Time Complexity

→ what is a good algorithm ??

→ Efficient

→ Running time
→ Space used

P1 → S1 (1.02 ms) ✓ , S2 (2.01 ms) ✓

① Experimental analysis

→ Write a program that implements the algorithm
→ Run the program with data sets of varying size
→ Use a method like System.currentTimeMillis();

Limitations

→ necessary to implement the algorithm in order to determine its running time.

→ In order to compare two algorithms, same hardware & software should be used.

* Execution time of the algorithm.

Order of growth
↳ how the time of execution depends on the length of input.

① O- notation [worst time of an algo]
② Θ - notations
③ Ω - notations

→ execution time in worst case

Big O notations

1. complexity in terms of input size N.
2. machine - independent
3. basic computer steps

→ time
→ space] → Big O

Types of measurement

→ Worst - case (Big O)
→ best - case
→ average case

General rule → $O(5n) \simeq O(n)$, $\simeq 10^9$

constant

① x = 5 + (15 * 20);
independent of input size
→ O(1)

② x = 5 + (15 * 20); → O(1)
y = 15 - 2; → O(1)
print x + y; → O(1)
TC → O(1) + O(1) + O(1) = 3 * O(1) = O(1)

linear

n ∝

③ for x in range (0, n):
print x;
→ O(n)

n = 5: 1 2 3 4 5
n = 3: 1 2 3

④ y = 5 + (15 * 20); → O(1)
for x in range (0, n):
print x;] → O(n)

TC → O(1) + O(n) = O(n)

certain terms dominate others

⑤ for x in range (0, n):
for y in range (0, n):
print (x * y);

TC → $O(n^2)$

n = 5
0 * 0
0 * 1
0 * 2
0 * 3
0 * 5
1 * 0
1 * 1
1 * 2

n = 5 ↳ 0 = 25
n = 3 ↳ 9

⑥ x = 5 + (15 * 20); → O(1)
for x in range (0, n):
print x;] → O(n)
for x in range (0, n):
for y in range (0, n):
print (x * y);] → $O(n^2)$

TC → $O(1) + O(n) + O(n^2)$

⑦ if x > 0:
// O(1)
else if x < 0:
// O(log n)
else :
// $O(n^2)$ ✓

TC → $O(n^2)$

⑧ i = n;
a = 0;
while (i > 0):
a += i;
→ i /= 2;

n → n/2 → n/4 → n/8 ..... 0] → log n

TC → O(log n)

⑧ int count = 0;
for (int i = n; i > 0; i /= 2)
for (int j = 0; j < i; j++) {
count ++
}
}

→ How many times count ++ run;

if i = N; → N times
if i → N/2 → n/2 times
if i = n/4 → n/4 times

Total count →

$n + \frac{n}{2} + \frac{n}{4} + \ldots 1 = 2 * N$

TC → O(N)

$S_n = \frac{a(r^n - 1)}{r - 1}$

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$

N	Accepted Complexity
≤ [10 ... 11]	O(N!) , $O(N^6)$
≤ [15 ... 18]	$O(2^N * N^2)$
≤ [18 ... 22]	$O(2^n * n)$
≤ 100	$O(n^4)$
≤ 400	$O(n^3)$
≤ 2k	$O(N^2 * \log N)$
≤ 10k	$O(N^2)$
≤ 1M	O(n log n)
≤ 100 M	O(n), O(log n), O(1)

Q Take as input x and n.
Calculate the value of the following polynomial.

$[1 \cdot x^n + 2 \cdot x^{n-1} + 3 \cdot x^{n-2} + \ldots n x^1]$

TC → O(n)

left → right ×
right → left ←

$(n-1) x * x + x^2 n$

Q Write a program for checking if a no. is prime or not.

Q Given a no, tell all prime no from 2 → n.

n = 13
output → 2, 3, 5, 7, 11, 13

* Sieve of Eratosthenes

n = 25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

TC → O(n * (log(log(n))))

* Prime decomposition

Q Given a string, count all the palindromic substrings in it.

babad

b	a	b	a	d
ba	ab	ba	ad	
bab	aba	bad		
baba	abad			
babad				

output → 7