→ Share a file with your friend.

↓

PC.

↓

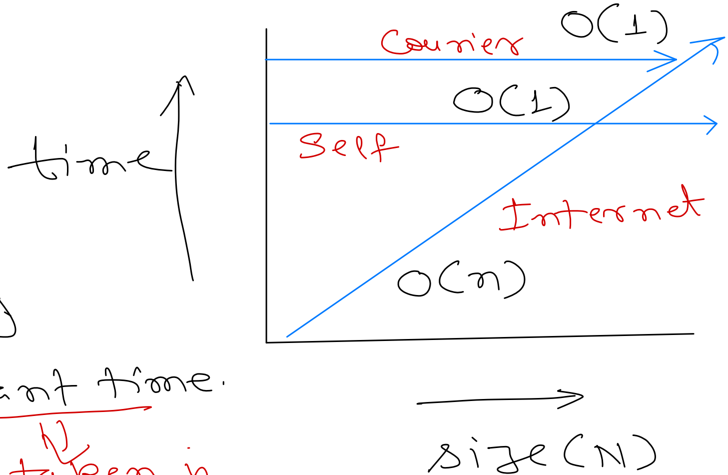
internet ⇒ Time taken? Depend of size

Physical medium → Courier
→ Self

Big O

O( )

Asymptotic Time Complexity



O(1) → <u>Constant time.</u>

Time taken is independent of size of data set / input.

O(n) → Linear time.

→ Find min & max of N numbers

```
int elem[..];
int min, max;
min = max = elem[0];                  → 1
for i → 1 to n-1                      → (n-1-1+1) ⇒ times
  if (elem[i] < min)                  → 1
     min = elem[i];                   → 1
```

```
if (elem[i] > max)      → 1
    max = elem[i]       → 1
```

$1 + 4 * (n - 1 - 1 + 1)$

$= 4n - 3$

→ Remove constants

$= n \Rightarrow O(n)$

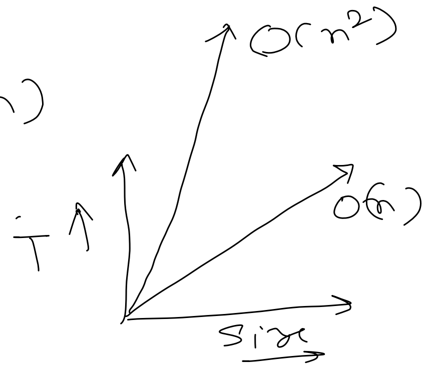
```
int elem[..];
int min; max;
min = max = elem[0];         → 1
for i → 1 to n-1             → (n-1-1+1)
    if elem[i] < min         → 1
        min = elem[i];       → 2
for i → 1 to n-1             → (n-1-1+1)
    if (elem[i] > max)       → 1
        max = elem[i];       → 1
```

$1 + 2 * (n - 1 - 1 + 1) + 2 * (n - 1 - 1 + 1)$

$= 2n + 2n - 3$

$= 4n - 3 \Rightarrow O(n)$

$1 + 2n - 2 + 2n - 2$



for $i \to 0$ to $n-2$ → $(n-2+1)$

$\quad$ ══ → 1

$\quad$ ══ → 1

$\quad$ ══ → 1

$\quad$ for $j \to 1$ to $n-1$ → $(n-1-1+1)$

$\quad\quad$ ══ → 1

$\quad\quad$ ══ → 1

$$\left(3 + 2 * (n-1-1+1)\right) * (n-2+1)$$

$$= (2n+1) * (n-1)$$

$$= 2n^2 - 2n + n - 1$$

$$= 2n^2 - n - 1$$

→ Remove constants

$$= n^2 - n$$

As n increases to larger value

$$(n^2 - n) \approx n^2$$

→ Pick n with highest power

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

Searching

→ Linear Search ⇒ Find a paper in a group of random papers.

⇓

Inspect each paper one by one, starting from first until either found or not found.

→ Binary Search

⇓

Find word meaning in dictionary book.

Requires data to be arranged / sorted.

Can be done on data that is either arranged (sorted) or unarranged (unsorted)

Linear Search

→ Find (elem) : Linear Search

→ for i → 0 to n − 1

→ if (elem = arr[i]) ← 1

→ return found; ← 1

→ return not found; → 1

0	1	2	3	4
5	9	1	3	2

N = 5

elem → 3

$$\underbrace{n-1-0+1}_{(n)} * 2 + 1$$

$= 2n+1 \Rightarrow O(n)$

Time Complexity of Linear Search.

0 1 2 3 4

Binary Search

1	2	5	9	15

elem < 5   middle   elem > 5   N = 5

→ BinarySearch(elem)

→ left = 0 → 1

→ right = n−1 → 1

→ while (left <= right)

→ mid = (left + right)/2

→ if (elem == arr[mid])

return found.

→ if (elem < arr[mid])

right = mid − 1

else

left = mid + 1.

→ return not found. → 1

left → ~~0~~ ~~3~~ 5

right → 4

mid → ~~2~~ ~~3~~ 4

elem → 15

20

(left > right)

when loop should end

(left <= right) ⇐ !(left > right)

↳ when loop should run

N elements : Start

N/2 elements After 1st iteration

N/4 " After 2nd iteration

N/8 " After 3rd iteration

1 element After ___ iterations

$N \to N/2 \to N/4 \to N/8 \to N/16 \;\ldots\; 1$

$$N/2 \Rightarrow N/2^1 \qquad N/4 \Rightarrow N/2^2 \qquad N/8 \Rightarrow N/2^3 \qquad N/16 \Rightarrow N/2^4$$

$$\log_2 N$$

N 2

10,000 $\qquad \log_{10} 10{,}000 = 4$

$$\frac{10{,}000}{10} = 1000 \;—\; 1$$

$$\frac{1000}{10} = 100 \;—\; 2$$

$$\frac{100}{10} = 10 \;—\; 3$$

$$\frac{10}{10} = 1 \;—\; 4$$

$$\log_2 16 = 4$$

$$\frac{16}{2} = 8 \;—\; 1$$

$$\frac{8}{2} = 4 \;—\; 2$$

$$\frac{4}{2} = 2 \;—\; 3$$

$$\frac{2}{2} = 1 \;—\; 4$$

$3 + 6 * (\log n)$

$= 6 \log n + 3$

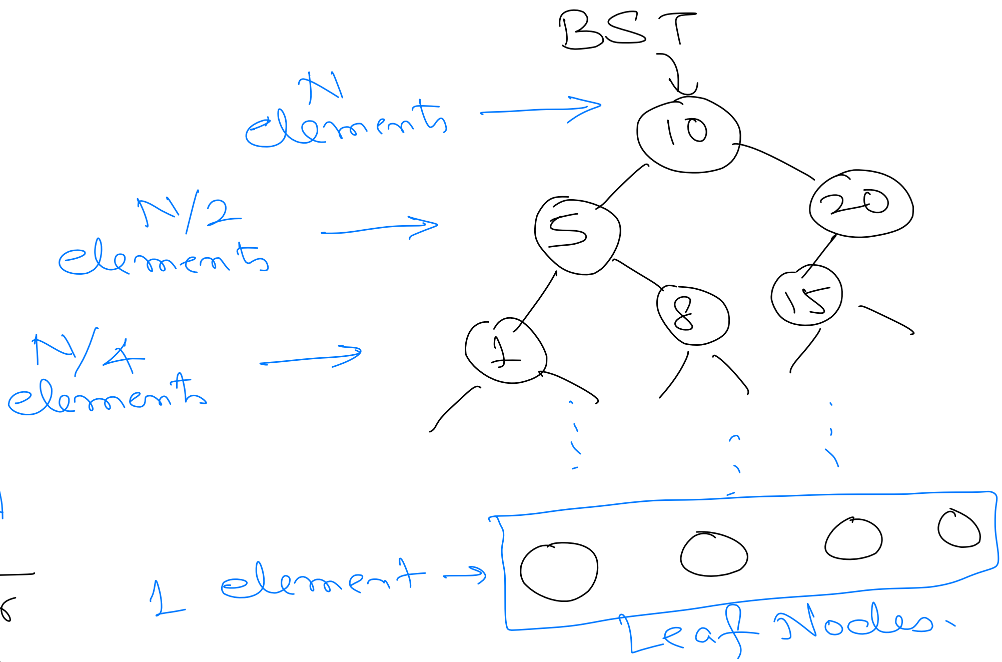
$\rightarrow$ Remove constants

$= \log n \Rightarrow O(\log n)$

Binary Search $= O(\log_2 n)$

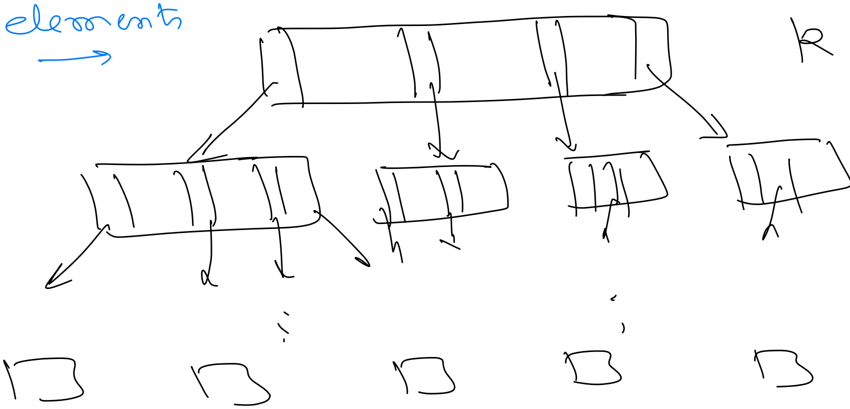
Ternary Search $= O(\log_3 n)$

K-ary Search $= O(\log_k n)$



$\log_2 N$
___
Number of levels in a perfect binary tree with N nodes.

N elements →



K - Keys per node

← $\frac{N}{K}$ elem

Levels = $(\log_K n)$

← leaf

Search in a node = K

Iteration count = $\log_K n$

$K * \log_K n$

Time complexity = $O(K \log_K n)$

↳ Search in multi way Search tree of order K

$K \underline{\ll} n$

Very small

$O(\log_2 n)$

$O(K \log_K n)$

Sorting

→ Bubble Sort

1. To sort in increasing order

bring smallest element to front in Each iteration, of the elements left

OR

bring largest element at rear in Each iteration, of the elements left

~~3~~ 1   ~~1~~ 3   ~~5~~ 2   ~~2~~ 5

Left < right if NOT then swap

OR

right < left

$\boxed{1}$ ~~3~~ 2   ~~2~~ 3   5

$\boxed{2}$ 3   5

$\boxed{3}$ 5

N = 4

0	1	2	3
3	1	5	2

→ Bubble Sort

for PutElemAt → 0 to (n-2)

for right → (n-1) to (putElementAt + 1)

→ left = right - 1

→ $\boxed{\text{if ! (elem[left] < elem[right])}}$

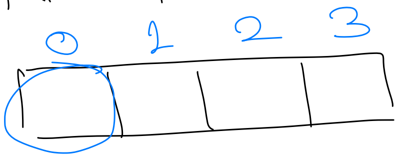
if (elem[right] < elem[left]) → Swap left & right elements.

BubbleSort(elements, n) // ‘n’ number of values in ’elements’.
/* Start by putting correct element at 1st position,
then 2nd position then 3rd position, until (n-1)st position.
Once we put (n-1) elements at correct positions,
nth element will be at correct position automatically.
*/

- for putElementAtPos -> 0 to (n - 2) → (n-2) - 0 + 1
  // Start with last element until before the putElementAtPos
  - for right -> (n -1) downto (putElementAtPos + 1) → (n-1) - (PEAP+1)+1
    // Compare right and left element
    - left = right - 1 → 1
    // If right element is smaller than left then swap them.
    - if element[right] < element[left] then → 1
      - Swap left and right elements. → 1
- Stop.

putElementAtPos	# times for inner loop
0	$(n-1) - (0+1) + 1 = (n-1)*3$
1	$(n-1) - (1+1) + 1 = (n-2)*3$
2	$(n-1) - (2+1) + 1 = (n-3)*3$
⋮	⋮
$(n-2)$	$(n-1) - (n-2+1) + 1 = 1*3$

N = 4



Iteration 1

putElementAt → 0

right → 3 Compare (3, 2)

2 Compare (2, 1)

1 Compare (1, 0)

(n−1)

$(n-1) - (n-2+1) + 1$

$\cancel{n} - 1 - \cancel{n} + 2 \cancel{-1} \cancel{+1}$

$= 1$

Iteration 2

putElementAt → 1

right → 3 Compare (3, 2)

2 Compare (2, 1)

putElementAtPos → # times inner loop runs

0 → $(n-1)*3$

1 → $(n-2)*3$

2 → $(n-3)*3$

⋮ ⋮

$(n-2)$ → $1*3$

Sum

$1*3 + \ldots (n-3)*3 + (n-2)*3 + (n-1)*3$

$3*[\underbrace{1 + \ldots + (n-3) + (n-2) + (n-1)}_{\text{First } (n-1) \text{ numbers}}]$

Sum of first N numbers

$$= \frac{n(n+1)}{2}$$

$$= 3 * \left[\frac{(n-1)((n-1)+1)}{2}\right]$$

$$= 3 * \left(\frac{(n-1)(n)}{2}\right)$$

$$= \frac{3}{2} \times (n^2 - n)$$

→ Remove constants

$$= n^2 - n$$

→ Pick n with highest power

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