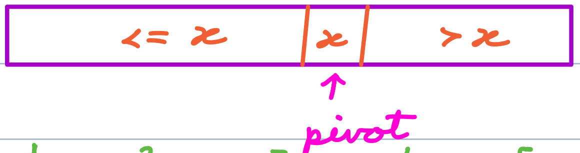
Q → Given an integer array, consider first element as pivot & rearrange the elements s.t elements on left < pivot & elements on right > pivot. SC = O(1)



	0	1	2	3	4	5	6	
A = [	54	26	93	17	77	31	44	]

$\underbrace{26\quad 17\quad 31\quad 44}\quad 54\quad \underbrace{93\quad 77}$

in any order

	0	1	2	3	4	5	6	7	
A = [	10	13	7	8	25	20	23	5	]

7 8 5 10 13 25 20 23

	0 31	1	2	3	4 54	5	6	
A = [	~~54~~	26	~~93~~	17	~~77~~	~~31~~	~~44~~	]
			44		~~31~~	77	93	
	p	~~l~~	~~l~~	~~l~~	~~l~~	~~r~~	~~r~~	
					r	l		

```
p = A[0]
l = 1
r = N-1
while (l <= r) {
    if (A[l] <= p)   l++
    else if (A[r] > p)   r--
    else   swap (A[l], A[r])
}
```

swap (A[l], A[r])   TC = O(N)   SC = O(1)
return r

Quick Sort

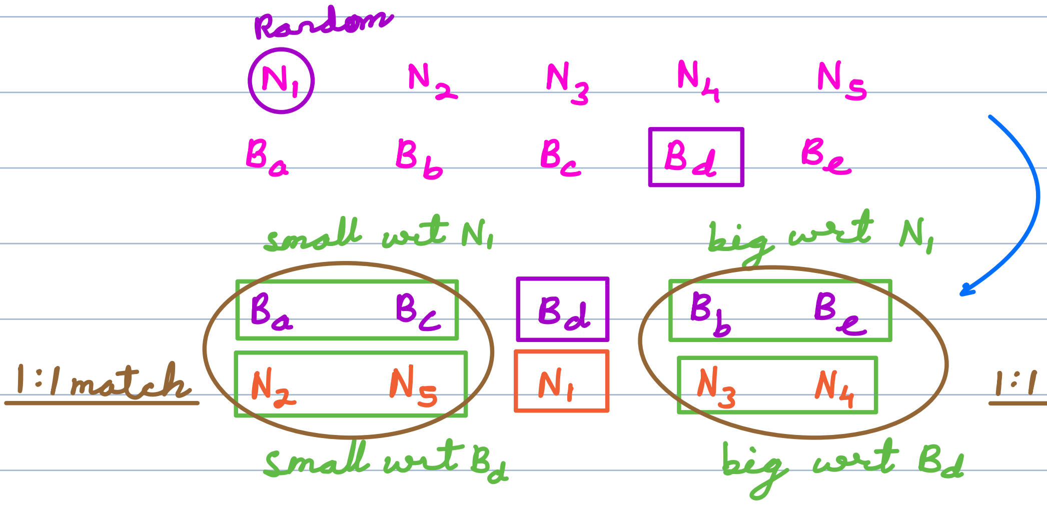
Idea → N nuts & N bolts all of unique size with 1:1 mapping.
Find the nuts & bolts match with constraint that comparing a nut with another nut & bolt with another bolt is not allowed.

N_1 N_2 N_3
B_a B_b B_c

If we compare a nut with a bolt →

1) Exactly fit → match found
2) Nut is small wrt bolt
3) Nut is big wrt bolt

Bruteforce → Compare every nut with every bolt.
TC = O(N^2)



Partitioning
TC = O(N)

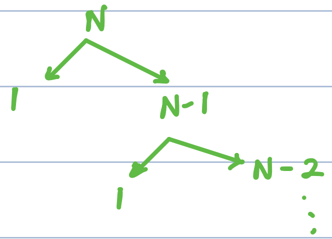
⇒ solve subproblems recursively

Similar to Merge Sort → Divide & Conquer

Advantage of Quick Sort → SC of Partition = O(1)

Disadvantage of Quick Sort →

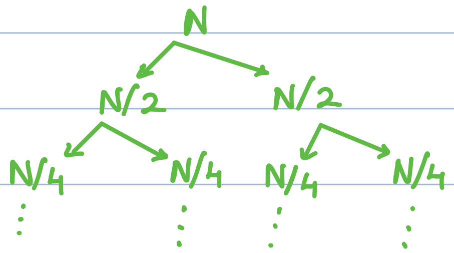
Worst Case TC = $O(N^2)$



```
       0     1     2     3     4     5
A = [ (4)    1     8     9     3     8 ]
     [(1)   3]     4    [(8)   9     8]
      1   [3]              [8]   8   [9]
```

```
                   0        N-1
void sort (A, l, r) {
    if (l >= r) return
    pidx = partition (A, l, r)
    sort (A, l, p_idx - 1)
    sort (A, pidx + 1, r)
}
```

Best Case

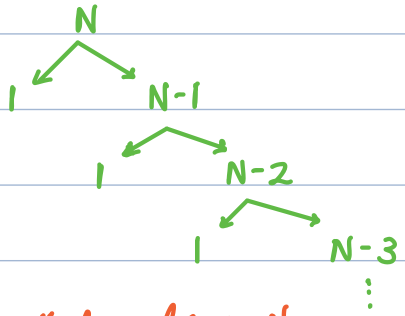


levels = log (N)

TC = O(N log (N))

SC = O(log (N))

Worst Case

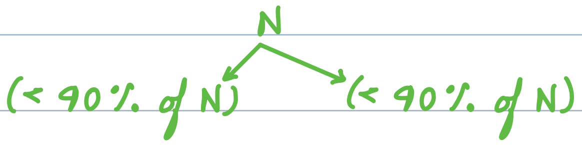


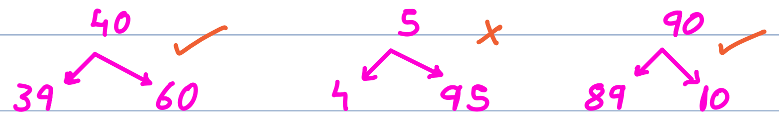
levels = N

TC = $O(N^2)$

SC = O(N)

1 —— 100

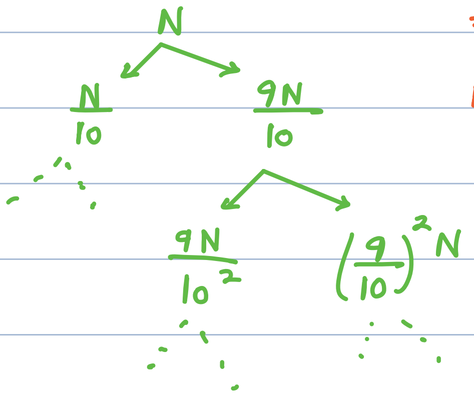




Allowed No. → [11 90] → $90 - 11 + 1 = \underline{80}$

There are 80% chances of this.

Worst Case



#levels →

$N \rightarrow \frac{9N}{10} \rightarrow \left(\frac{9}{10}\right)^2 N \rightarrow . . .$

after K steps

$$\left(\frac{9}{10}\right)^K N = 1$$

$$\Rightarrow N = \left(\frac{10}{9}\right)^K$$

$$\Rightarrow \boxed{\log_{10/9}(N) = K}$$

$$\boxed{N <= 10^5}$$

$\log_{10/9}(N) \approx 109 \rightarrow 10^2$

$TC = O(N \log_{10/9}(N)) \rightarrow 10^5 * 10^2 = \underline{10^7 \text{ iterations}}$ ✓

Comparator

```
int compare ( x , y ) {
        if ( x to be on left of y) → return -ve
      if ( they are equal ) → return 0
      if ( x  to  be  on right of y) → return +ve
}
```

Q → Given an integer array, sort the array wrt count of factors of each element.

A = [9 3 10 6 4]

#factors → 3 2 4 4 3

o/p → 3 9 4 10 6

A = [10 4 5 13 1]

#factors → 4 3 2 2 1

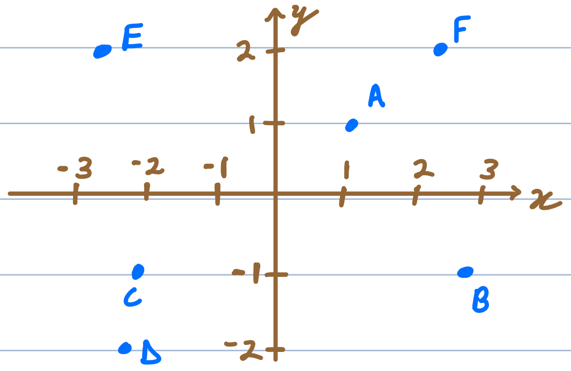
o/p → 1 5 13 4 10

```
int compare ( x , y ) {
    xf = count_factors (x)
    yf = count_factors (y)
    return xf - yf
}
```

Q → Given a list of points in 2D plane & an integer K. Find K closest points to origin (0,0)

Distance → $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$\sqrt{x^2+y^2}$



K = 3

A → $\sqrt{1^2+1^2}=\sqrt{2}$ ✓

B → $\sqrt{3^2+(-1)^2}=\sqrt{10}$

C → $\sqrt{(-2)^2+(-1)^2}=\sqrt{5}$ ✓

D → $\sqrt{(-2)^2+(-2)^2} = \sqrt{8}$ ✓

E → $\sqrt{(-3)^2+ 2^2} = \sqrt{13}$

F → $\sqrt{2^2+ 2^2} = \sqrt{8}$ ✓

Distance → $\sqrt{x^2+y^2}$

⇓

Use to compare → $\underline{x^2+y^2}$

```
class Point {
    int x, y;
}
```

```
int compare( P1, P2) {
        d1 = P1.x * P1.x + P1.y * P1.y
        d2 = P2.x * P2.x + P2.y * P2.y
        return d1 - d2
}
```

TC = $\underline{O(N \log(N))}$

Q → Given a list of +ve numbers, arrange them s.t they form the largest number.

A = [10 2] → 102

210 ✓

A = [3 30 34 5] → $\underline{5}\ \underline{34}\ \underline{3}\ \underline{30}$ (Ans)

A = [10 5 2 8 200] → 8 5 2 200 10 (Ans)

556 ✗ 565 ✓

A = [5 52 56] → 56 5 52 (Ans)

552 ✓ 525 ✗

```
int compare (x, y) {
        xy = x.append(y)        → int ⇒ length 9 string
        yx = y.append(x)
        if (xy > yx) return -1
        else if (yx > xy) return 1
        else 0
    }
```

$TC = \underline{O(N \log(N))}$