Sorting

TABLE OF CONTENTS

Sorting → Arrangement of objects in a particular order wrt specific parameter.

2 8 1 3 ⟶ 1 2 3 8

ascending order

1 13 9 6 12

factors → 1 2 3 4 6

Why →
1) organizing
2) analyzing
3) searching

Question (Elements Removal)

Given N elements, at every step remove an array element.

Cost to remove an element = Sum of array of elements present in an array

Find minimum cost to remove all elements.

NOTE : First add the cost of removal and then remove it.

0 1 2
arr - [2 1 4]

index to remove	cost	array
0	2 + 1 + 4 = 7	[1 4]
1	1 + 4 = 5	[4]
2	4 = 4	x

16 x

---

0 1 2
arr - [2 1 4]

index to remove	cost	array
2	2 + 1 + 4 = 7	[2 1]
0	2 + 1 = 3	[1]
1	1 = 1	x

11 ✓

0 1 2
A = [4 6 1]

index	cost	rem array
1	4 + 6 + 1 = 11	[4 1]
0	4 + 1 = 5	[1]
2	1 = 1	x

17

A = [3 5 1 -3]
0 1 2 3

index	cost	rem array
1	3+5+1-3 = 6	[3 1 -3]
0	3+1-3 = 1	[1 -3]
2	1-3 = -2	[-3]
3	-3	X
	2	

Observation → Remove large values.

[a b c d]
0 1 2 3

wt → 1 2 3 4 → (i+1)

Remove	cost
a	a + b + c + d
b	b + c + d
c	c + d
d	d
	a + 2b + 3c + 4d

minimise cost

a > b > c > d

// sort A[] in descending → TC = O(N log(N)) SC = O(N) / O(1)

```
cost = 0
for i → 0 to (N-1) {
    cost += A[i] * (i+1)
}
return cost
```

TC = O(N log(N) + N) = O(N log(N)) SC = O(N)/O(1)

Question (Noble Integers) { Distinct data }

Given N array elements, calculate number of noble integers.

An element ele in arr [ ] is said to be noble if { count of smaller elements = ele itself }

0 1 2 3 4 5
arr - [1 , -5 , 3 , 5 , -10 , 4]

ele < A[i] → 2 1 3 5 0 4 Ans = 3

0 1 2 3
arr - [-3 , 0 , 2 , 5]

elements < A[i] → 0 1 2 3 Ans = 1

Bruteforce → ∀ A[i], find the # elements < A[i] & check for noble integer.
O(N) O(N)

TC = O(N²) SC = O(1)

```
ans = 0
for i → 0 to (N-1) {
    cnt = 0
    for j → 0 to (N-1) {
        if ( A[j] < A[i] ) cnt++
    }
    if ( cnt == A[i] ) ans++
}
return ans
```

	0	1	2	3	4	5
A = [	3	8	2	-1	5	-3]
#e < A[i] →	3	5	2	1	4	0

Ans = 2

move all elements < A[i] on 1 side

⇒ counting may be faster

Arranging data s.t smaller values are on 1 side ⇒ Sorting

(left) → (ascending order)

	0	1	2	3	4	5	
Sorted → A = [	-3	-1	2	3	5	8]	
#e < A[i] →	0	1	2	3	4	5	= index

```
// sort A[] in ascending order
cnt = 0
for i → 0 to (N-1) {
    if ( A[i] == i ) cnt++
}
return cnt
```

TC = O(N log((N) + N) = O(N log(N))

SC = O(N) / O(1)

Question (Noble Integers) : { Data can repeat }

```
                0   1   2   3   4
arr - [ -10 , 1 , 1 , 3 , 100 ]
#e < A[i]  →    0   1   1   3   4        Ans = 3
```

```
                0   1   2   3   4   5   6   7   8
arr - [ -10 , 1 , 1 , 2 , 4 , 4 , 4 , 8 , 10 ]
#e < A[i] →     0   1   1   3   4   4   4   7   8        Ans = 5
```

```
                0   1   2   3   4   5   6   7   8   9   10  11  12  13
arr - [ -3 , 0 , 2 , 2 , 5 , 5 , 5 , 5 , 8 , 8 , 10 , 10 , 10 , 14 ]
#e < A[i] →     0   1   2   2   4   4   4   4   8   8   10  10  10  13        Ans = 7
```

```
// sort A[] in ascending order
ans = 0        cnt = 0
for i → 0 to (N-1) {
    if (i == 0 || A[i] != A[i-1])
        cnt = i
    if (cnt == A[i])    ans++
}
return ans
```

TC = O(N log (N) + N) = O(N log (N))

SC = O(N) / O(1)

Selection Sort

idea : Select the minimum element and send that elements to correct position by swapping.

	0	1	2	3	4	5
A = [	3	8	2	-1	5	-3]

Find max element in A[] → TC = O(N) SC = O(1)

Find second max element → TC = O(2N) = O(N)

SC = O(2) = O(1)

Find third max element → TC = O(3N) = O(N)

SC = O(3) = O(1)

⋮

Find K^{th} largest element → TC = O(K * N) ✓

SC = O(K) → O(1) ✓

	0	1	2	3	4	5
A = [	~~3~~	~~8~~	2	~~-1~~	~~5~~	~~-3~~]
	-1	-3		3	5	8

K elements is sorted position

∴ if K = N-1 ⇒ sorted array

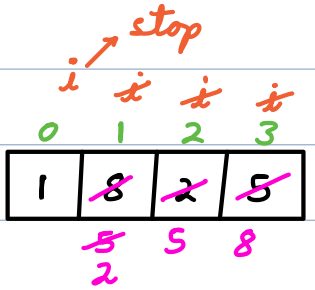
	0	1	2	3	4	5
A = [	~~3~~	~~8~~	~~2~~	~~-1~~	~~5~~	~~-3~~]
	~~-1~~	~~-3~~	2	3	5	8
	-3	-1				

Selection sort

< / > Code

```
for i → (N-1) to 1 {    // 0 — i
    m = 0    // index of max element
    for j → 1 to i {
        if (A[j] > A[m])  m = j
    }
    swap (A, m, i)
}
```

index



m = ~~0~~ 1

$TC = O(N^2)$

$SC = O(1)$

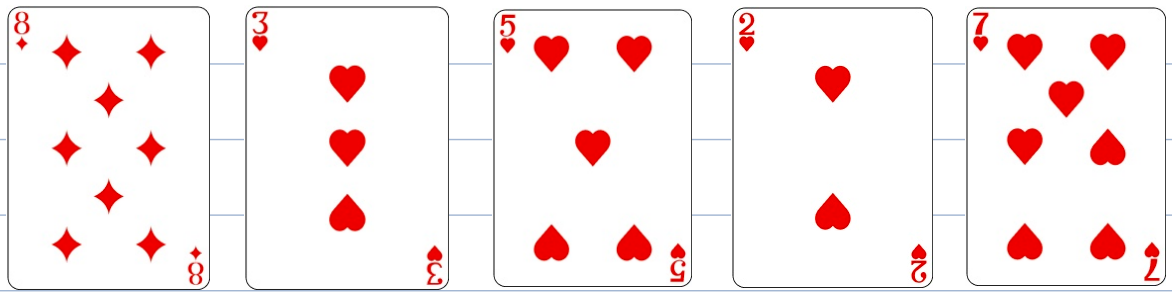
- Max → last
- Max → first
- Min → last
- Min → first

Descending

Ascending

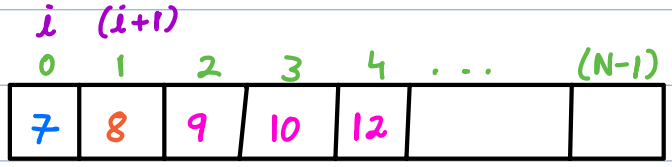
H.W → Study Bubble Sort.

Insertion Sort (Arrangement of playing cards)



Why used → It can sort running stream of data.

i/p → 7 9 12 10 8 (x)



n = 4

for any input → min swaps = 0

max swaps = # of elements in array

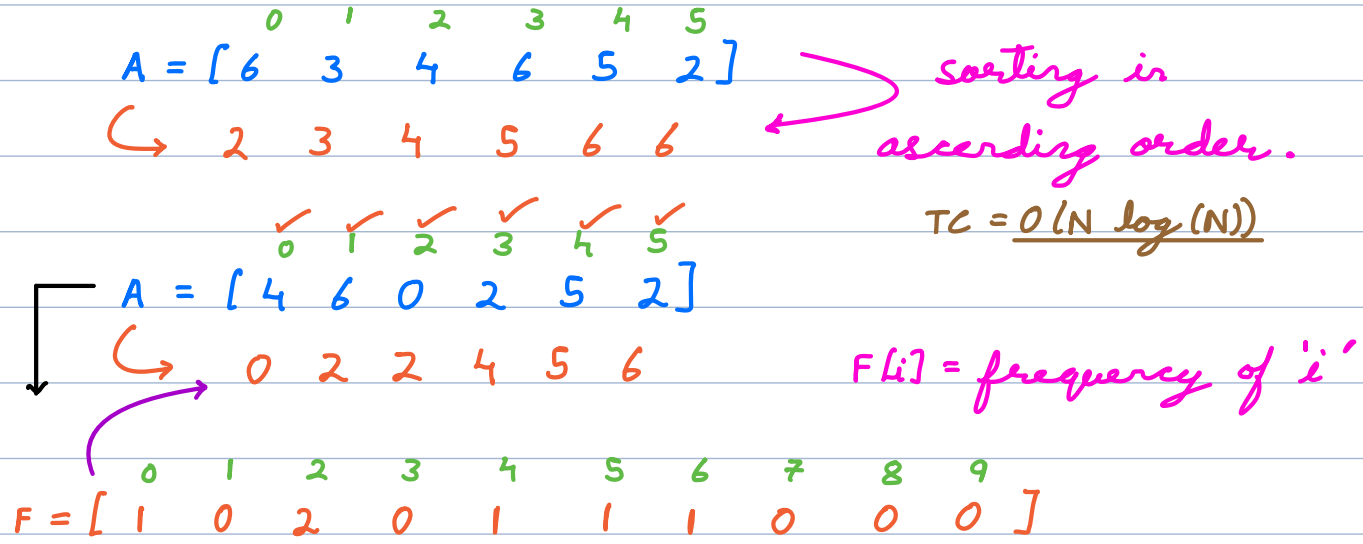
```
n = 0
for Vinputs, x {
    i = n-1                    // 0 ___ i (current array)
    while (i >= 0) {
        if (A[i] > x)
            A[i+1] = A[i]      // shift right
        else
            break              // i → index of small/equal
        i--
```

```
    }  A[i+1] = x
}  return A
```

TC = O(N²) SC = O(1)

Q → Find the smallest number that can be formed by rearranging the digits in the given array.

$0 <= A[i] <= 9$




```
for i → 0 to (N-1) {
|    F[A[i]]++
}

for d → 0 to 9 {
|   for i → 1 to F[d] {
|   |    print(d)
|   }
}
```

Count Sort

Total TC = O(N + N) = O(N)

SC = O(|A[i]|) = O(10) = O(1)

If $A[i] <= 10^9$ ⇒ MLE error (SC is very high)

Works for range ~ 10^6 to 10^7

Sort array where $-5 <= A[i] <= 5$.

	0	1	2	3	4	5	6	7
A =	3	-5	2	3	0	-1	4	-1

x →	-5	-4	-3	-2	-1	0	1	2	3	4	5
	0	1	2	3	4	5	6	7	8	9	10
F →	1	0	0	0	2	1	0	1	2	1	0

index = x - min

o/p → -5 -1 -1 0 2 3 3 4

= x - (-5) = x + 5

```
// find minimum of A[] → m
for i → 0 to (N-1) {
    F[A[i] - m] ++
}

for i → -5 to 5 { // Range
    id = i - m
    for j → 1 to F[id] { // # times an element is present
        print (i)        // for (i = 1; i <= F[id]; i++)
    }
}
```

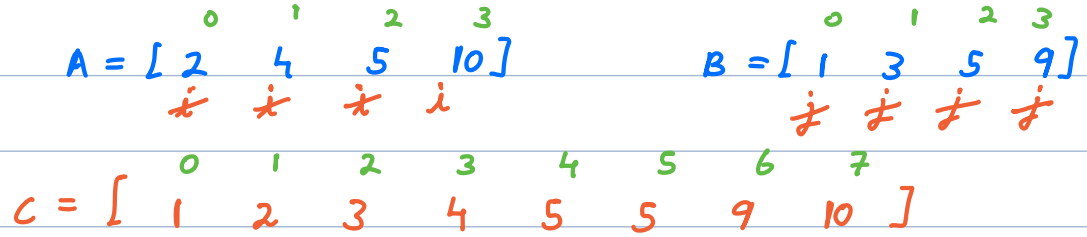
TC = O(N) SC = O(|A[i]|)

Q → Merge two sorted arrays into a single sorted array.

	0	1	2	3
A =	2	4	5	10

	0	1	2	3
B =	1	3	5	9

o/p → 1 2 3 4 5 5 9 10

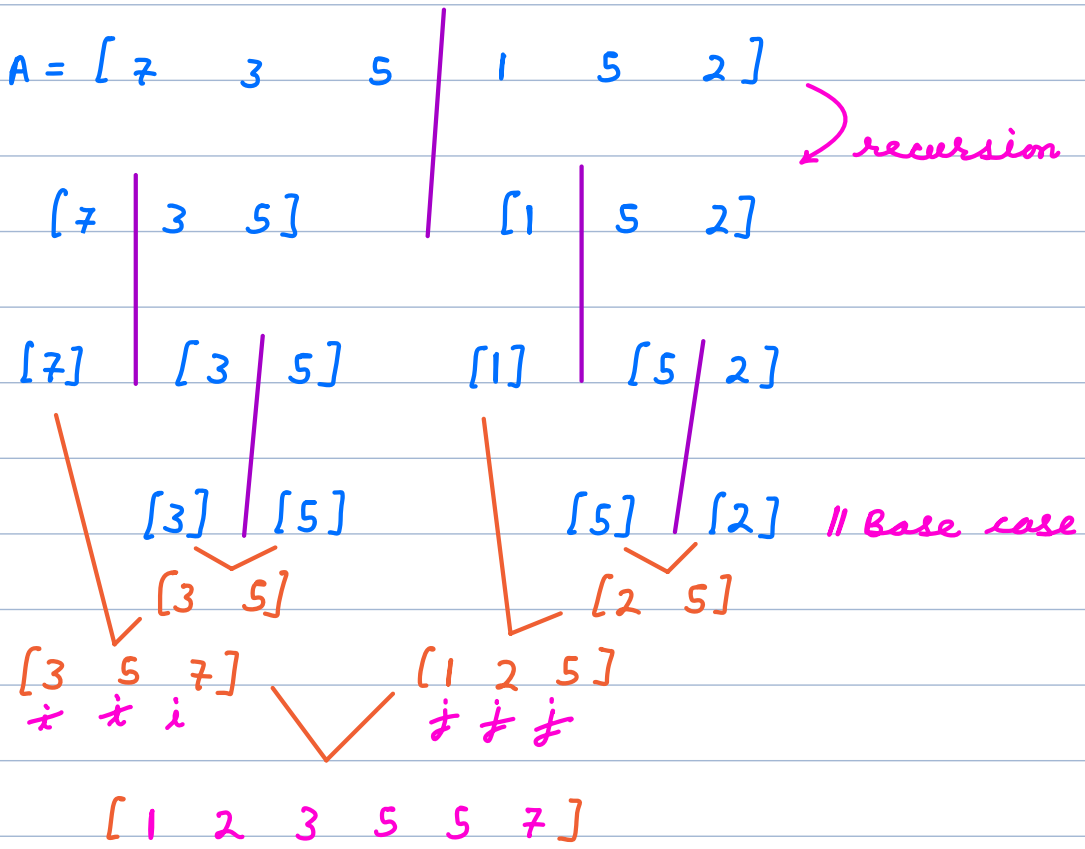



```
int[] merge (A[], B[]) {
    N = A.length        M = B.length
    // C[N+M] → Merged array
    i = 0      j = 0      k = 0
    while (i < N && j < M) {
        if (A[i] <= B[j]) {
            C[k] = A[i]
            i++    k++
        } else {                    // select from right half
            C[k] = B[j]             // cnt += N - i
            j++    k++
        }
    }

    while (i < N) {
        C[k] = A[i]
        i++    k++
    }

    while (j < M) {
        C[k] = B[j]
        j++    k++
    }

    return C
}
```

A[l — mid] A[mid+1 — r]

TC = O(N+M) SC = O(1) A, B → i/p C → o/p

Merge Sort (Divide & Conquer)




```
                                  0    N-1
void sort (A[], l, r) {
    if (l >= r) return
    mid = (l+r)/2
    sort (A, l, mid)
    sort (A, mid+1, r)
    merge (A, l, mid, r)      // l — mid   (mid+1) — r
}
```

TC = O(N) SC = O(N)

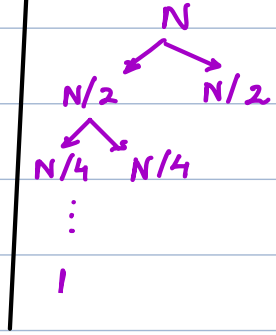
TC = O(N * # levels) ⇒ divide from mid

levels = log (N)

TC = O(N * log (N))

SC = O(N + log (N)) = O(N)

merge recursion



Q → Given two **sorted** arrays A[] & B[].
Find # pairs i,j s.t A[i] > B[j].

```
      0  1  2
A = [ 3  5  7 ]
B = [ 0  2  6 ]
```

Ans = 7

i	j
0	0
0	1
1	0
1	1
2	0
2	1
2	2

Bruteforce → ∀ i,j check
A[i] > B[j]. TC = O(N * M)

```
      ✓  ✓  ✓
      0  1  2  3
A = [ 1  3  5  7 ]
      ↑  ↑  ↑  i

      ✓  ✓  ✓
      0  1  2
B = [ 0  2  6 ]
      ↑  ↑  ↑
```

cnt = 0 + 4 + 3 + 1
= 8

when selecting B[j]
Ans += # remaining elements in A[]
= A.length − i

```
i = 0   j = 0
while ( i < N && j < M) {
    if (A[i] <= B[j])   i++
    else {  j++
            cnt += N-i
        }
} return cnt
```

TC = O(N + M) SC = O(1)

Q → Given an integer array A, find # pairs s.t i < j & A[i] > A[j].

A = [10 3 8 15 6] (0 1 2 3 4)

Ans = 5

i	j
0	1
0	2
0	4
2	4
3	4

A = [5 2 6 1] (0 1 2 3)

Ans = 4

i	j
0	1
0	3
1	3
2	3

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

Ans = 7

Bruteforce → $\forall i, j$ s.t $i<j$ check $A[i] > A[j]$.

TC = $O(N^2)$ SC = $O(1)$

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

[5 3 1] [4 2]

[5 3] [1] [4] [2]

[5] [3] [2 4]

[3 5] [1 3 5]

[1 2 3 4 5]

cnt += # of remaining elements in left half when selecting element from right half.

cnt = 0 + 1 + 2 + 1 + 2 + 1 = 7

TC = $O(N \log(N))$

SC = $O(N)$

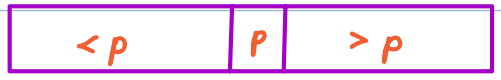
Stable Sorting → Relative order of equal elements should not change while sorting.

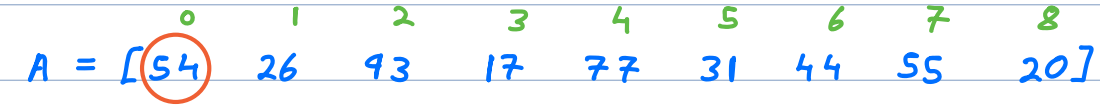
For equal elements → use index to compare.

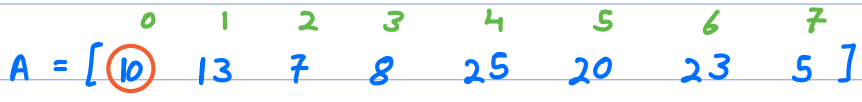
Q → Given an integer array, consider first element as pivot, rearrange the elements s.t

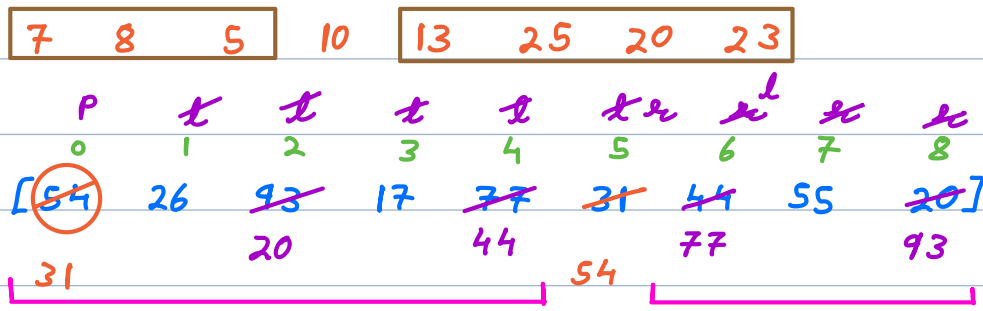
$\forall i$, $A[i] < p \rightarrow$ move left

$A[i] > p \rightarrow$ move right (distinct elements)










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

compare via comparator

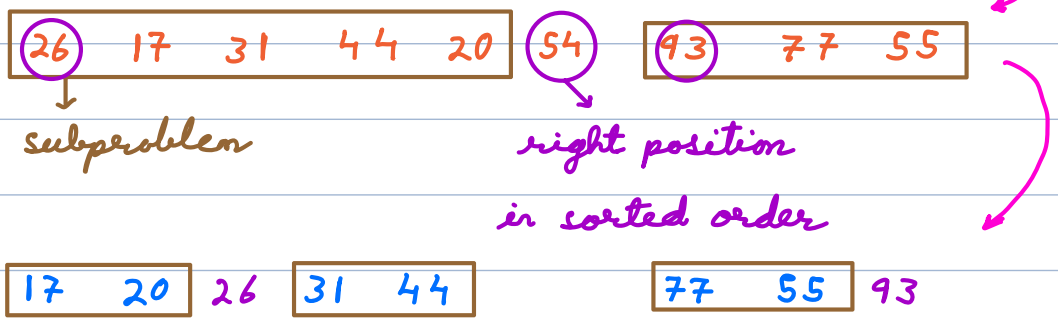
```
    }
}
```

TC = O(N) SC = O(1)

swap (A, r, 0) // pi = r

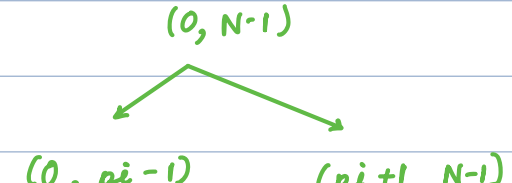
Quick Sort (Divide & Conquer)

	0	1	2	3	4	5	6	7	8
A = [	54	26	93	17	77	31	44	55	20]



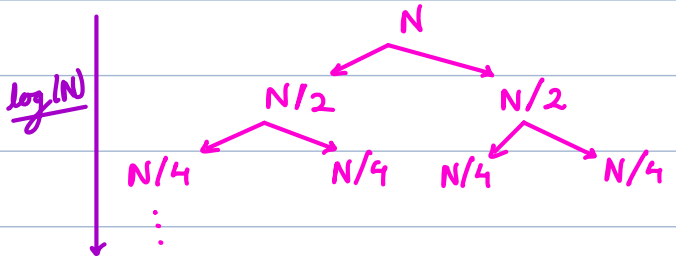

```
void quickSort (A, L, R) {          // L: 0, R: N-1
    if (L >= R) return
    pi = partition (A, L, R)
    quickSort (A, L, pi - 1)
    quickSort (A, pi + 1, R)
}
```

TC = O(N)
SC = O(1)



Time Complexity

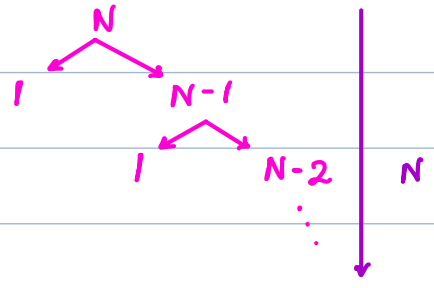
Best Case



TC = O(N log (N))

SC = O(log N)

Worst Case



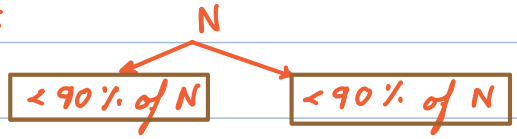
TC = O(N^2)

SC = O(N)

Random Pivot

1 2 3 . . . 9 10 11 . . . . 98 99 100

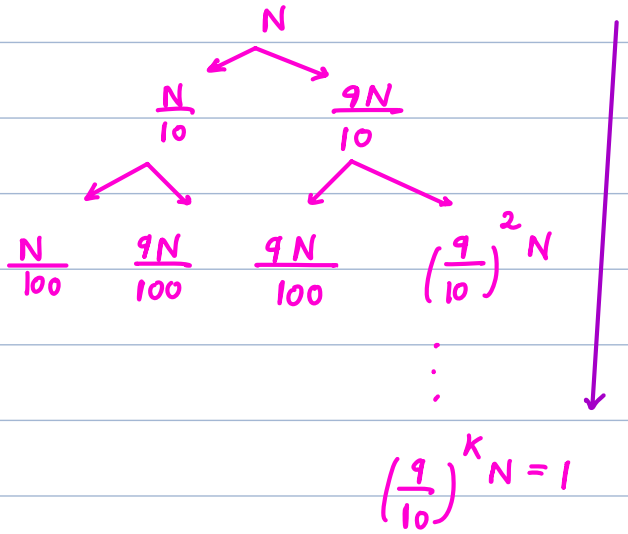
Probability of selecting pivot s.t



valid numbers → [11 90] → 90 - 11 + 1 = 80

⇒ probability = $\frac{80}{100}$ = 0.8 (80 % cases)

Worst scenario with 80 % probability



levels →

$(9/10)^K N = 1 \Rightarrow N = (10/9)^K$

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

$N = 10^5 \Rightarrow \log_{10/9}(N) \approx 10^2$

TC = $O(N \log_{10/9}(N))$ → 10^7

SC = $O(\log_{10/9}(N))$ → 10^2 (better SC wrt Merge sort)

Comparators

It defines the parameter wrt which we organise data.

```
int compare (object u, object v) {
    if 'u' should be on left wrt v → return -ve
    if 'u' should be on right wrt v → return +ve
    if 'u' & 'v' are same → return 0
}
```

Q → Sort the given integer array wrt #factors in ascending order. If count of factors are same use their actual values to compare.

A = [9 3 10 6 4]

#factors → 3 2 4 4 3

o/p → [3 4 9 6 10]

A = [10 4 5 13 1]

#factors → 4 3 2 2 1

o/p → [1 5 13 4 10]

```
int compare (Integer u, Integer v) {
    uf = factors(u)    vf = factors(v)
    if (uf < vf) return -1

    else if (uf > vf) return 1
    else {
```

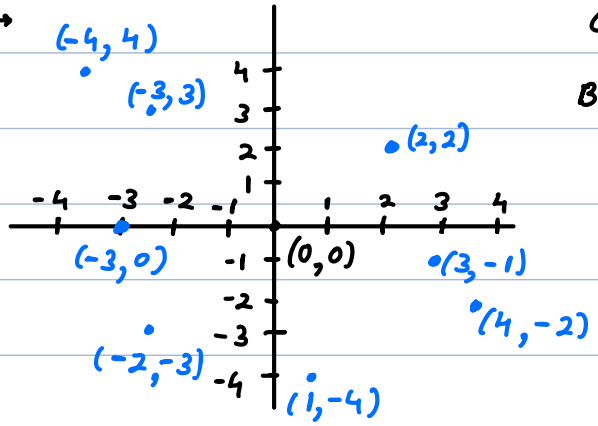
```
        if (u < v) return -1
        else if (u > v) return 1
        else return 0
    }
}
```

return u - v

H.W → check comparator syntax in your programming language.

Q → Given array of points find B closest points to origin.

Distance from origin = $\sqrt{x^2+y^2}$



Eg → B = 3

o/p → (2, 2) (-3, 0) (3, 1)

-4, 4 → $\sqrt{32}$ 2, 2 → $\sqrt{8}$ ✓

-3, 3 → $\sqrt{18}$ 3, 1 → $\sqrt{10}$ ✓

-3, 0 → $\sqrt{9}$ ✓ 4, -2 → $\sqrt{20}$

-2, -3 → $\sqrt{13}$ 1, -4 → $\sqrt{17}$

$x < y$

$\Rightarrow \sqrt{x} < \sqrt{y}$

```
int compare (Point u, Point v) {
    sdu = u.x * u.x + u.y * u.y
    sdv = v.x * v.x + v.y * v.y
    return sdu - sdv
}
```

Q → ... an integer array with +ve numbers. Arrange the array s.t it forms largest number & return it as a string.

A = [10 2 55 100]

↳ "55 2 10 100" (Ans)

(large)
MSD ——— LSD

A = [10 5 2 8 200]

↳ "8 5 2 200 10" (Ans)

53 > 402 → 53 40 2

53 < 532 → 53253 ✗ ⇒ (string comparison)

53532 ✓

53 < 536 → 536 53

```
// int → Strings
int compare (String u, String v) {
    x = u + v   // concatenation
    y = v + u
    if (x < y) return 1
    else if (x > y) return -1
    else return 0
}
```

---

u = 53 (left of 534)

v = 534

x = u+v = 53534 ✓

y = v+u = 53453