Analysis of Algorithms

Analysis of Algorithms looks at the correctness and efficiency of algorithms

EMPIRICAL ANALYSIS

Implement the algorithm in a programming language, run the program on a computer and see how much time it takes on different inputs

ASYMPTOTIC ANALYSIS

Mathematical techniques for analysizng the scalability of an algorithm on large inputs.

Time Complexity: A function describing how the time taken by an algorithm increases as a function of the size of the input.

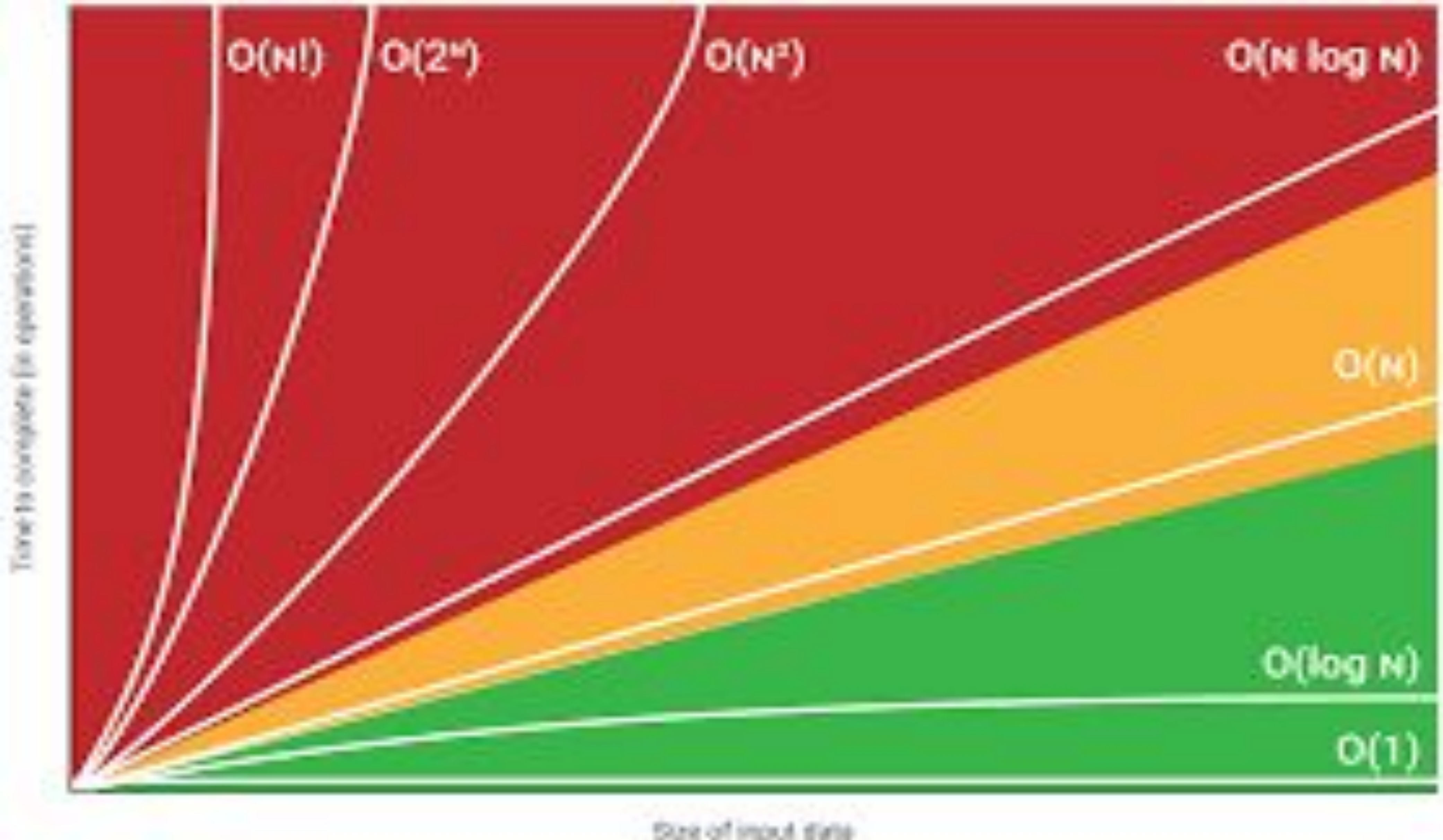
ASYMPTOTIC NOTATION

Big Oh O(n)
Big Omega Ω(n)
Big Theta ϴ(n)

$g(n) = O(f(n))$ iff there exists constants C, n_0 such that, for all $n \geq n_0$ $g(n) \leq Cf(n)$

$g(n) = \Omega$(f(n)) iff there exists constants C, $n0$ *such that for all* $n \geq n_0$ $Cf(n) \leq g(n)$

g(n)=ϴ$(f(n))$ *iff there exists constants* $C_{1,}$ C_2 ,n_0 such that for all $n \geq n_0$ $C_1 f(n) \leq g(n) \leq C_2$f(n)


O(N!)
O(2^N)
O(N^2)
O(N log N)
O(N)
O(log N)
O(1)
Size of input data

The time complexity of BUBLE SORT is

$O\left(n^2\right) \quad where\ n\ is\ the\ number\ of\ elements\ being\ sorted.$

The time complexity of BUBLE SORT is

$O(n^2)$ *where n is the number of elements being sorted.*

```
void bubleSort(std::vector<int>& numbers)
{
    bool swaped=false;
    do
    {
       swaped =false;
    for( size_t
i=0;i<numbers.size()-1;++i)
    {
        if(numbers[i]>numbers[i+1])
        {
          exchange(numbers,i,i+1);
          swaped=true;
        }
    }
    }while(swaped);
}
```

The time complexity of Selection Sort is

$O\left(n^2\right) \quad where\ n\ is\ the\ number\ of\ elements\ being\ sorted.$

The time complexity of Insertion Sort Sort is $O(n^2)$ $where\ n\ is\ the\ number\ of\ elements\ being\ sorted.$

Given an array with n items, describe an algorithm that determines if a given item x is in the array or not.

LINEAR (SEQUENTIAL) SEARCH : In the worst case , x is not in the array => $T(n) = O(n)$

DIVIDE AND CONQUER

The array is sorted => We can use Binary Search (Bisection Method). Compare x to the item y in the middle of the array :

- if x==y, found return
- Else if x <y : Search in the lower half of the array
- Else Search in the upper half of the array

Analysis of Divide and Conquer Algorithms

DIVIDE : BREAK THE PROBLEM INTO SIMILAR SMALLER PROBLEMS

CONQUER : SOLVE THE SMALLER PROBLEMS

COMBINE : PUT TOGETHER THE SOLUTIONS INTO AN OVERALL SOLUTION

Let the time complexity of Binary Search be

$$T(n) = T\left(\frac{n}{2}\right) + C$$

Analysis of Divide and Conquer Algorithms

DIVIDE : BREAK THE PROBLEM INTO SIMILAR SMALLER PROBLEMS

CONQUER : SOLVE THE SMALLER PROBLEMS

COMBINE : PUT TOGETHER THE SOLUTIONS INTO AN OVERALL SOLUTION

MERGE SORT

MERGE TWO SORTED ARRAYS :
time complexity is $O(n)$

1 4 7
9

2 3 5 6 8

Your text here

QUICK SORT

BEST CASE : Each choice of the pivot divides the array into 2 equal subarrays => $T(n) = 2T\left(\frac{n}{2}\right) + Cn \Rightarrow T(n) = O(n \log n)$

WORST CASE : Each choice of the pivot removes just one element => $T(n) = T(n-1) + Cn \Rightarrow T(n) = O(n^2)$

AVERAGE CASE : Consider the expected time complexity over all possible inputs (to be considered later)

THE MASTER THEOREM PROVIDES A GENERAL METHOD OF SOLVING SOME OF THE MOST COMMON RECURRENCE RELATIONS (not all)

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

Where is a is the number of subproblems $a \geq 1$

$T(\frac{n}{b})$ is the time complexity of each sub problem $\frac{n}{b} \geq 1$

f(n) represents the work done out of the recursive calls

The Master theorem does not apply to :

- The recurrence has non constant coefficients $T(n) = n * T\left(\frac{n}{2}\right) + n^2$
- The recurrence relation s multiple different terms $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + n$
- Recurrence relations in which f(n) is not polynomial $T(n) = 2T\left(\frac{n}{2}\right) + 2^n$

THE MASTER THEOREM PROVIDES A GENERAL METHOD OF SOLVING SOME OF THE MOST COMMON RECURRENCE RELATIONS (not all)

$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

Where is a is the number of subproblems $a \geq 1$

$T(\frac{n}{b})$ is the time complexity of each sub problem $\frac{n}{b} \geq 1$

f(n) represents the work done out of the recursive calls

CASE 1:if $f(n) = O(n^{\log_b a - \mathcal{E}})$ then $T(n) = \Theta(n^{\log_b a})$ $Where\ \mathcal{E}$ >0

CASE 2 : if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

THE MASTER THEOREM PROVIDES A GENERAL METHOD OF SOLVING SOME OF THE MOST COMMON RECURRENCE RELATIONS (not all)

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

Where is a is the number of subproblems $a \geq 1$

$T(\frac{n}{b})$ is the time complexity of each sub problem $\frac{n}{b} \geq 1$

f(n) represents the work done out of the recursive calls

CASE 3 : if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ then $T(n) = \Theta(f(n))$

Provided the regularity condition holds :

$$af\left(\frac{n}{b}\right) < cf(n)\ \ for\ some\ c < 1\ and\ sufficiently\ large\ values\ of\ n.$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 7T\left(\frac{n}{4}\right) + n^2$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$T(n) = 3T\left(\frac{n}{2\hat{}\{\frac{1}{2}\}}\right) + n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 5\,n\log n$$

GIVEN TWO ARRAYS OF SIZE n DESCRIBE AN ALGORITHM THAT DETERMINES IF THEY HAVE ANY ELEMENT IN COMMON. What is the time complexity of the algorithm you have given slido

Sort Array B. For each element x in A, use Binary Search to find if x is in B. ? what is the time complexity of this algorithm

ⓘ Start presenting to display the poll results on this slide.

slido

For each element x in A, use linear search to find x in B.
What is the time complexity of this algorithm?

ⓘ Start presenting to display the poll results on this slide.

WHAT IS THE TIME COMPLEXITY OF THE DIFFERENT ALGORITHMS FOR THE MAJORITY ELEMENT PROBLEM

GIVEN AN ARRAY OF SIZE N, DESCRIBE AN ALGORITHM THAT FINDS IF THERE ARE TWO ELEMENTS IN THE ARRAY SUMMING TO x.

GIVEN AN ARRAY OF SIZE n, DESCRIBE AN ALGORITHM THAT GIVES THE k BIGGEST ELEMENTS IN THE ARRAY. WHAT IS THE TIME COMPLEXITY OF THE ALGORITHM PROPOSED?