Big O notation and Complexity Analysis

Nafis Ul Haque

14 January, 2022

১টি প্রোগ্রাম লিখার পর নিশ্চয়ই আমরা ভাবি যে প্রোগ্রাম টি কতখানি efficient? ইনপুট বাড়ার সাথে সাথে এটি কেমন সময় নিবে চলতে? কতখানি মেমরি নিবে? বুঝাই যাচ্ছে, বিষয় গুলো ইনপুট সাইজের উপর নির্ভর করে। কাজেই উত্তর টা 5 সেকেন্ড বা 100MB এমন ভাবে নির্দিষ্ট করে বলে দেয়া যাবে না। তবে কাজটা কীভাবে করা যায়?

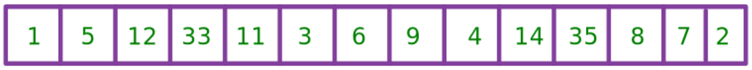
উত্তর হচ্ছে **Big O notation**!

Big O notation মূলত কোনো ১টি অ্যালগরিদম ইনপুট সাইজ n বাড়ার সাথে সাথে worst-case (অর্থাৎ যখন অ্যালগরিদমটি সবচেয়ে খারাপ perform করে) এ কি পরিমাণ অপারেশন নেয় তা n এর ১টি ফাংশন রূপে প্রকাশ করে।

আচ্ছা এখানে worst-case দিয়ে কি বুঝাচ্ছি?

মনে কর তোমার কাছে n টি সংখ্যার array দেয়া আছে, এর মধ্যে থেকে তুমি দেখতে চাচ্ছো 1 সংখ্যাটি আছে কিনা।

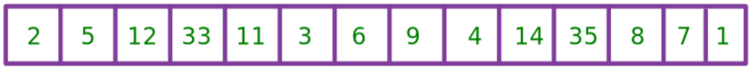
এর জন্য তুমি array এর বাম থেকে একটি একটি করে সংখ্যা চেক করে দেখতে পার, যতক্ষণ না পর্যন্ত 1 পাও। সবগুলো সংখ্যা চেক করার পর ও যদি 1 পাওয়া না যায়, তার মানে 1 আসলে array তেই নেই।



1	5	12	33	11	3	6	9	4	14	35	8	7	2

যদি array টা এমন হয়, তবে আমাদের কপাল অনেক ভালো, array এর শুরুতেই 1 পেয়ে গেছি, array এর পরের উপাদান গুলো আর চেক করতে হচ্ছে না! এরকম ইনপুট এ, অর্থাৎ যা খুঁজছি তা যদি একদম শুরুতেই থাকে, তখন আমাদের এই খোঁজাখুঁজি এর অ্যালগরিদমটি সবচেয়ে ভালো কাজ করবে। তাই এটাকে আমরা অ্যালগরিদমটির best-case বলতে পারি!

এবার worst-case দিয়ে কি বুঝাচ্ছি তাও নিশ্চয়ই বুঝা যাচ্ছে!



2	5	12	33	11	3	6	9	4	14	35	8	7	1

এবার আর আমাদের কপাল এতো ভালো না, এবার 1 খুঁজতে গিয়ে পুরো array এর সব গুলো উপাদান ই আমাদের চেক করতে হচ্ছে! এটি ই হচ্ছে আমাদের এই অ্যালগরিদমটির worst-case!

কোনো অ্যালগরিদম কতখানি efficient তা মাপার একটি উপায় হচ্ছে এটি worst-case এ কিরকম perform করে তা দেখা।

ধরা যাক অনেক হিসাব নিকাস করে আমরা ১টি অ্যালগরিদম worst-case এ কতগুলো অপারেশন নিবে তা n এর একটি ফাংশন $f(n)$ দ্বারা প্রকাশ করলাম। আপাতত ধরে নেই $f(n)$ এমন কিছু,

$$f(n) = n^3 + 2n^2 + 5n + 3$$

কিন্তু এত জটিল একটি ফাংশনের কি দরকার আছে? চিন্তা করে দেখ, n এর মান একটু বড় হলেই $2n^2+5n+3$ অংশটুকু n^3 এর তুলনায় এতই ছোট হয়ে যায় যে এদের হিসাবে না আনলেও কিছু যায় আসে না! n এর মান মাত্র 1000 এর জন্যই পরীক্ষা করে দেখতে পার!

তাই আমরা বলে দিতে পারি, অ্যালগরিদমটি worst-case এ মোটামোটিভাবে n^3 টি অপারেশন নেয়। এটি ই অ্যালগরিদমটির Time Complexity! বিষয়টি Big O Notation দিয়ে এভাবে লিখা হয় - $O(n^3)$ ।

বুঝাই যাচ্ছে, Big O ফাংশনে n এর সবচেয়ে বড় ঘাতটি রেখে বাকি গুলো ফেলে দেয়া হয়। যেমন - $O(n^2+66n-4) = O(n^2)$।
শুধু তাই নয়, আমরা n এর সাথে কোনো constant গুণ আকারে থাকলে তাও ফেলে দেই! যেমন - $O(3n^2) = O(n^2)$ কিংবা $O(5n^3) = O(n^3)$ । কিন্তু কেন? শুরুতেই বলেছিলাম, Big O notation ইনপুট সাইজ n বাড়ার সাথে সাথে ফাংশনটি কেমন দ্রুত বাড়তে থাকবে তা নিয়ে মাথা ঘামায়, কাজেই একটি constant দিয়ে গুন করলে linear ফাংশন linear ই থেকে যাবে, একটি quadratic ফাংশন quadratic ই থেকে যাবে। তাই Big O notation constant নিয়ে মাথা ঘামায় না।

এবার কিছু জিনিস-পাতির Complexity নির্নয় করা যাক!

ধর আমরা 1 থেকে n পর্যন্ত সংখ্যা গুলোর যোগফল নির্নয় করতে চাচ্ছি।

```
int sum = 0;
for(int i = 1; i <= n; i++) {
    sum = sum + i;
}
cout << sum << endl;
```

এখানে যোগফল নির্নয় করতে আমরা ১টি লুপ চালিয়েছি, 1 থেকে n পর্যন্ত। বুঝাই যাচ্ছে, যোগফল নির্নয় করতে মোটামোটি n টি অপারেশন লাগছে, তাই এটির Time Complexity হচ্ছে $O(n)$।

যারা হালকা গণিত জানে তারা নিশ্চয়ই ভাবছে এই কাজটির জন্য তো কষ্ট করে লুপ চালাতে হয় না, ১টি খুব সহজ সূত্রই আছে। 1 থেকে n পর্যন্ত সংখ্যা গুলোর যোগফল হচ্ছে $\frac{n\times(n+1)}{2}$!

```
int sum = (n * (n + 1)) / 2;
cout << sum << endl;
```

এবার দেখ যোগফল নির্নয় করতে কয়টি অপারেশন লাগবে তা আর n এর উপর নির্ভর করছে না, $n = 5$ হলে

যত গুলো অপারেশন লাগবে, $n = 100$ হলেও ঠিক ততগুলোই লাগবে! কাজেই এবার অ্যালগরিদমটি সবসময় constant সময় ই নিবে! Big O notation এ এটিকে লিখা হয় এভাবে - $O(1)$। অর্থাৎ এই যোগফল নির্নয়ের অ্যালগরিদমটির Time Complexity হচ্ছে $O(1)$।

আচ্ছা নিচের code টির Complexity কেমন হবে?

```
for(int i = 1; i <= 8 * n + 4; i++) {
    //some constant time code here
}
```

বুঝাই যাচ্ছে, code টি মোটামোটি $8 \times n + 4$ এর মত অপারেশন নিবে, আমরা constant factor বাদ দিয়ে এর Complexity পাই, $O(n)$।

এবার আরেকটি উদাহরণ দেখা যাক।

```
for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        //some constant time code here
    }
}
```

এখানে ২টি লুপ চলছে, $i = 1, 2, 3, ...n$ প্রতি ক্ষেত্রেই ভিতরে ১টি লুপ 1 থেকে n পর্যন্ত চলবে। মোট অপারেশন দাঁড়াচ্ছে $n \times n = n^2$ টির মত, কাজেই এর complexity $O(n^2)$।

আচ্ছা এবার?

```
for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= i; j++) {
        //some constant time code here
    }
}
```

$i = 1$ এর জন্য j এর লুপ টি 1 পর্যন্ত চলবে, $i = 2$ এর জন্য চলবে 2 পর্যন্ত, একই ভাবে $i = n$ এর জন্য n পর্যন্ত। সব মিলিয়ে অপারেশন হচ্ছে $1 + 2 + ... + n = \frac{n \times (n+1)}{2} = \frac{n^2+n}{2}$ টি। Constant factor $\frac{1}{2}$ এবং n এর ছোট ঘাত বাদ দিলে এর complexity দাঁড়াচ্ছে $O(n^2)$।

Complexity যে সব সময় n এর polynomial হবে এমন কোনো কথা নেই, যদি আমরা কোনো সেট এর সকল উপসেট এর উপর iterate করতাম তবে হয়তো complexity হত $O(2^n)$। আবার যদি n সাইজের সকল permutation চেক করতে হয় কখনো তবে complexity হবে $O(n!)$।

Competitive Programming এর সমস্যা গুলোর constraint দেখেই বুঝে ফেলা যায় কেমন Complexity এর solution লিখতে হবে। মোটামোটিভাবে ধরে নেয়া যায় 1 সেকেন্ডে কম্পিউটার 10^8 টি অপারেশন করতে পারে, তার উপর ভিত্তি করে আমরা এমন ১টি টেবিল দাঁড়া করে ফেলতে পারি!

Constraint on n	Possible complexities
$n \leq 10$	$O(n!), O(n^7), O(n^6)$
$n \leq 20$	$O(2^n \times n), O(n^5)$
$n \leq 80$	$O(n^4)$
$n \leq 400$	$O(n^3)$
$n \leq 7500$	$O(n^2)$
$n \leq 2 \times 10^5$	$O(n\sqrt{n}), O(n \log^2 n)$
$n \leq 5 \times 10^5$	$O(n \log n)$
$n \leq 5 \times 10^6$	$O(n)$
$n \leq 10^{18}$	$O(1), O(\log n), O(\log^2 n)$

উল্লেখ্য, টেবিলটি এখান থেকে নেওয়া হয়েছে।