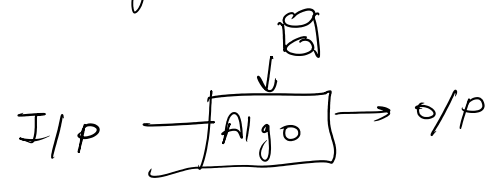
Algorithm Analysis



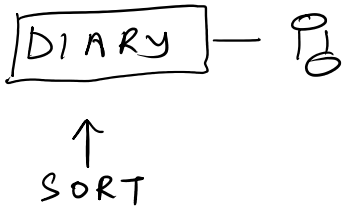
- well defined procedure
- computer/prog. language independent

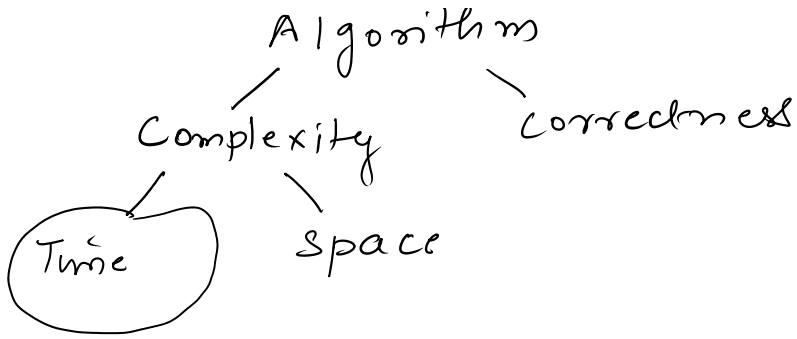
#g Sorting – array/Linked list
Searching – search 'x' array/LL
} Java or C++

Data structure – way to organize/store the data
- facilitate efficient access/update

#g 100 Names to Sort → Diary

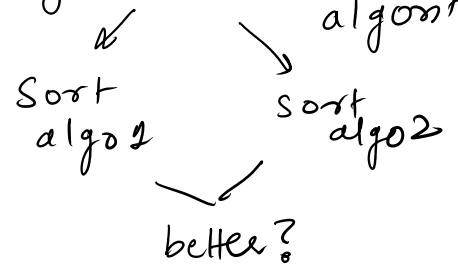
1. Any Order
2. Alphabetic
} DS





complexity Analysis

— comparing performance of different algorithms



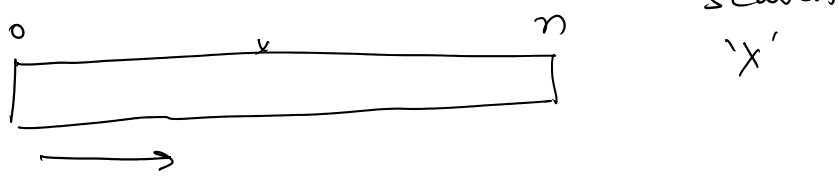
? why cannot we do real analysis

{CPU, DISK, OS, } Too many parameters

Theoretical Analysis. — Time

Big OH Big Theta Big Omega } CS 330

Linear Search

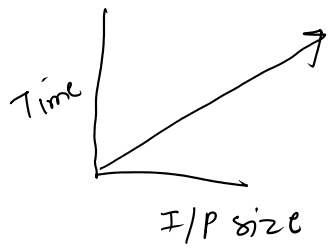


int LS (int * array, int size, int val)

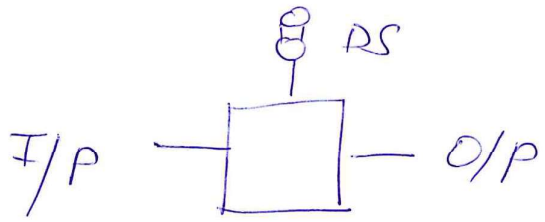
Best - 0pos

Worst - last pos

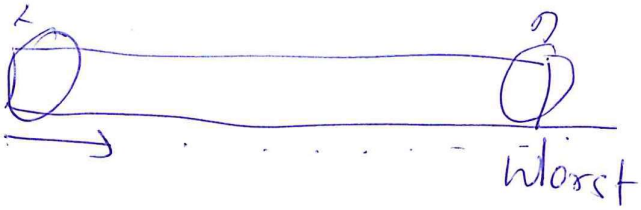
```
{   i = 0;
    while (Val != array[i])
        i++;
    return i
}
```



ALGORITHM ANALYSIS



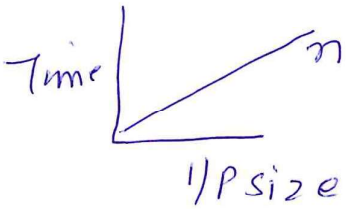
I LINEAR SEARCH



Best

$n = 10$

$n = 10000$

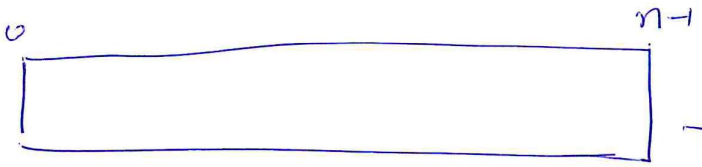


#8

3	5	7	9	11	13

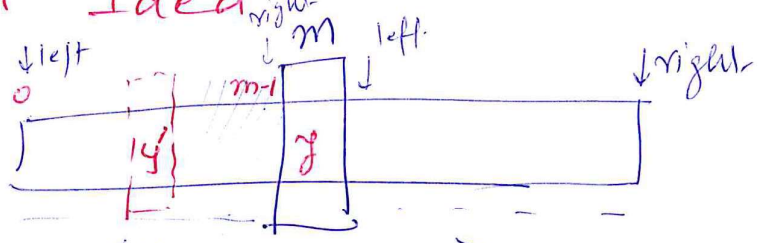
what if we have a sorted array can we improve complexity?

BINARY SEARCH



— n elements sorted fashion

— Main Idea



'x'

$< y$ $> y$. $x < y$.

$x > y$' (value == A[m]) → found the value

if (value < A[m])

{ // Do Binary search on left sub array

}

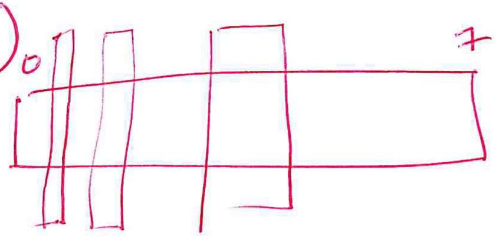
else

{ // Do Binary search on right subarray

}

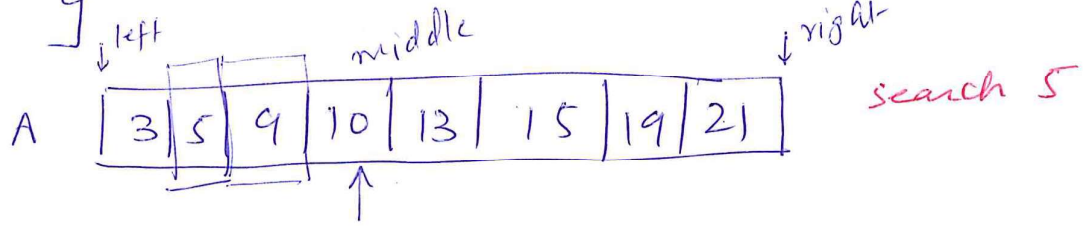
$O(\log n)$

$\left(\frac{n}{8}\right) = 2^{\lceil 3 \rceil}$ →



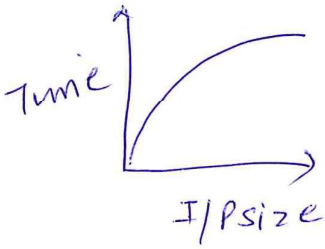
```
int BS ( int * array, int size ; int val)
{
    int (size < 1)   return -1;
    int left = 0;    right = size - 1;
    while (right >= left)
    {
        middle = (left + right) / 2.
        if (val == array [middle])
            return middle.
        else if (val < array [middle])
        {
            right = middle - 1
        }
        else.
            left = middle + 1
    }
```

#g



3 iterations — 5

log (n) → I/P size



THEORETICAL ANALYSIS

↓

Size of the problem — I/P

- constant time
- variable time

Big OH → worst case

— upper bound for complexity of algorithm

Eg 1) int a = 0 // constant time O(1)

2) for(int j=0; j<n; j++) // n times O(n)
= if (j==5) return;

3) for(int i=0, i<n; i++)
{ {
for(j=0; j<n; j++) O(n²)

a = 5+i // n² times

} }

$T(n) = c_1 + c_2 n + c_3 n^2$

foo(n) → Time complexity, — A + B + C

```
{
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)          A - C1.n^2
            a[i][j] = i*j
    }

    for(k=0; k<n; k++)              B - C2.n
    {
        v1[k] = 0
        v2[k] = 0
    }

    int a1=0, b1=0, c1=0            C - C3
}
```

$T(n) = A + B + C = C_1 n^2 + C_2 n + C_3$

$\Downarrow$

$O(n^2)$

$f_1(n) = 12n^2 + 120n + 402$.

n	$f_1(n)$	n^2	n^2 of $f_1(n)$
10	220	100	45.45%
100	10,300	10,000	97.087%
1000	1,002,100	1,000,000	99.790%
10,000	1,00,020,100	1,00,000,000	99.98%

$f_2(n) = 40n^2 + 100n + 372$

n^2 - Dominant Term.

Our focus. → $f(n)$ — in terms of n / n^2 / $n\log n$ / $\log n$

Formal Definition — $O(g(n))$ Big O

We say $f(n)$ is in $O(g(n))$ aka Oh of $g(n)$

means: $\exists\ n_0 > 0$ & $c > 0$ s.t.

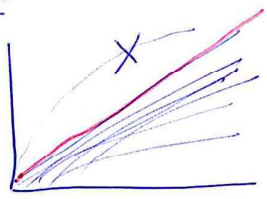
$\forall\ n > n_0$ (Problem size) $\quad f(n) \le cg(n)$ (Algorithm)

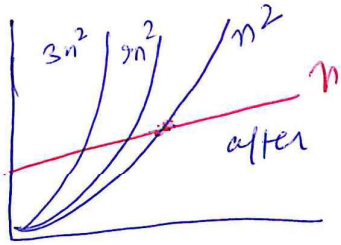
running time is always some multiple of $g(n)$

---

9 $T(n) = 8cn + 2$.

$O(n)$





after this pt $T(n)$ is definitely $< cn^2$

$\Rightarrow T(n) = O(n^2)$

How to determine $g(n)$?

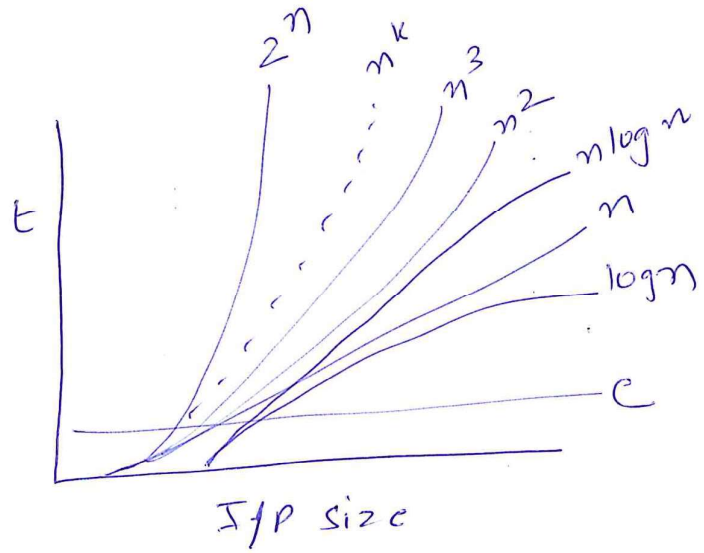
1. Determine $T(n)$ "the running time"

   #g $8n^2 + n\log n + n$

2. Drop all but the most significant term. $8n^2$

3. Drop the constant coeff.

   $\Rightarrow O(n^2)$.



$\Omega(g(n))$

lower bound

$O(g(n))$

tighter bound