

Algorithms + Problems Time Complexity

C1 - 15 sec.

C2 - 1 min.

~~xxx Not measured in time~~

Number of operations taken to complete something.

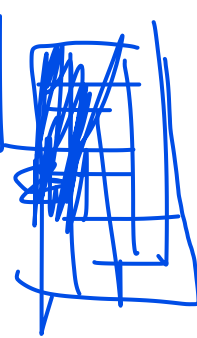
~~faster computer takes less time~~

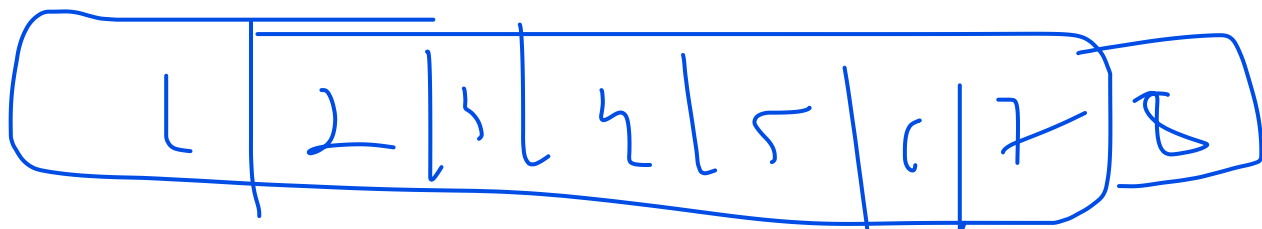
In addition to time complexity, we measure space complexity.

C1 - 15 sec. - more memory

may be



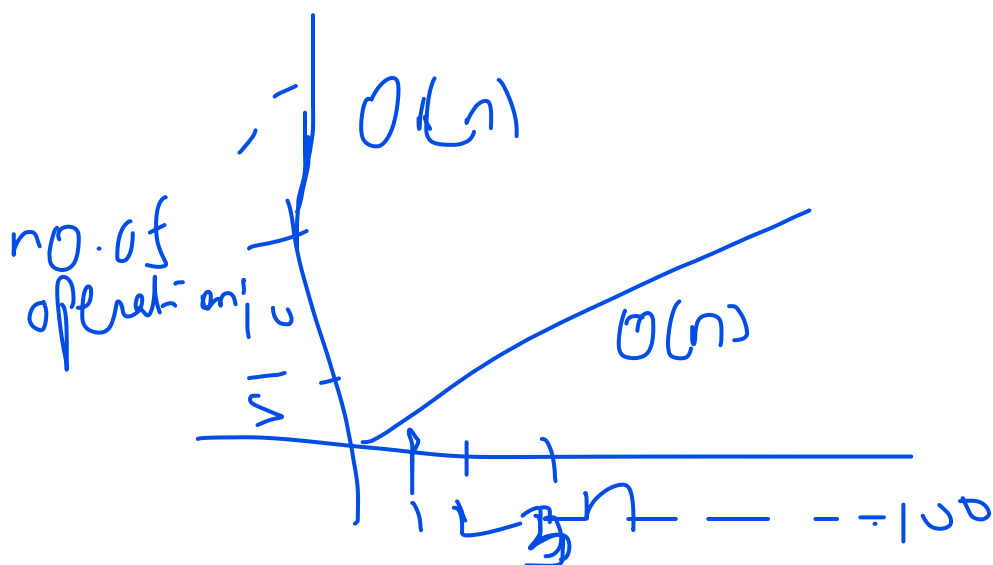
$C_2 - 1 \text{ min.}$ — ben memory 



Ω Θ O
(Best case) (Avg case) (Worst case)

interview — Big O only. Case — Θ
 Big O best case — Ω

for $(1 \leq i \leq n)$



$$f(i, j) = 0 \dots n$$

$$n + m = 2n$$

$$O(2n) \times$$

$$O(n) - \text{step 2 / constant}$$

```

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        h
    }
}

```

```

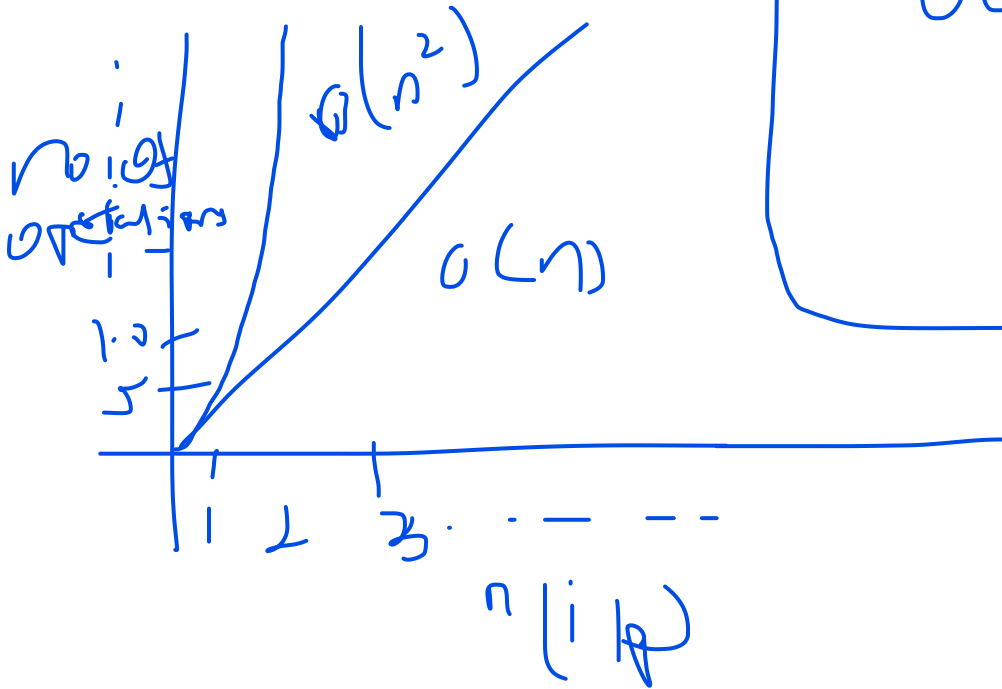
0 0
1 1
.
.
.

```

g g

$$n \times n = n^2$$

$$O(n^2)$$



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

huge gain
in efficiency.

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

```
for (int k = 0; k < n; k++) {
```

}

total no. of operations = $O(n^2 + n)$

As $n \uparrow$ size, n -non-dominant. iff.
 $O(n^2)$, drop n -non.

$O(1)$

n q no. of operation — (time)

add1 (int n) {

return n + 1;

}

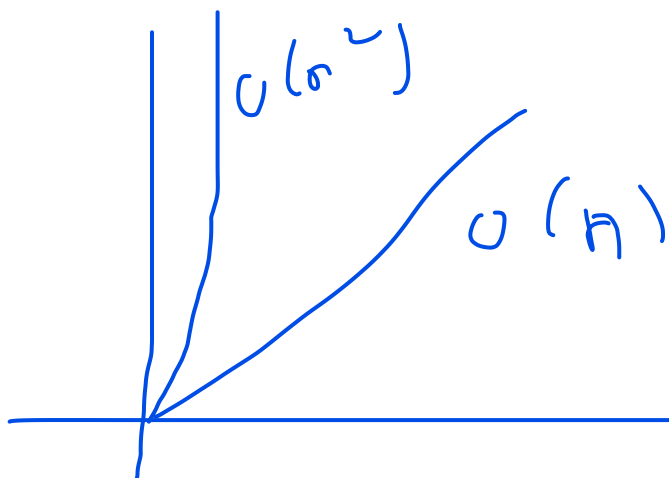
1-no. of operation stay constant

~~$O(2)$~~

$O(1)$

✓

$n + n + n$

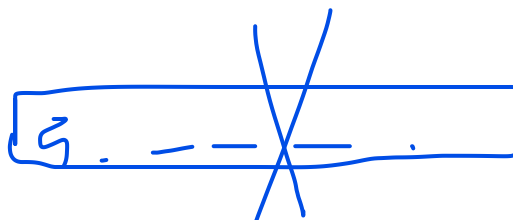
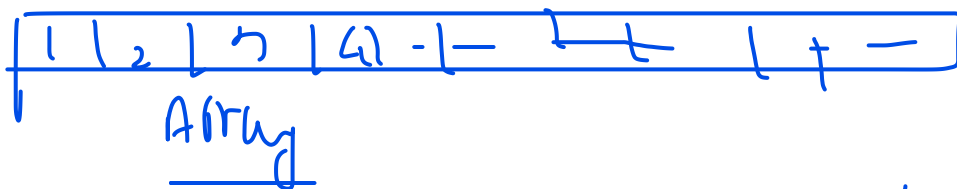


$O(1)$

most efficient
big O.

if $O(1)$ ✓

$O(\log n)$



step 1



step 2



step 3

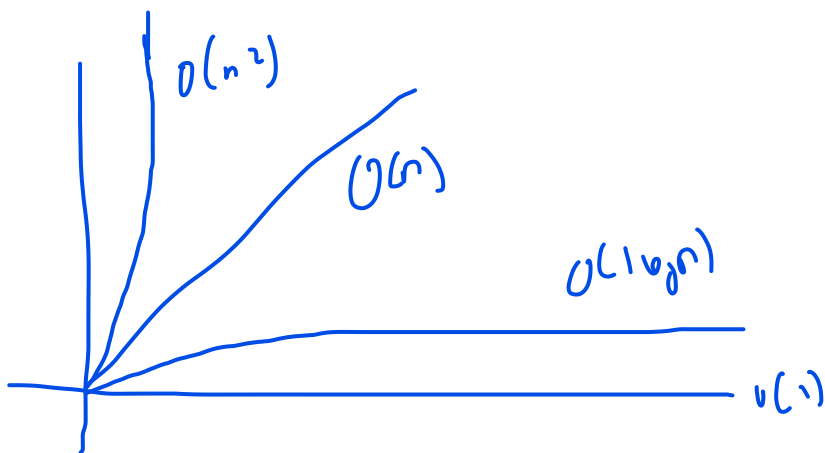
8 times in the array

$$2^3 = 8$$

$$\log_2 8 = 3$$

$$\log_2 1,073,741,824 = 31$$

31 times cut
1073.74...
in half(2)
to get
one item.



```
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
        // ...
    }
}
```

$O(a+b)$ ✓
 $O(a+b)$ ✓

$O(n \neq n)$ ✗, as a could
be 1.
 $O(n+n)$ ✗

Array list

[1, 3, 23, 7]

mylist.add(71);

11	3	23	7	17
0	1	2	3	4

$O(1)$

mylist.remove(4);

11	...	7
0		3

$O(1)$

my...rem(0);

3	23	7
1	2	3

↖ ↗ ↘

3	23	7
0	1	2

my...ad.(0, 11)

--	--	--



11	3	23	7	17
0	1	2	3	4



$O(n)$



$O(1)$

