Todays Content:

1. Issues with Iterations 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

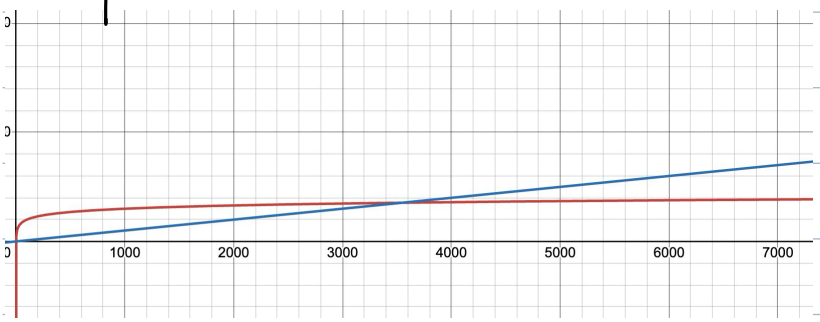
Con: We use iterating two compare 2 Algo's

Q1: Sort ar[N] input elements

S_1 : DSort S_2 : MSort

Code1 : $100 \log_2 N$ Code2 : $N/10$

Graph:



obs1:

$N <= 3500$ Dsort more iterating : Msort is efficient

$N > 3500$ Msort more iterat : Dsort is efficient

Con: For input size increases : Prefer DSort.

Issue: Comparing with iterating is hardy to resolve it

Asymptotic Analysis : Tracking performance of algorithm for large inputs

1. Big O 2. Θ Theta 3. Ω omega.

Steps to Calculate Big O : What? Why? How?

1. Calculate iterations of program based on input
2. Neglect lower order terms or Consider only highest order term
3. Neglect constant coefficients.

According to Big O

Code 1: $100 \log_2^N \rightarrow O(\log_2^N)$ } $\log_2^N$ is much more efficient.

Code 2: $N/10 \longrightarrow O(N)$ }

Order of Terms

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

$N = 64$

$\log_2^{64}$	$\sqrt{64}$	64	$64 \log_2^{64}$	$64\sqrt{64}$	64^2	64^3	2^{64}	$64!$	64^{64}
6	8		$64 * 6$	$64 * 8$					

		Highest Order	Big O
Q1	$F(N) = 4N + 3N\log N + 1$	$3N\log N$	$O(N\log N)$
Q2	$F(N) = 4N\log N + 3N\sqrt{N} + 10^3$	$3N\sqrt{N}$	$O(N\sqrt{N})$
Q3	$F(N) = 4N^2 + 5N\log N + 10^2$	$4N^2$	$O(N^2)$

Why Consider only higher order terms?

$N \rightarrow$ input

Q. Say $f(n) = N^2 + 10N$: Iterating

Input Size:	Total iterations	% lower order terms iterations in Total
$N: 10 \rightarrow N^2 + 10N$	$= 10^2 + 10*10 = 200$	$\frac{100}{200} * 100\% = 50\%$
$N: 10^2 \rightarrow N^2 + 10N$	$= (10^2)^2 + 10*10^2 = 10^4 + 10^3$	$\frac{10^3}{10^4 + 10^3} * 100\% \approx 10\%$
$N: 10^4 \rightarrow N^2 + 10N$	$= (10^4)^2 + 10*10^4 = 10^8 + 10^5$	$\frac{10^5}{10^8 + 10^5} * 100\% \approx .1\%$

obs: If we inc input n, contribution of lower order term decreases, hence it won't effect comparing so we can neglect it.

Neglect Constant Coefficient

	Algo 1	Algo 2	For longer inputs efficient algo
Q1	$10 \log_2^N$	N	Algo 1
Q2	$100 \log_2^N$	N	Algo 1
Q3	$10^3 \log_2^N$	$N/10$	Algo 1
Q4	$10N$	$N^2/10$	Algo 1

obs2: Constant coefficient won't effect your comparing hence we neglect them.

Issue1:

	Q: Algo1	Algo2
Iterations:	10N	N^2
N=10 :	100	100
N=11 :	10*11	11*11
N=12 :	10*12	12*12
BigO :	O(N)	$O(N^2)$

Con: Algo1 is better than Algo2 N>=10

Con: With BigO we can only claim Algo1 is better than Algo2 for large inputs

Issue2: If 2 BigO notations are same go back to iterations.

	Q: Algo1	Algo2
Iterations :	$10N^2+N$	$3N^2+10N$
BigO :	$O(N^2)$	$O(N^2)$

Algo2 is better.

Note1: For BigO: We consider worst case scenario for calculating iterations.

Q: search k in arr[]

```
bool search(int arr[], int N, int k){
    for(int i=0; i<N; i++){
        if(arr[i]==k){ return true; }
    }
    return false;
}
```

Best Worst

Iterations = 1 N.

BigO = O(N)

Note2:

Q: Given an arr[] return max of first 4 elements of array.

```
int max4(int arr[], int N){ // N>4.
    int max = INT_MIN;
    for(int i=0; i<4; i++){
        if(arr[i]>max){
            max = arr[i];
        }
    }
    return max;
}
```

Iterations = 4, → constant iterations, Independent of input

BigO = O(1) constant iterations

TLE: Time limit Exceeded

Online Editm:

In general timelimit = 1 sec

processing speed = 1 GHz = 10^9 instructions/sec

```
int countfactr(int N){
    int c=0;        +1
    for(int i=1; i<=N; i++){      +1   +1   +1
        if(N%i==0){               +1   +1
            c++;                  +1
        }
    }
    return c;       +1
}
```

Total Instructions = 5N+3

1 iteration ≈ 5 instructions

Assumption:

1 iteration ≈ 10 instructions.

Online editm = 10^9 instruction/sec

= 10^8 * 10 instructi/sec

Online editm = 10^8 iteration/sec