Time Complexity

↳ Baically the amt of time it takes to execute something

Eg:

```
int count = 0;
for (int i=0; i<N; i++){
    for (int j=0; j<i; j++){
        count++;
    }
}
```

$1 + N + N + \frac{N(N+1)}{2} + \frac{N(N+1)}{2}$

$1+2+3+\ldots+N = \frac{N(N+1)}{2}$

So baically time taken to run the above code = $N^2 + 3N + 1$

Order Notations

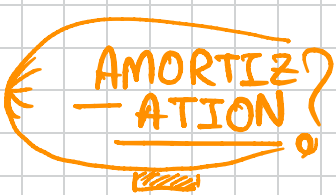
↳ Types:

- → $O(f(n))$ → upper bound
- → $\Omega(f(n))$ → lower bount
- → $\Theta(f(n))$ → Sandwhich of O & Ω

Sequence	Nesting
↓	↓
addition	multiplication

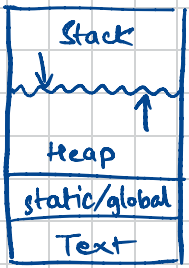
↳ JUST KEEP IN MIND

$n \leq 400$	$O(n^3)$
$n \leq 7500$	$O(n^2)$
$n \leq 10^5$	$O(N\sqrt{N})$
$n \leq 5 \times 10^5$	$O(n \log n)$
$n \leq 5 \times 10^6$	$O(N)$
$n \leq 10^{12}$	$O(\sqrt{N} \log N), O(\sqrt{N})$
$n \leq 10^{16}$	$O(\log^2 N), O(1), O(\log N)$



Memory Complexity

In cpp,



arr[n][N]

↳ O(N²)

NP-HARD → You should know what these problems are. TO LEAVE THEM !

↳ Problems not solvable in polynomial time.

finding time complexities for RECURSIVE PROBLEMS

Master Theorem

↳ Requirement:

$$T(n) = aT\left(\frac{n}{b}\right) + c$$

↓ How much time will it take to solve instance of size 'n'.

applicable

→ let say MERGE SORT

So,

$$T(n) = 2T\left(\frac{n}{2}\right) + O(N)$$

Calculating for merge sort,

Step 1: Note a, b, c

Step 2: Calculate $\left(n^{\log_b a}, c\right)$

$\Rightarrow O(n^{\log_2 2}), O(N) \Rightarrow O(N), O(N)$

Step 3 if $O(n^{\log_b a})$ & $c \rightarrow$ SAME

↳ THEN → ans is $c \log n$

if $O(n^{\log_b a}) > C$

↳ THEN → ans is $O(n^{\log_b a})$

else

↳ ans is $O(C)$

∴ for merge sort → TC → $O(n \log n)$

Examples

↳

① $T(n) = 2T(n/2) + O(n^2)$

Now, $n^{\log_2 2} < n^2$

∴ TC → $O(n^2)$

② $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$

$n^{\log_2 2} > 1$

⇒ TC → $O(N)$

③ $T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\log n}$

↳ Master Th. doesn't apply when you don't have a POLYNOMIAL EXPRESSN

But still if you have to comment on TC.

↳ $C \log n$

$= O(n^3 \log n)$ → leaving out the log part.

Problems

①
```
for (int i=0; i*i<=N ; i++) {
    // O(1) work here
}
```

→ $i^2 = N \Rightarrow$ TC : $O(\sqrt{N})$

$i = \sqrt{N}$

②
```
for (int i=1 ; i*i<=N ; i++){
    for (int j=1 ; j<=i ; j++) {
        // O(1) work done here
    }
}
```
↳ it takes O(i) to complete

$O(1) + O(2) + \ldots + O(i_{end})$ ($i_{end} \to \sqrt{N}$)

$\Rightarrow O(1+2+\ldots\sqrt{N}) = O\left(\frac{\sqrt{N}(\sqrt{N}+1)}{2}\right)$

$= O(N)$

③
```
for (int i=1; i<=N ; i++){
    for (int j=1 ; j<=i ; j*=2) {
        // O(1) work
    }
}
```
O(log i)

How ?

So for x^{th} step j goes to some i

for $x-1^{th}$ " j —·— i/2

∴ $\log_2 i$

This is a bit upper bound but still it's an upper bound.

For whole $\Rightarrow \sum_{i=1}^{N} O(\log i) \leq \sum_{i=1}^{N} O(\log N)$

$\leq O(N \log N)$

④
```
for (int i=1 ; i<=N ; i++) {
    for (int j=i ; j<=N ; j+=i) {
        // O(1) work
    }
}
```

$1i, 2i, 3i, \ldots, xi$ ← no. of times this loop runs

& $xi \le N$

$\Rightarrow x \le \frac{N}{i}$

↳ Basically this loop runs $\frac{N}{i}$ times.

$\therefore TC = \sum_{i=1}^{N} O(N/i) = O\left(\frac{N}{1} + \frac{N}{2} + \frac{N}{3} + \ldots + \frac{N}{N}\right)$

$= O\left(N\left(\frac{1}{1} + \frac{1}{2} + \ldots \frac{1}{N}\right)\right)$

↳ can be bound by $\log N$

$= O(N \log N)$ //

Recursive Problems

①
```
int power(int a, int b) {
    if (!b) return 1;
    int temp = power(a, b/2) * power(a, b/2)
    if (b%2 == 1) temp *= a;
    return temp;
}
```

Now, $T(n) = aT(n/b) + c$

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

Then, $n^{\log_b a} = n$ & $n > 1$

$\Rightarrow O(n)$ //

②
```
int power(int a, int b){
    if(!b) return 1;
    int temp = power(a, b/2)
    temp *= temp
    if(b%2== 1) temp *=a ;
}   return temp;
```

$T(n) = T(\frac{n}{2}) + O(1)$

$\Rightarrow n^{\log_b a} = n^0 = 1 = c = 1$

$\Rightarrow O(c\log n) = O(\log n)$ //