Big O notation

Explained with a Company Party Analogy

Imagine you're planning a company party and need to invite all employees.

Let’s say:

- **n = number of employees**

Different ways to handle the invitations represent different algorithms. Big O helps us describe how the **time or effort** grows as the number of employees increases.

Extrimely good

🟢 O(1) - Constant Time

Analogy: You announce, “Party at 6 PM!” during an all-hands meeting.

- ✅ One action, no matter how many people are there.
- ⏱️ Time does **not** increase with more employees.

> **Scalability**: Excellent

🟢 O(log n) - Logarithmic Time

Analogy: You tell one person, who tells two others, who each tell two more, like a tree.

- 👤 Spreads quickly, even as the company grows.
- Extremely efficient.

> **Scalability**: Excellent - ideal for large teams.

Just good

🟡 O(n) - Linear Time

Analogy: You go around and tell each person individually.

- 👥 More employees = more time.
- 10 employees = 10 minutes.
- 100 employees = 100 minutes.

> **Scalability**: Okay, but effort grows with company size.

🟡 O(n log n) - Log-linear Time

Analogy: You break the team into small groups for some characteristic or let's say departments, e.g. managers, sales, IT, etc., then meet with each group one by one.

- 🗂️ Smart compromise between personal and scalable.
- Grows reasonably with size.

> **Scalability**: Good - a balanced approach.

So so

🔴 $O(n^2)$ - Quadratic Time

Analogy: You arrange for every employee to talk to every other employee to coordinate party details.

- 💬 10 employees = 100 conversations.
- 100 employees = 10,000 conversations!
- Effort grows **explosively**.

> **Scalability**: Very poor - avoid this pattern.

🔁 Summary Table

Big O	Party Analogy	Scales Well?
O(1)	One announcement to all	✅ Excellent
O(log n)	You tell one, they tell two others	✅ Very good
O(n)	Tell each person one by one	🟡 Okay
O(n log n)	Group then talk to each	🟡 Good
$O(n^2)$	Everyone talks to everyone	❌ Bad

Examples

Add vs Multiply Complexity

Drop constants and non-dominant

Best, worst and expected cases

BigO / Theta / Omega

Time vs/and Space Complexity

Amortized Time

LogN Runtimes

Recursive Runtimes

Artem Hushcha