Time Complexity

```
for (i=0; i<n; i++)
{   for(j=0; j<i; j++)
    {
        stmt;
    }
}
```

i	j	no. of times
0	0 ✗	0
1	0 ✓ 1 ✗	1
2	0 ✓ 1 ✓ 2 ✗	2
3	0 ✓ 1 ✓ 2 ✓ 3 ✗	3
⋮	⋮	
n		n

$$1+2+3+\dots+n = \frac{n(n+1)}{2}$$

$$f(n) = \frac{n^2+n}{2}$$

$$O(n^2)$$

Ex:

```
p=0;
for (i=1; p<=n; i++)
{
    p=p+i;
}
```

i	p
1	0+1 = 1
2	1+2 = 3
3	1+2+3
4	1+2+3+4
5	1+2+3+4+5
⋮	
k	1+2+3+...+k

$1+2+3+\dots+k = \frac{n(n+1)}{2}$

Assume $p > n$

$\because p = 1+2+3+\dots+k$

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

$$\frac{k^2+k}{2} > n$$

$$k^2 > n$$

$$k > \sqrt{n}$$

$O(\sqrt{n})$ ✓

Ex

```
for(i=1; i<n; i=i*2)
{
        stmt;
}
```

i
i = 1
= 1×2 = 2
= 2×2 = 2^2
= $2^2 \times 2 = 2^3$
= $2^3 \times 2 = 2^4$
⋮
= 2^k

Assume $i >= n$

$\because$ $i = 2^k$

$2^k >= n$

$2^k = n$

$k = \log_2 n$

$O(\log_2 n)$

```
for (i=0; i<n; i++)          —— n+1
{  for (j=1; j<n; j=j*2)     — n(log n)
   {
         stmt;
   }
}
```

$f(n) = n+1 + n(\log n)$

$i = 1 \times 2 \times 2 \times 2 \cdots = n$

$2^k = n$

$k = \log_2 n$ $\quad O(\log_2 n)$

```
for (i=1; i<n; i=i*2)
  {   stmt;
  }
```

$O(\log_2 n)$

```
for (i=1; i<n; i=i/2)
  {   stmt;
  }
```

i/n

$n/2$

$n/2^2$

$n/2^3$

$\frac{n}{2^k}$

Assume $i < 1$

$$\frac{n}{2^k} < 1$$

$$\frac{n}{2^k} < 1$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

$O(\log_2 n)$

```
p=0
for (i=1; i<n; i=i*2)
  {   p++;  }
for (j=1; j<=p; j=j*2)
  {   stmt;  }
```

Imp

$p = \log n$

$\log p$

$\log n \log n \rightarrow O(\log \log n)$

$\log_n \frac{\log n}{n}$

```
for (i=0; i<n; i++)
```

$O(n)$

for (i=0; i<n; i=i+2) —— O(n)
n/2 O(n/2) = O(n)

for (i=n; i>1; i--) —— O(n)

for (i=1; i<n; i=i*2) —— $O(\log_2 n)$

for (i=1; i<n; i=i*3) —— $O(\log_3 n)$

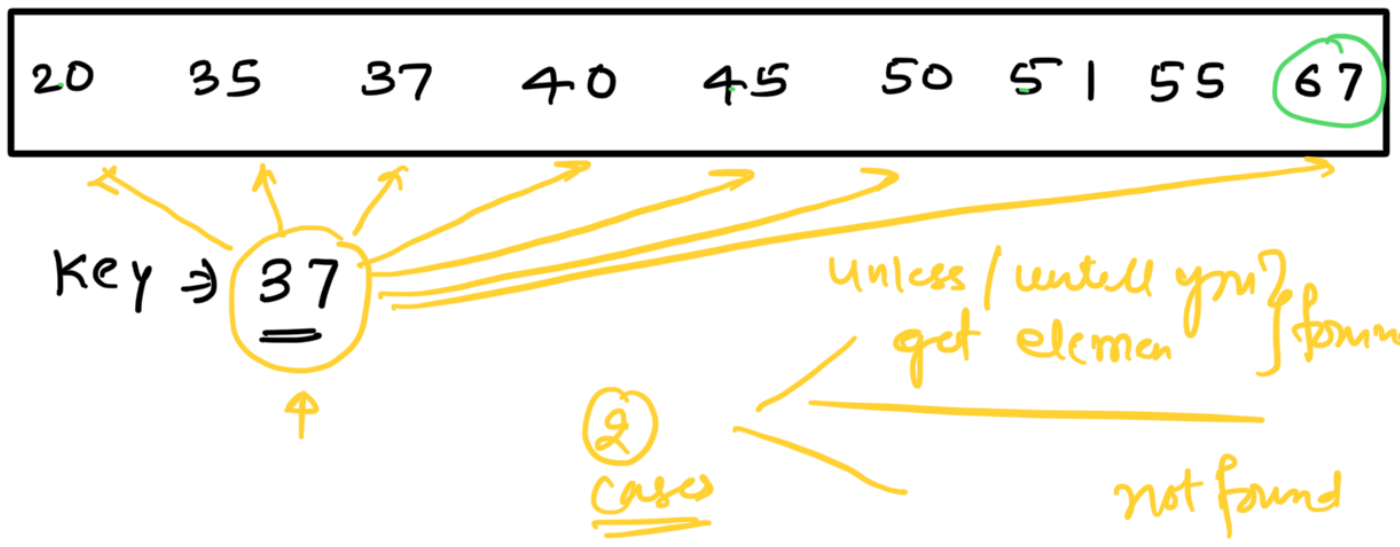
for (i=1; i>1; i=i/2) —— $O(\log_2 n)$

Searching Algorithms -

Searching - find out some location for a particular element

1. Linear Search Algorithm
2. Binary Search Algorithm

Linear Search -



for (int i = 0; i<n; i++) → n+1

{ if (arr[i] == key) ✓ → 1 — case 1 — found

n → Comparisons

case 2 — not found }

Time Complexity

Case 1 ⇒ Key = 20 — 1 comp — min = 1 (Best case)

Case 2 ⇒ Key = 67 — n comp — max = n (Worst case)

Element not found → Worst case

Average case = $\frac{\text{all possible case time}}{\text{no. of cases}}$

Average time ⇒ $\frac{\frac{n(n+1)}{2}}{n} \Rightarrow \boxed{\frac{n+1}{2}}$

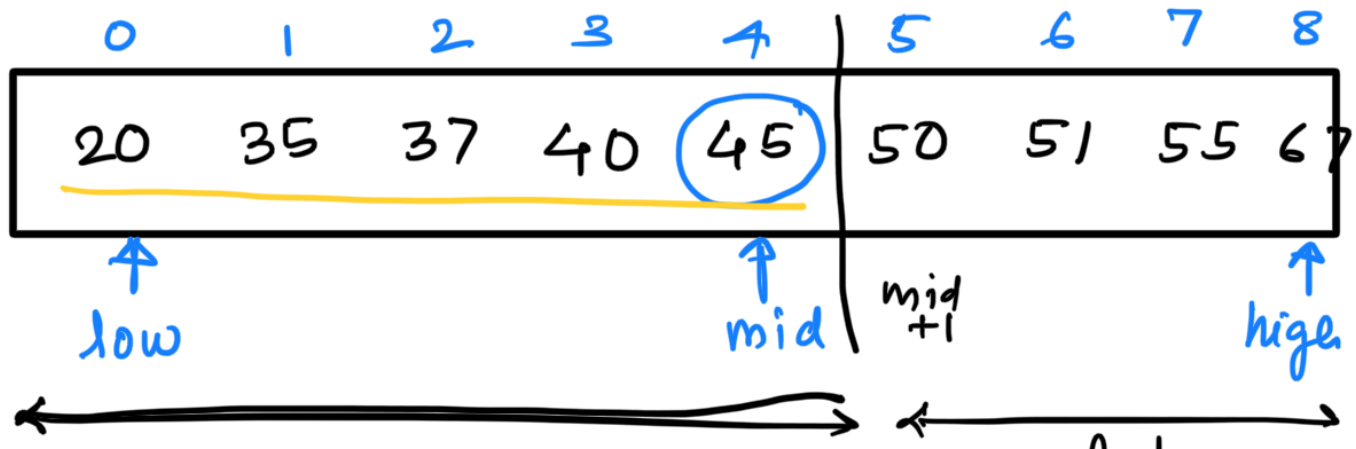
$\Omega(n) = \Omega(1)$

$O(n) = O(n)$

$\Theta(n) = \Theta\left(\frac{n+1}{2}\right) \Rightarrow \Theta(n)$

Best, Worst and Average case Analysis -

Binary Search -



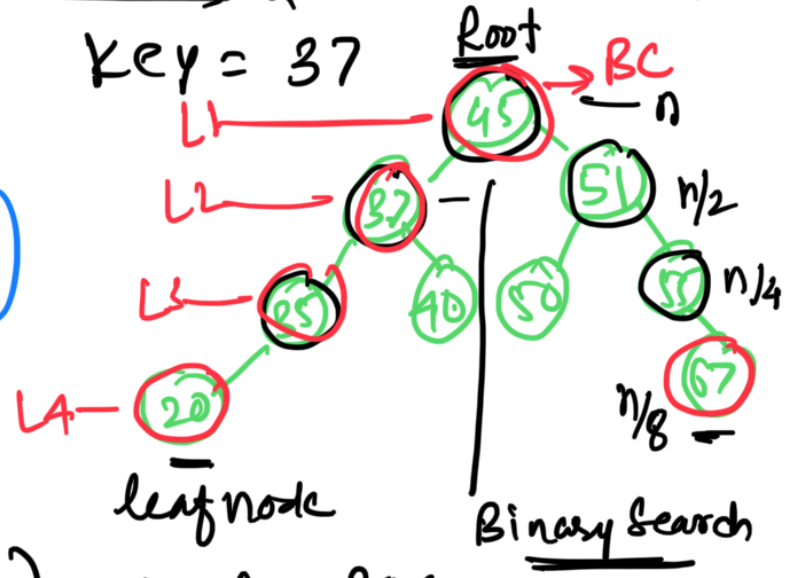
(1) low = 0

high = 8

mid = $\frac{0+8}{2}$ ($\frac{\text{low} + \text{high}}{2}$)

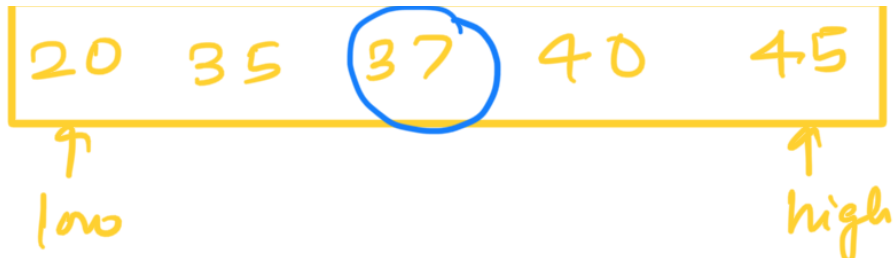
$= \frac{8}{2} = 4$

Key = 37



(2) if (mid & Key) → compare

n = 1K

① 

$\frac{low+high}{2} = \frac{0+4}{2} = 2$

mid = 2

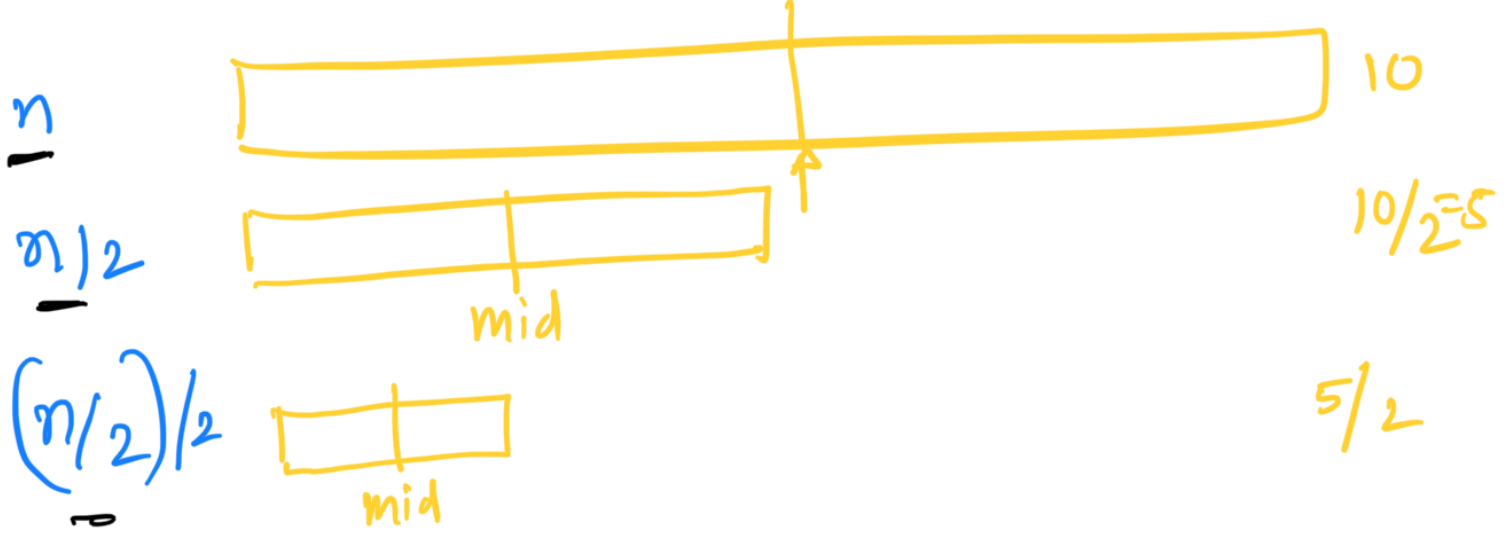
8

$\frac{n}{2^3} = k$

$k = \log_2 n$

② if (mid == key)

→ True



n/8

Algorithm Strategy ⇒ Decrease & Conquere

Prerequisit ⇒ Sorted Array →

Best Case ⇒ Search @ Root node (1st mid value)

1 comp.

min = 1 ⇒ O(1)

Worst Case ⇒ Search @ Leaf node

log n

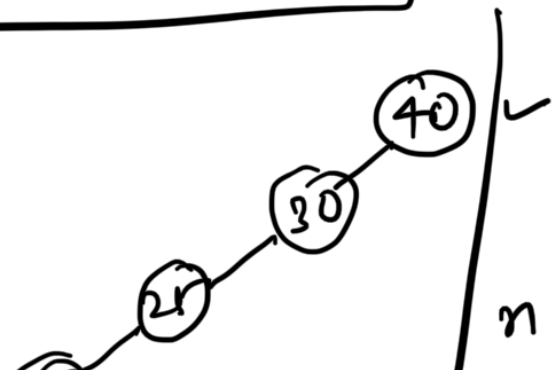
min = log n ⇒ O(log n)

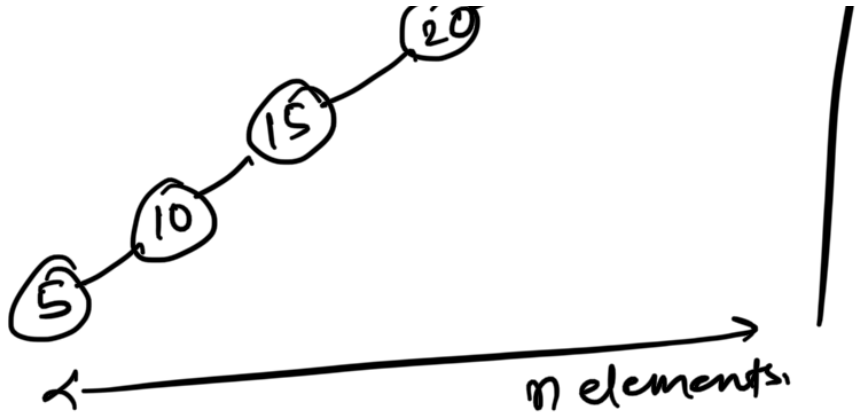
Special Case →

5	10	15	20	25	30	40

Best case ⇒ O(1)

Worst case ⇒ O(n)





Sorting ⇒ arranging a set of data in ascending or in descending.

Algorithm design → Sorting

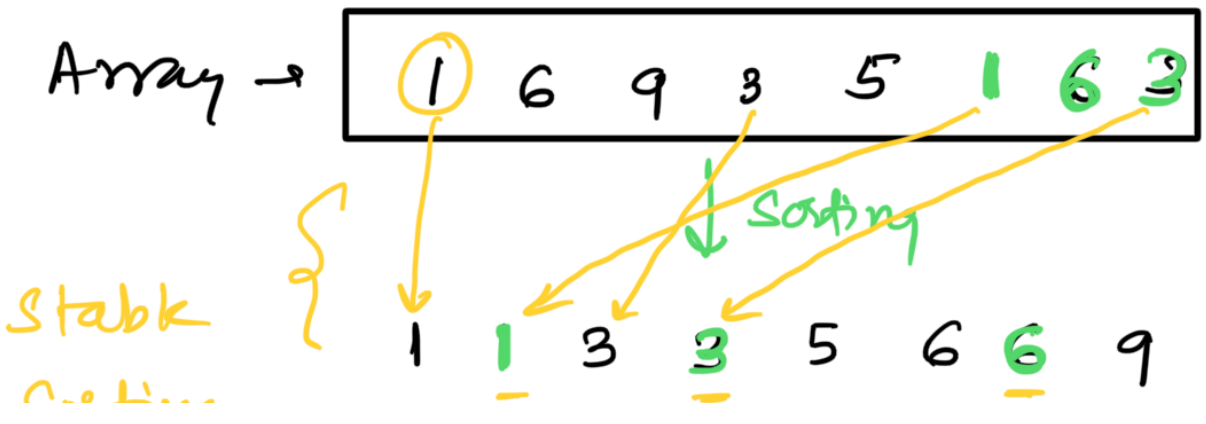
Categories of Sorting –

1. Internal Sorting →
2. External Sorting →

Internal Sorting → data that is to be sorted can be adjusted at a time in main memory

External sorting → data to be sorted can't be accommodated in the memory at the same time and some additional memory is required called auxillary memory to complete the sorting process.

Stable and not stable Sorting



Sorting

not Stable ← { 1 ! 3 3 5 6 6 9

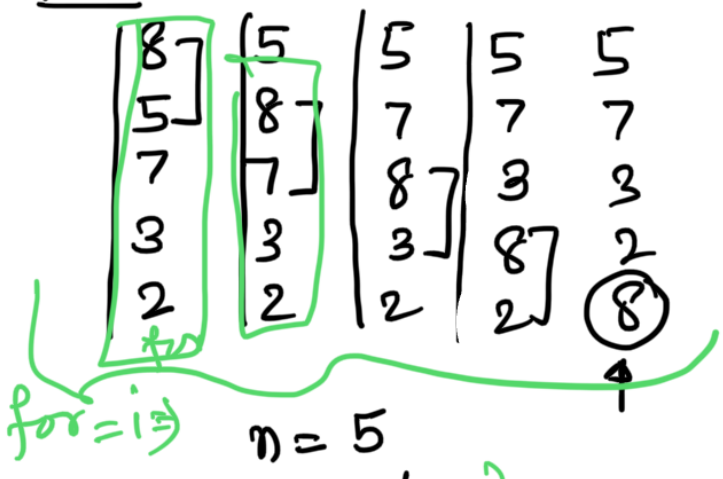
Stable → does not change the sequence of similar elements

Bubble Sort

8	5	7	3	2

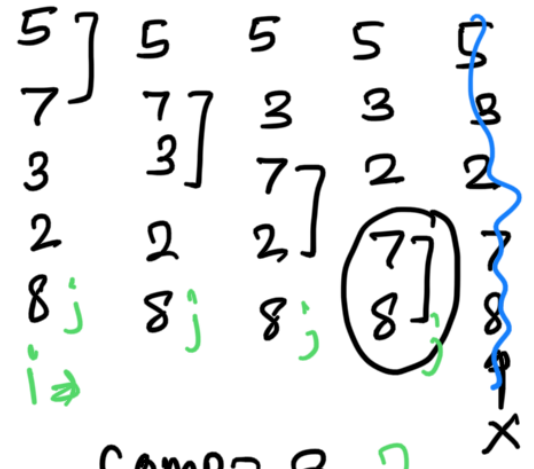
n=5

Pass 1



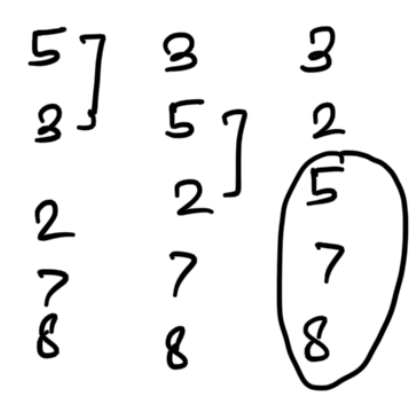
n = 5
Comp = 4
Swap = 4

Pass 2



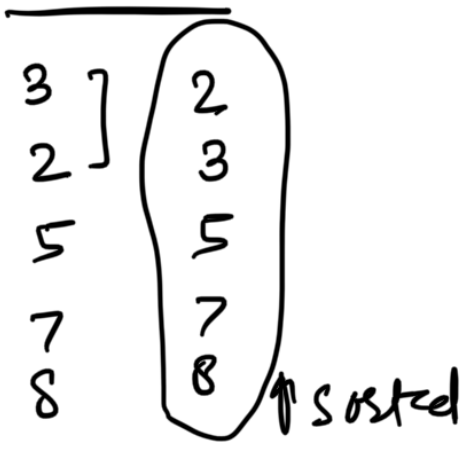
Comp = 3
Swap = 3

Pass 3



Comp = 2
Swap = 2

Pass 4



Comp = 1
Swap = 1

n
(n-1)

① No of passes ⇒ (n-1) / 4 passes

② No of Comp = 1+2+3+4+ ... (n-1) = $\frac{n(n+1)}{2}$ = $O(n^2)$

③ Max no. of swaps = 1+2+3+4 ... (n-1) = n(n+1)

$$\frac{}{2}$$

$$= O(n^2)$$