Introduction (2)

Monday, February 1, 2021 11:57 AM

Asymptomatic Analysis

Order of growth

1	Scanning, printing	constant
logN	Binary search	logarithmic
Sqrt(N)	Find all divisors of a number	
N	Linear search	Linear
NlogN	Merge sort	Linearithmic
N^2	Matrix addition	Quadratic
N^3	Matrix multiplication	Cubic
2^N	Sum of subsets	Exponential
N!	Permutation of a number	

Polynomial complexity: n, n^2, n^3…
Logarithmic/sublogarithmic: logn, nlogn…

To manage exponential algorithms, we have design techniques:
1. Divide and conquer
2. Backtracking
3. Greedy
4. Dynamic Programming
5. Branch and bound
6. Approximation(reduce accuracy of output)/randomization(pick a random suitable output) algorithms (for algorithms with no solution)
7. NP, NP-hard, NP-complete

1. Divide-and-conquer:

Algorithm can be solved:
a. Recursively (faster)
b. Iteratively/non-recursively

Problem with n (big) data, break into small parts (solve those parts) and merge the solution (if required)

Iterative algorithm: runs n times
Recursive algorithm: base/initial condition (breaks the algorithm) and call the algorithm again with different (reduced) para meters
Ex. Factorial of n

```
Factorial(n)
{
    If(n==1)
        Return 1
    Return n * Factorial(n-1)
}
```

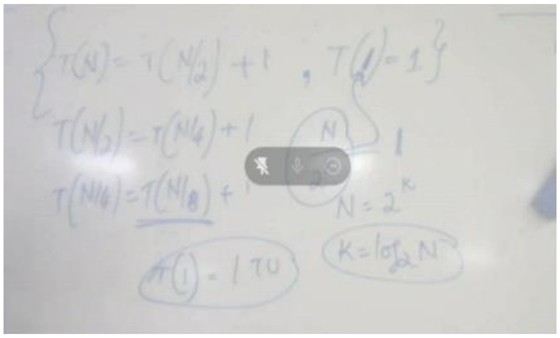
Increase reducing factor to reduce complexity

Binary search: ignore the second half every time a comparison is made. (**recursive algorithm, O(log(2)n)**)
T(N) = T(N/2) + O(1)

$$T(N) = T(N/2) + 1 = T(N/4) + 2 = T(N/8) + 3$$

$$T(N/2) = T(N/4) + 1 \qquad \boxed{T(1) = 1} \qquad \boxed{T(N) = T(N/2^k) + k}$$

$$N/ \quad 1 \quad N = 2^k$$



$$N/2^k = 1 \quad , \quad N = 2^k$$

$$K = \log_2 N$$

$$T(N) = T(1) + k = 1 + \log_2 N$$

$$T(N) = O(\log_2 N)$$

T(N) = T(N-1) + 1 ==> O(N)

Merge sort
Quick sort

X	Best	Average	Worst	Unsuccessful
Binary Search	1	Logn	Logn	Logn
Merge sort				
Quick sort				

Unsuccessful case: algorithms that can detect fail case before the algorithm reaches unsuccessful case complexity