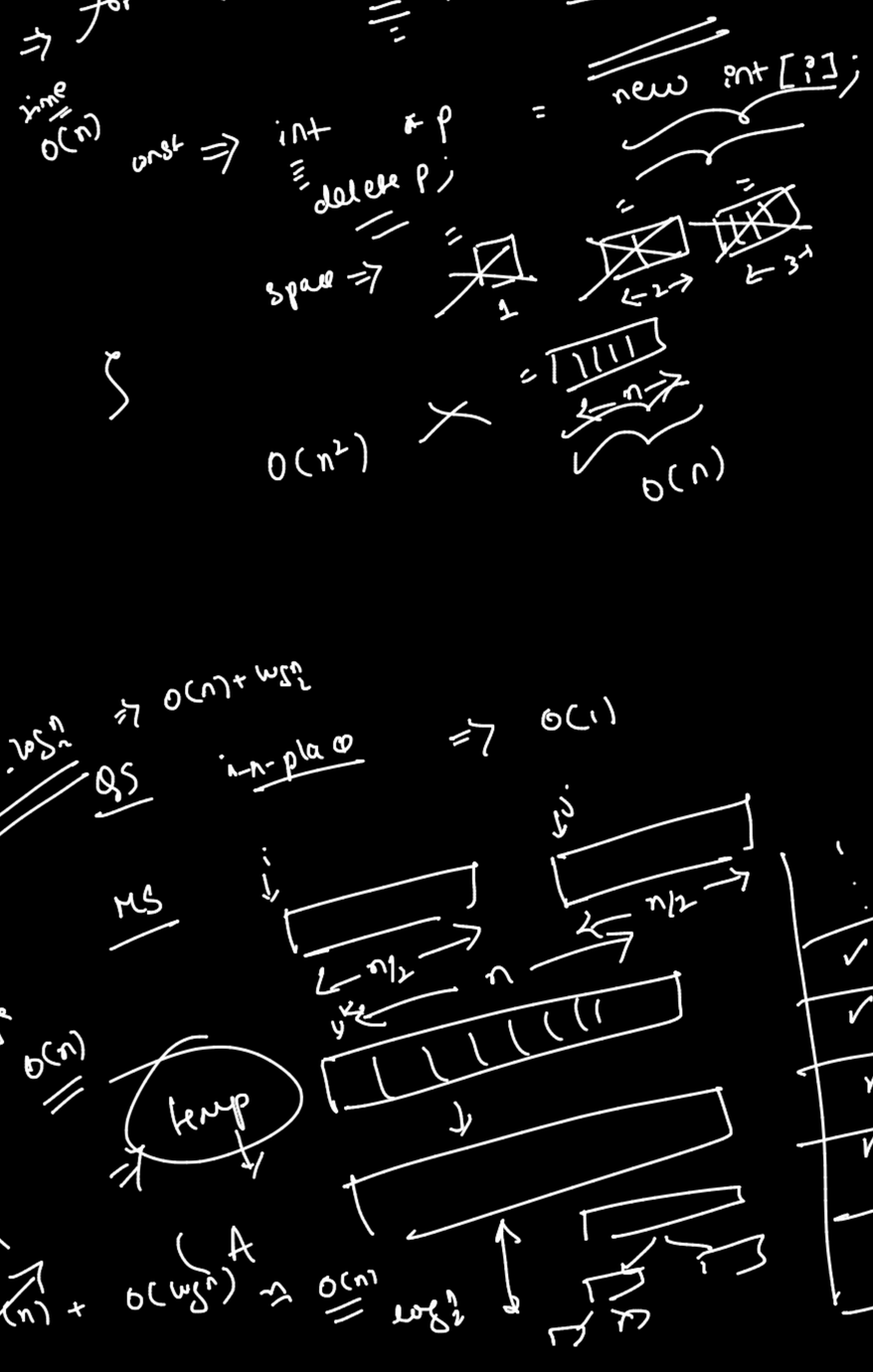
O(1)

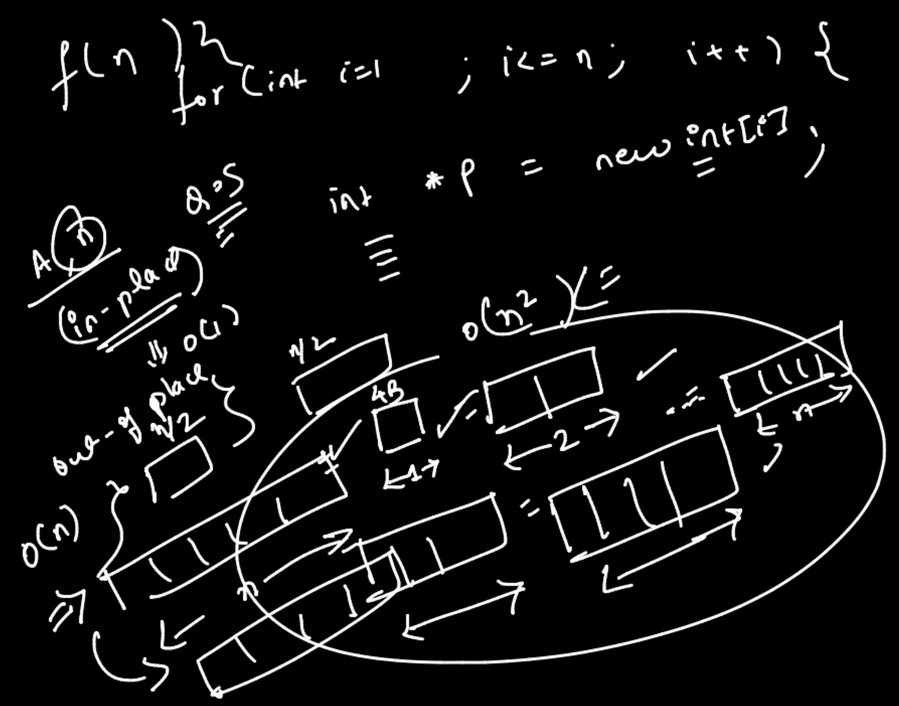
```
int fiboRec(int n) {
    if (n==0 || n==1) {
        return n;
    }
    return fiboRec(n-1) + fiboRec(n-2)
}
```

$2^n \leq$ const

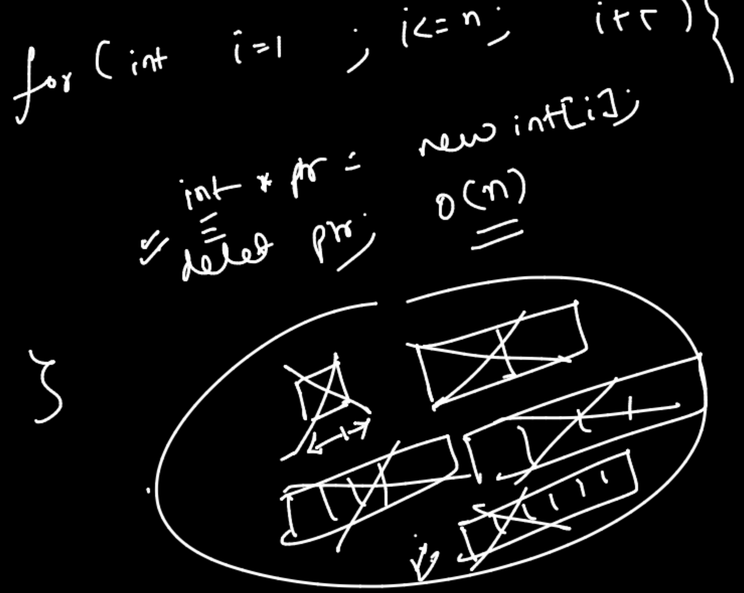
$\simeq O(2^n)$




for (int i=1 ; i<=n ; i++) {
time O(n)
const ⇒ int *p = new int[i];
delete p;
space ⇒
1
2
3
n
O(n²) ×
O(n)


f(n)
for (int i=1 ; i<=n ; i++) {
int *p = new int[i];
A (in-place)
O(1)
out-of-place
O(n)
n/2
4B
1
2
n
O(n²)

O(n)·log n ⇒ O(n)+ log n
QS in-place ⇒ O(1)
MS
i
n/2
n
n/2
n/2
log n
merge
O(n)
temp
O(n) + O(log n) ⇒ O(n)
A
log n


for (int i=1 ; i<=n ; i++) {
int *ptr = new int[i];
delete ptr;
O(n)
}

why algo. analysis is done ?

↳ to compare

= Sort → SS
= → BS
→ IS
⇝ MS
⟶ QS
⋮

Search → linear search
Sorted → binary search ✓

O(n)
⇑
⇒ f(n) → f(n-1) → f(n-2) → ⋯ f(1) ← base
= =

Time complexity n · n = n² worst case

Theoritical
x
⇒ = Experimental

↳ system dependent

10 cm

Pentium 2 | M1
= A → 25 | A → 1 ms
B → 45 | B → 0.9999 ms

⇒ write code for each

= iterative
contains loop
↳ count # of steps in the loop
⇒ n * time for each
=

= recursive ✓
recursion
↳ count # recurs * time for each recursive

constant O(1)

how to compare algorithms?

— less space → space complexity
/ less time → time complexity

✓ A
✓ B

Mergesort

Merge $\Rightarrow O(n)$ $\Rightarrow$ $O(n) + O(\log n)$ $\Rightarrow O(n)$