

Todays Content:

1. Issues with Iteration to compare
2. Asymptotic Analysis
 - a. Big O
 - b. Theta
 - c. Omega
3. How to calculate Big O
4. Issues with Big O
5. Why TLE occurs?
6. Idea from constraints

Note: Iterating \uparrow The Time \uparrow

Con: We can iterate two compare 2 Algo's

Q1: Sort arr[N] elements

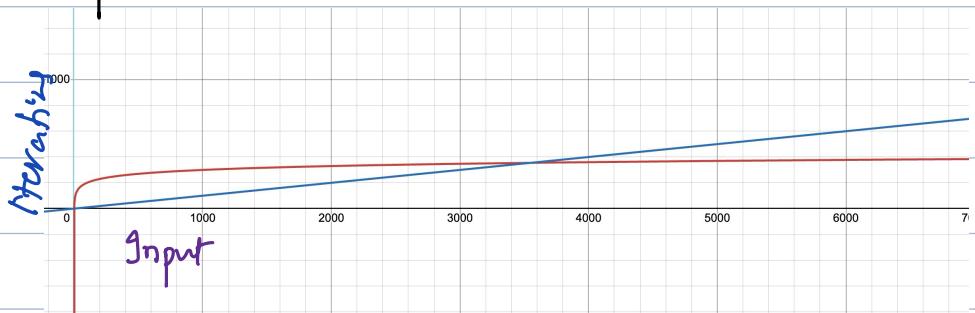
S₁: Gsort

S₂: Nmersort

Code1: $100\log_2^N$

Code2: $N/10$

Graph:



obs: Iterating Efficiency

$N \leq 3500$ Grfan Nazneen } For large inputs we prefer Grfan's code
 $N > 3500$ Nazneen Grfan }

Issue: Comparing with iteration is hectic.

Asymptotic Analysis: Analysing performance of algo for very large inputs

1. Big O
2. Omega
3. Theta.

Big O: 1. What 2. Why 3. How.

How to calculate

1. Calculate iterations based on input

- 2. Neglect lower order terms or consider only highest order term
- 3. Neglect constant coefficients

According to Big O

Code1: $100\log_2 N \rightarrow O(\log_2 N)$ } $O(\log_2 N)$ is more efficient
Code2: $N/10 \rightarrow O(N)$ }

Higher Order Terms:

$$\log_2^N < \sqrt{N} < N < N \log N < N\sqrt{N} < N^2 < N^3 < 2^N < N! < N^N$$

$N=64$

$$\begin{array}{ccccccccc} \log_2^{64} & \sqrt{64} & 64 & 64 + \log_2^{64} & 64 * \sqrt{64} & 64 * 64 & 64^3 & 2^{64} & 64! \\ 6 & 8 & & 64+6 & 64*8 & & & \downarrow & \\ & & & & & & & & \\ & & & & & & & (2^6)^3 & \\ & & & & & & & & \\ & & & & & & & 2^{18} & 2^{64} \end{array}$$

$$Q_1: F(N) = 4N + 3N\log N + 1 : O(N\log N)$$

$$Q_2: F(N) = 4N\log N + 3N\sqrt{N} + 10^3 : O(N\sqrt{N})$$

$$Q_3: F(N) = 4N^2 + 5N\log N + 10^2 : O(N^2)$$

Why Consider only higher order terms?

→ N: Input

Q. Say $f(n) = N^2 + \log N / \text{iterations}$

% lower order terms

Input size:

Total iterations

Iterations fn Total

$$N: 10 \rightarrow N^2 + \log N = 10^2 + 10 * 10 = 200$$

$$\frac{100}{200} * 100\% = 50\%$$

$$N: 10^2 \rightarrow N^2 + \log N = (10^2)^2 + 10 * 10^2 \\ 10^4 + 10^3$$

$$\frac{10^3}{10^4 + 10^3} * 100\% \approx 10\%$$

$$N: 10^4 \rightarrow N^2 + \log N = (10^4)^2 + 10 * 10^4 \\ 10^8 + 10^5$$

$$\frac{10^5}{10^8 + 10^5} * 100\% \approx 0.1\%$$

obs: We neglect lower order terms, because it's contribution is negligible

Neglect Constant Coefficient

Algo1

Algo2

For larger inputs
faster algo is

Q1

$\log_2 N$

N

Algo1

Q2

$100 \log_2 N$

N

Algo1

Q3

$10^2 \log_2 N$

$N/10$

Algo1

Q4

$\log N$

$N^2/10$

Algo1

obs: Constant coefficient won't effect your comparison, hence we neglect them.

Issue 1: Q: Algo1 Algo2

Iterations: $10N$ N^2

$N=5$: 50 25 : Algo2 is efficient

$N=10$: 100 100 : Same

$N=11$: 10×11 11^2 : Algo1 is efficient

$N=12$: 10×12 12^2 : Algo1 is efficient

BigO: $O(N)$ $O(N^2)$

As per BigO Algo1 is efficient than Algo2 for larger inputs.

Issue 2: If 2 Algo have same BigO, we compare iterations.

Q: Algo1 Algo2

Iterations: $10N^2 + N$ $3N^2 + 10N$ Algo2 is efficient

BigO: $O(N^2)$ $O(N^2)$

Note: While calculating BigO: We consider worst case iterations.

Q: Search k in arr[]

bool search(int arr[], int N, int k) { Iteration

for (int i = 0; i < N; i++) {

 if (arr[i] == k) { return true; }

return false;

Best Worst Avg:

1 N

Amortized Analysis

3

Note 2:

Q: Given an arr[] return max of first y elements in array

int maxy(int arr[], int N) { // N > y

int max = INT_MIN;

Iteration: $i = [0, 3] = 4$. \Rightarrow Constant Iteration.

for (int i = 0; i < y; i++) {

BigO = $O(1)$ \Rightarrow Constant Iteration.

 if (arr[i] > max) {

 max = arr[i];

 }

TLE: Time limit exceeded

Online Editors processing speed = 1.6×10^9 instruc~~tion~~ / sec
 $= 10^9$ instruc~~tion~~ / sec

```
int countfactor(int N) {
```

```
    int c=0; → +1
```

```
    for(int i=1; i <= N; i++) { Total Instructions = 3 + 5N
        → +1   → +1   → +1
    } Iteration = 5 instructions
```

```
        → +1   → +1
```

```
        if(N%i == 0) {
```

```
            c++; } +1
```

```
}
```

```
return c; +1
```

Assumption: 1 iteration = 10 instructions.

$$\begin{aligned} \text{Our Code} &= 10^9 \text{ instructions/sec} \\ &= 10^8 \times 10 \text{ instructions/sec} \\ &= 10^8 \text{ iterations/sec} \end{aligned}$$

Online servers = 10^8 iterations/sec

Con: If code encodes $> 10^8$ iterations = TLE.