Data Structures and Algorithms (DSA)

I claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important.

Bad programmers worry about the code.

Good programmers worry about data structures and their relationships.

Linus Torvalds, 2006

Literature

- **10 Best books in DSA:** https://medium.com/javarevisited/10-best-books-for-data-structure-and-algorithms-for-beginners-in-java-c-c-and-python-5e3d9b478eb1
- **Algoritms:** https://algs4.cs.princeton.edu/home/
- **Princeton Videos:** https://algs4.cs.princeton.edu/lectures/
- **TutorialsPoint:** https://www.tutorialspoint.com/data_structures_algorithms/index.htm
- **Geeksforgeeks:** https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/
- **Progamiz:** https://www.programiz.com/dsa
- **W3schools:** https://www.w3schools.blog/data-structure-algorithm

Some main DSAs

Data Structures	Algorithms
Linked lists	Search
Stacks	Sorting
Queues	Graph/tree traversing
Sets	Dynamic programming
Maps	Regex
Hash tables	Hashing
Search trees	AI

The Big O

O() is the asymptotic notation of the worst case senario.

O() is also called time complexity.

Definition:

$f(x) = O(g(x))$

for $x \to \infty$

It is equivalent with the following postulat:

There is a M $\in$ R and a $x_0 \in R$

such for all $x \geq x_0 \in R$

we have, $|f(x)| \leq Mg(x)$

Example

$f(x) = 6x^4 - 2x^3 + 5$

and,

$g(x) = x^4$

For M $\in$ R sup +

$f(x) = O(g(x)) \rightarrow |f(x)| \leq Mg(x)$

$|6x^4 - 2x^3 + 5| < |6x^4| + |2x^3| + |5|$

$|6x^4 - 2x^3 + 5| < 6x^4 + 2x^4 + 5^4$

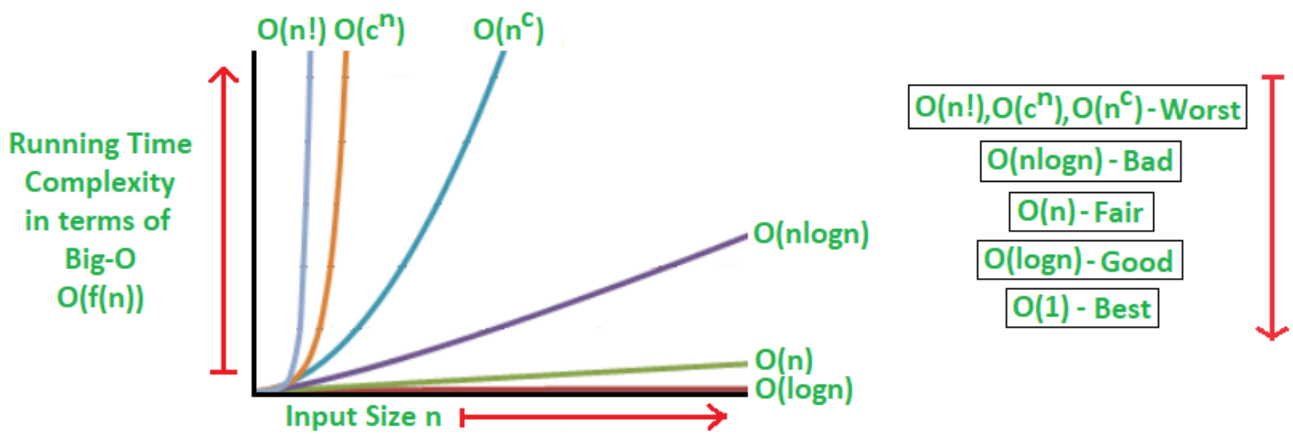
for all, $x \geq x_0 = 1$

$$|6x^4 - 2x^3 + 5| < 13x^4 \rightarrow M = 13$$

Implies that

$$f(x) = O(x^4)$$

constant	O(1)	Best
logarithmic	O(log n)	Good
linear	O(n)	Fair
nlog(n)	O(nlog n)	Bad
potential	O(nˆc)	Worst
exponential	O(cˆn)	Worst
factorial	O(n!)	Worst


O(n!) O(c^n)
O(n^c)
Running Time Complexity in terms of Big-O O(f(n))
O(nlogn)
O(n)
O(logn)
Input Size n
O(n!),O(c^n),O(n^c) - Worst
O(nlogn) - Bad
O(n) - Fair
O(logn) - Good
O(1) - Best

Time complexity of single and nexted loops

The time complexity of a loop is O(N)

A code containing two single loops

Loop1: M times
    O(1) operation
Loop2: N times
    O(1) operation

The teoretical time complexity is O(M+N+1) = O(M+N)

Time complexity of two nexted loops

Outside loop M times
    Inner loop N times
        O(1) operation

The time complexity is O(M*N)

Example i C

```
for(i=0;i<M;i++)
   for(j=0;j<N;j++)
      k=i+j;
```

Time complexity of nexted loops with same iterator

Outer Loop for i=1 to N
  Inner Loop for j=1 to i
    O(1) operation

=> The time complexity is O(N log N)

The time complexity of a loop when the iterator is divied by k is:
O(log_k N)

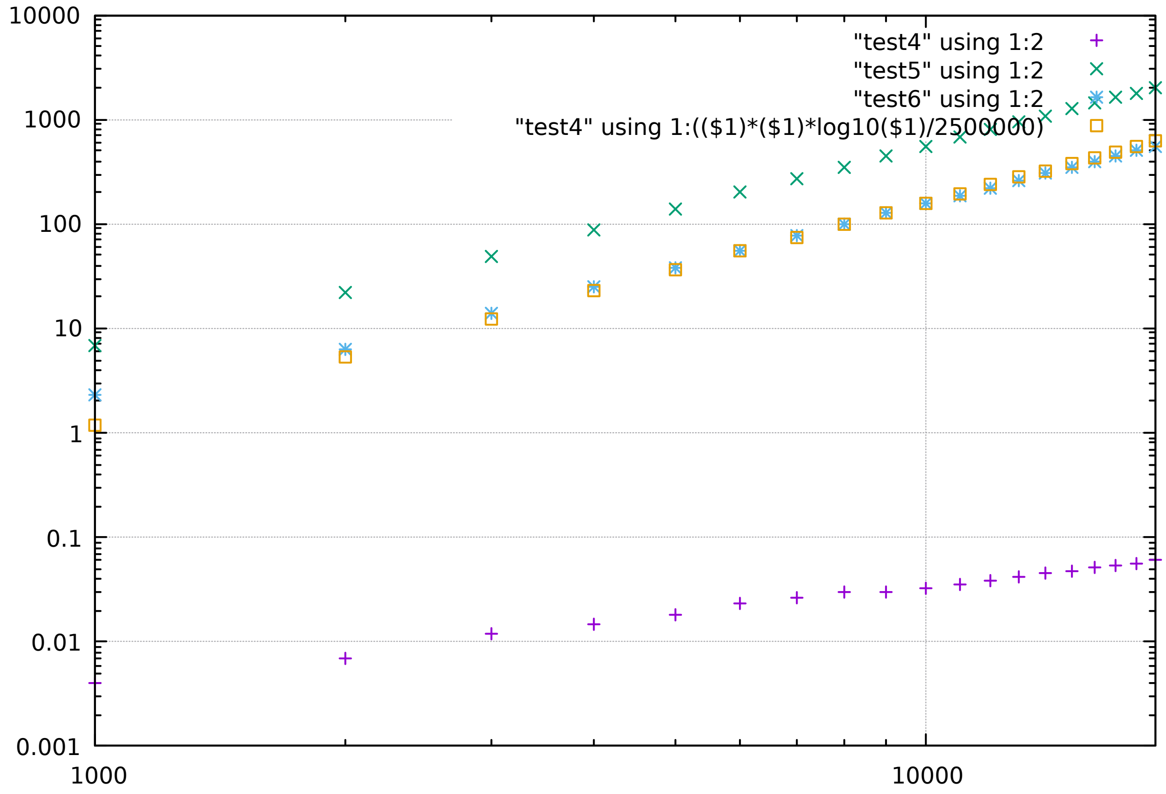
```
int N=8,k=0;
for(i=N/2;i<=N;i++){
    for(j=2;j<=N;j=j*2)
         j=2*j;
         printf("%d ",k);
        k=k+N/2;
```

The inner loop is O(log N) and the outer loop goes N/2 times => O(N/2)

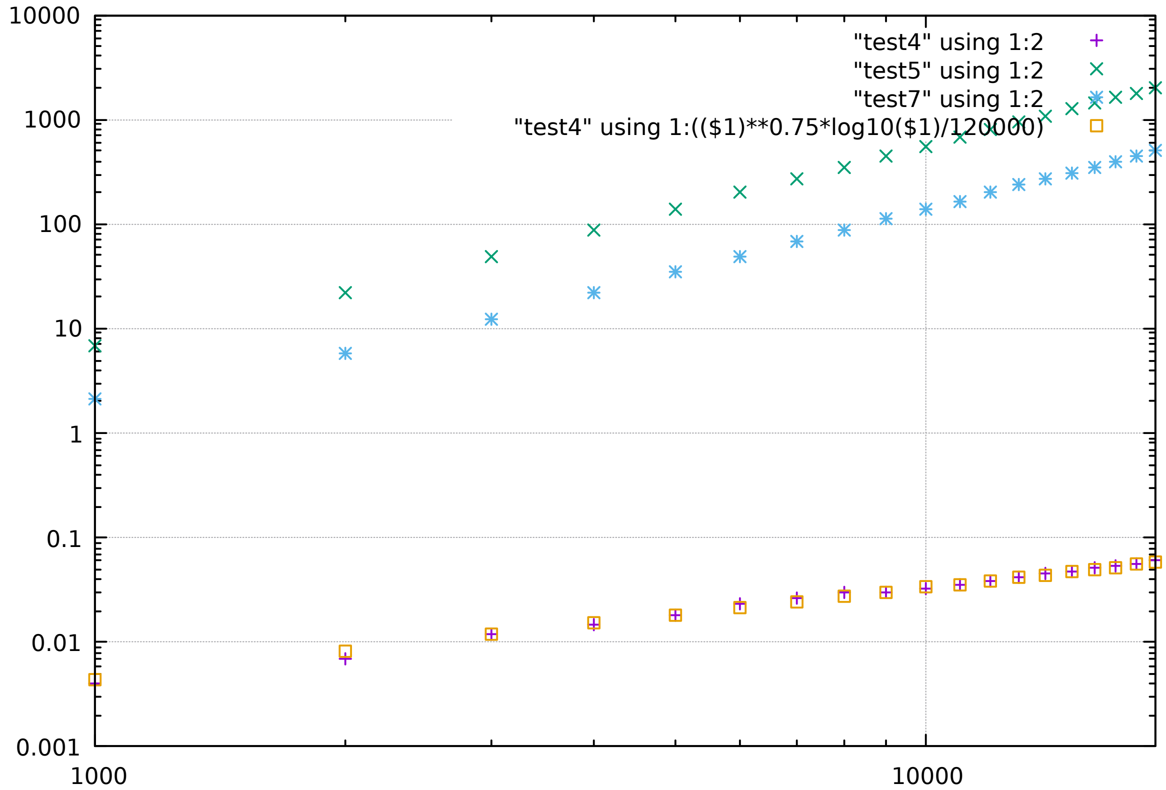
=> The entire algorithm is O(log2 N)

’Hands on’ laborationer

- Using clock() benchmark a function containing loops
- Using Gnuplot plot all results in loglog scales



10000
1000
100
10
1
0.1
0.01
0.001
1000
10000
"test4" using 1:2
"test5" using 1:2
"test6" using 1:2
"test4" using 1:(($1)*($1)*log10($1)/2500000)


10000
1000
100
10
1
0.1
0.01
0.001
1000
10000
"test4" using 1:2
"test5" using 1:2
"test7" using 1:2
"test4" using 1:(($1)**0.75*log10($1)/120000)