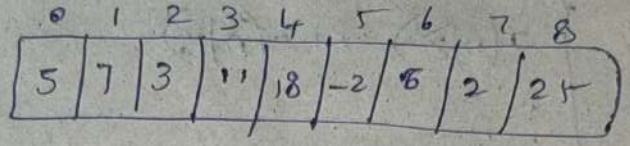
Time And Space Complexity

Linear Search



for n elements

$T(n) = k + T(n-1)$ → for n-1 elements

k → checking operation

$T(n-1) = k + T(n-2)$

$T(n-2) = k + T(n-3)$

$T(1) = k + T(0)$

$T(n) = k + \dots$ n times

$= n \cdot k \Rightarrow TC = O(n)$ $\boxed{cn \geq k \cdot n}$

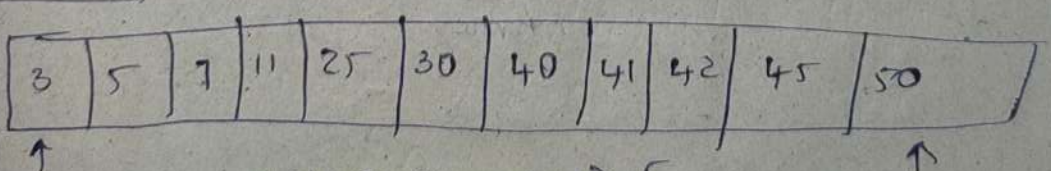
Generally for computing Time complexities,

Best : $\Omega \rightarrow \Omega(1)$

Average $\Theta \rightarrow \Theta(n/2)$

Worst $O \rightarrow O(n)$

Binary Search:




```
while (left < right) {
    // find mid
if (a[mid] < data)
    else if (a[mid] > data)
else
```

comparing operations

$T(n) = K + T(n/2)$ → performing search for half of array

$T(n/2) = K + T(n/4)$

$T(n/2^2) = K + T(n/8)$

$T(n/2^3) = K + T(n/16)$

m terms

$T(n/2^K) = K + T(0)$

$\frac{n}{2^{m-1}} = 1$

$n = 2^{m-1}$

$\log_2 n = \log_2 2^{m-1}$

$m - 1 = \log_2 n$

$m = 1 + \log_2 n$

$T(n) = K + \cdots + K$ [m-terms]

$= mK$

$T(n) = (1 + \log_2 n)K$

~~$T(n) = \log n$~~

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

For Factorial

$T(n) = K' + T(n-1) + K''$ → condition → for n-1 elements → multiplying with (n-1)

$K' + K'' = K$

$T(n) = K + T(n-1)$

$T(n-1) = K + T(n-2)$

$T(n-2) = K + T(n-3)$

$T(1) = K + T(0)$

$T(n) = \underbrace{K + K + \cdots + K}_{n \text{ times}} = nK$

$T(n) = O(n)$

Power Linear

```
public static int power(int x, int n) {
    if (n == 0)
        return 1;
    int xpm = power(x, n-1);
    int xpn = x * xpm;
    return xpn;
}
```

$T(n) = K + T(n-1) + K$

$\boxed{T(n) = O(n)}$

Power logarithmic

```
public static int power2(int x, int y) {
    if (n == 0)
        return 1;
    int xpn = power(x, n/2) * power(x, n/2)
    if (xpn % 2 == 0)
        xpn *= 2;
    return xpn;
}
```

$n \simeq 2^x$

$2^0\ T(n) = K + T(n/2) + T(n/2) = K + 2T(n/2)$

$2^1\ T(n/2) = 2K + 2T(n/4)$ ⟶ [multiply with 2 on both sides]

$2^2\ T(n/4) = 4K + 8T(n/8)$ ⟶ multiply with 2^2 on both sides

$2^3\ T(n/8) = 8K + 16T(n/16)$ ⟶ 2^3

x terms

$T(1) = K + 2T(0)$

$T(0) = \underbrace{K + 2K + 4K + 8K + \cdots}_{x \text{ terms}} = K[1 + 2^0 + 2^1 + \cdots + 2^{x-1}]$

$= K\frac{(1 * 2^{x-1})}{2-1} = K \cdot 2^x - 1 \simeq K \cdot 2^x \quad [n \simeq 2^x]$

$= K \cdot n$

$\boxed{T(n) = O(n)}$

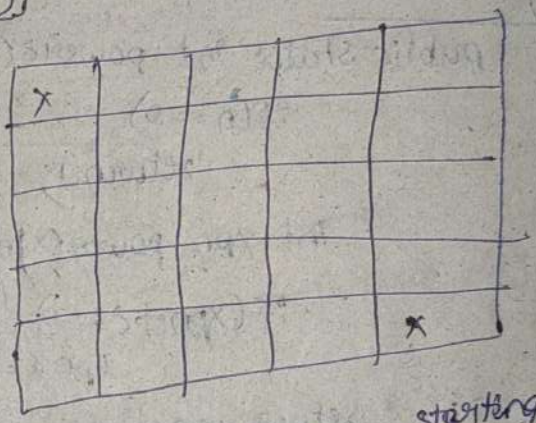
→ Other Approach

```
public static int power3(int x, int n) {
        if(n==0)
            return 1;
        int xpn2= power(x,n/2)
        int xpn= xpn2*xpn2;
        if(n%2!=0)
            xpn*=x;
        return xpn;
}
```

$T(n) = k + T(n/2)$

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

TC for Maize Paths



starting position

n v steps, m h steps $(0,0) \Rightarrow n+m$

n v steps; (m-1) h steps $(0,1) \Rightarrow n+m-1$

n-1 v steps; m h steps $(1,0) \Rightarrow n+m-1$

$T(n+m) = T(n+m-1) + T(n+m-1) + k$

Let $n+m = x$

$T(x) = 2T(x-1) + k$

$2T(x-1) = 4T(x-2) + 2k \quad [\times 2]$

$4T(x-2) = 8T(x-3) + 2^2 k \quad [\times 2^2]$

$8T(x-3) = 16T(x-4) + 2^3 k \quad [\times 2^3]$

$\therefore T(1) = 2^x T(0) + k$

$T(n) = k + 2k + 2^2 k + 2^3 k + \ldots + f$ (x terms)

$T(n) = k * (1 + 2 + 4 + 8 + \cdots + 2^{k-1})$

$= k \cdot \left(\frac{2^k - 1}{2 - 1}\right)$

$= k \cdot 2^k - 1$

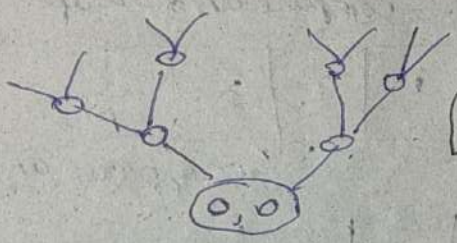
$\approx k \cdot 2^k$

$\approx k \cdot 2^{l+m}$

$T(n) \in O(2^{l+m})$

Recursion Tree – Ram Ban Tareek Technique

T.C = [Avg no of choices from a node] ^ height of the tree



$TC = (2)^{n+m}$

1. Bubble Sort:

Sample Input

```
5
7
-2
4
1
3
```

Sample Output

```
Comparing -2 and 7
Swapping -2 and 7
Comparing 4 and 7
Swapping 4 and 7
Comparing 1 and 7
Swapping 1 and 7
Comparing 3 and 7
Swapping 3 and 7
Comparing 4 and -2
Comparing 1 and 4
Swapping 1 and 4
Comparing 3 and 4
Swapping 3 and 4
Comparing 1 and -2
Comparing 3 and 1
Comparing 1 and -2
-2
1
3
4
7
```

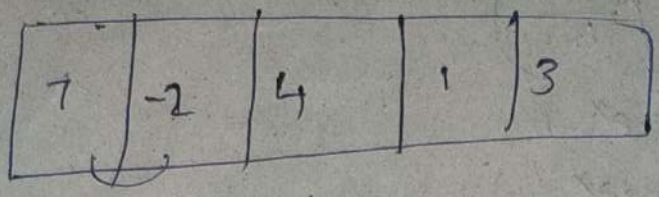
Constraints

$1 <= N <= 10000$

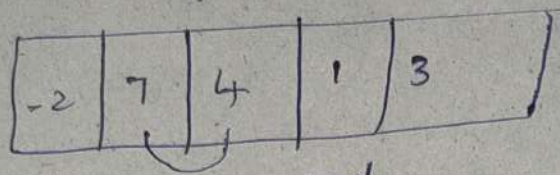
$-10^9 <= arr[i] <= 10^9$

Dry Run; for Bubble Sort

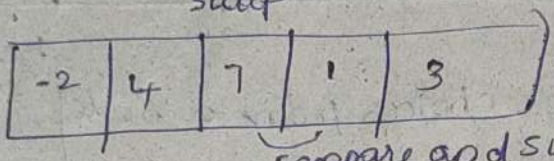
1st - ith



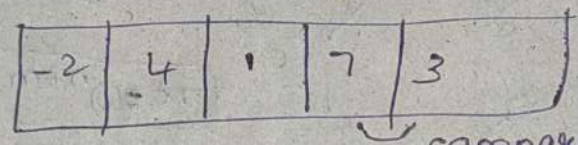
compare and swap



compare and swap

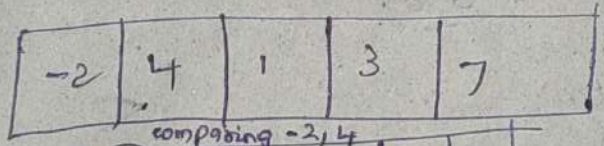


compare and swap

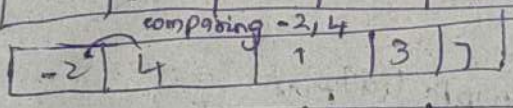


compare and swap

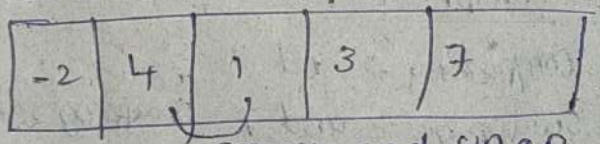
End of 1st ith



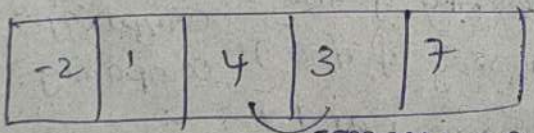
comparing -2, 4



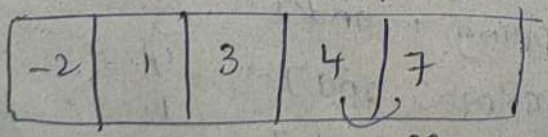
2nd - ith



compare and swap

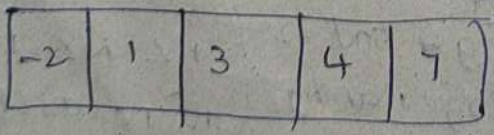


compare and swap

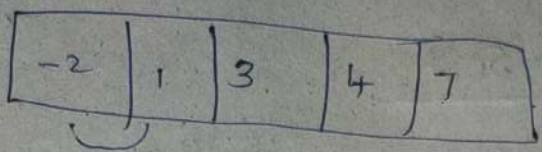


compare

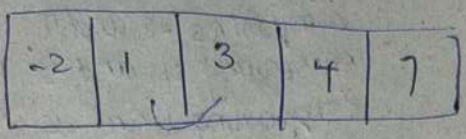
End of 2nd - ith



3rd ith

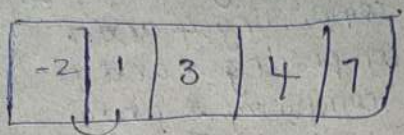


comparing 1, -2



comparing

4th ith



comparing

Steps:

1. Take Outer Loop which traverses through the array.
2. Inside take another loop which traverses through the array with decrementing the outer loop [because, in each outerloop, we will get the n^{th} largest elements, so we will skip those).
3. Check if the elements are smaller, then Apply swapping.

Program:

```
public static void bubbleSort(int[] arr) {
    for(int i=1; i<=arr.length-1; i++) {
        for(int j=0; j<arr.length-i; j++) {
            if(isSmaller(arr, j+1, j)){
                swap(arr, j+1, j);
            }
        }
    }
}
```

2. Selection Sort

Sample Input

5 - no of elements
7
-2
4
1
3

Sample output

comparing -2 and 7
comparing 4 and -2
comparing 1 and -2
comparing 3 and -2
swapping 7 and -2
comparing 4 and 7
comparing 1 and 4
comparing 3 and 1
swapping 7 and 1
comparing 7 and 4
comparing 3 and 4
swapping 4 and 3
comparing 4 and 7
swapping 7 and 4
-2
1
3
4
7

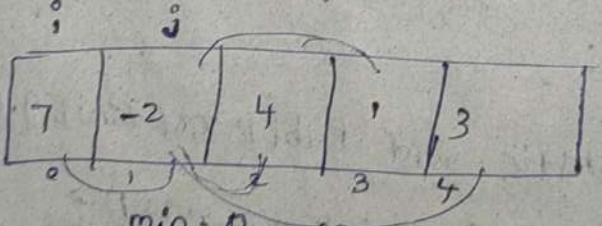
Constraints:

$1 <= N <= 100000$
$-10^9 <= arr[i] <= 10^9$

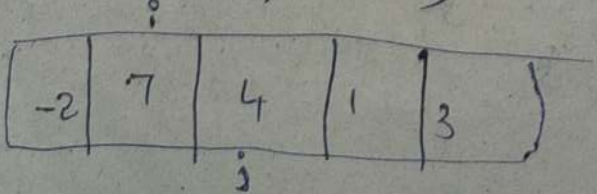
Algorithm:- Steps

1. Traverse through the array, assume minindex = i
2. Go through the rest of elements and check any element is minimum then minindex element, If yes reassign minindex to smallest element. and swap the elements
3. This process continues till all elements are covered and finally swapping is done.

Dry Run:



min = 0, comparing -2 and 7,
update min = 1, comparing 4 and -2
comparing 1 and -2
comparing 3 and -2
swap arr(i), arr[min)



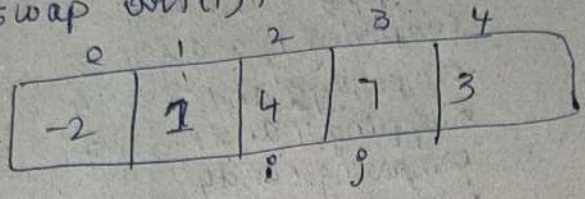
min = 1,

update min = 2 ∈ comparing 4 and 7

update min = 3 ∈ comparing 1 and 4

∈ comparing 3 and 1

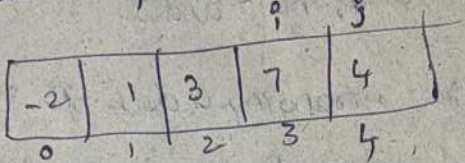
swap arr(i), arr(min)



min = 2

update min = 4 ∈ comparing 7 and 4

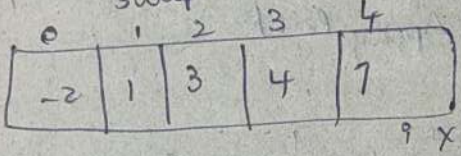
∈ comparing 3 and 4, swap arr(i), arr(min)



min = 3

update min = 4 ∈ comparing 4 and 7

swap arr(i), arr(min)



⟹ sorted

Program:

```
public static void selectionSort(int[] arr) {

    for(int i=0; i<arr.length-1; i++){
        int min=i;
        for(int j=i+1; j<arr.length; j++){
            if(isSmaller(arr, j, min)) {
                min=j;
            }
        }
        swap(arr, i, min);
    }
}
```

3. Insertion Sort

Sample Input

5 → no of elements
7
-2
4
1
3

Sample Output

comparing -2 and 7
swapping 7 and -2
comparing 4 and 7.
swapping 7 and 4
comparing 4 and -2
comparing 1 and 7
swapping 7 and 1
comparing 1 and 4
swapping 4 and 1
comparing 1 and -2
comparing 3 and 7
swapping 7 and 3
comparing 3 and 4
swapping 4 and 3
comparing 3 and 1
-2
1
3
4
7

Steps:

→ Insertion Sort is like Arranging a deck of cards.

1. Go through the array from first element
2. Take another loop and check backwards from index i, if element is greater than j, swap i with j else Apply Break

Why Break?

Eg: | 2 | 3 | 6 | 8 | 1 |

Already sorted

Now checking element 8 as it is greater than 6, No need to check 8, 2 → so Break

Dry Run:

| 7 | -2 | 4 | 1 | 3 |

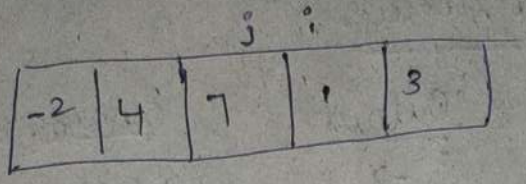
j i

comparing and swapping

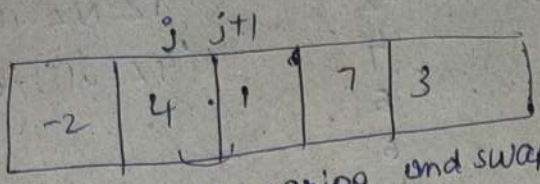
j i

| -2 | 7 | 4 | 1 | 3 |

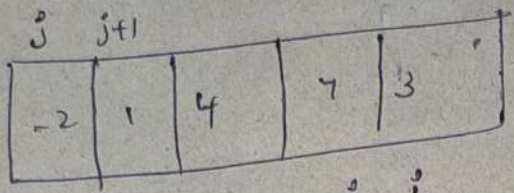
→ comparing and swapping



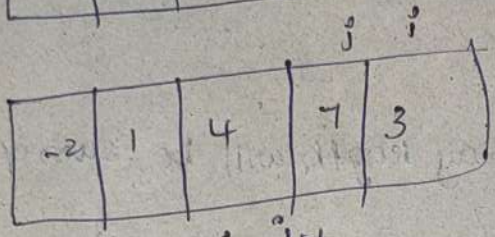
comparing and swapping



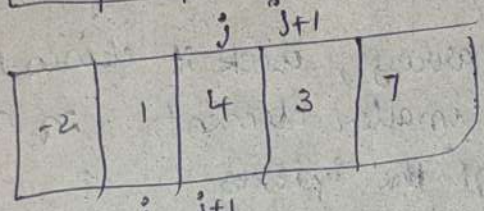
comparing and swapping



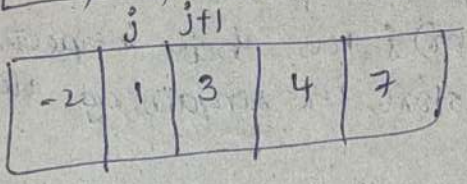
comparing



comparing 7 and 3 swapping



comparing and swapping



comparing and break
i==5 X

Program:

```
public static void insertionSort(int[] arr) {

    for(int i=1; i<arr.length; i++) {
        for(int j=i-1; j>=0; j--) {
            if(isGreater(arr, j, j+1)) {
                swap(arr, j, j+1);
            }
            else {
                break;
            }
        }
    }
}
```

4. Merge Two Sorted Arrays

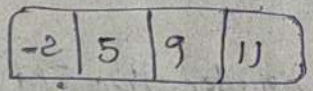
Input	Output	Constraints
4 — no of elements in 1st array	-2	$1 <= N <= 10^8$
-2	4	$-10^9 <= a[i], b[i] <= 10^9$
5	5	
9	6	
11	8	
3 — no of elements in 2nd Array	9	
4	11	
6		
8		

Steps;

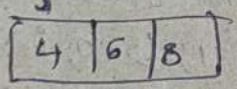
1. The resultant Array length will be sum of both arrays.
2. Iterate through both arrays, check if element is greater than store smaller element in new array and increment the indexes.
3. If still either i (or) j less their respective arrays length store the remaining elements in new array.

Dry Run;

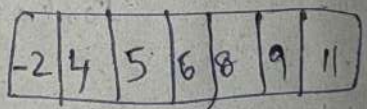
Ar1;



Ar2;



Res;



i<Ar1.length, j<Ar.length ✓ , k=0

-2>=4 X, Res(k)=-2, k++, i=1

1<4, 0<3 ✓

5>=4 ✓

Res(k)=4, j++, j=1

1<4, 1<3 ✓

5>=6 X, Res(k)=5, i++, i=2

2<4, 1<3 ✓

9>=6 ✓ Res(k)=6, j++, j=2

2<4, 2<3 ✓

9>=8 ✓ Res(k)=8, j++, j=3

2<4, 3<3 ✗

// Filling Remaining Elements from 1st Element Array.

2<4 ✓

Res[k]=9, i++, k++

3<4 ✓

Res[k]=11, i++, k++

4<4 ✗

// from 2nd Array

3<3 ✗ ⇒ Res

-2	4	5	6	8	9	11

Program:

```
public static int[] mergeTwoSortedArrays(int[] a, int[] b){
    int[] ans = new int[a.length + b.length];

    int i=0;
    int j=0;
    int k=0;
    while(i<a.length && j<b.length){
        if(a[i] >= b[j]){
            ans[k] = b[j];
            j++;
            k++;
        }
        else{
            ans[k] = a[i];
            i++;
            k++;
        }
    }
```

```
// Remaining Elements from 1st Array
while(i < a.length){
    ans[k] = a[i];
    i++;
    k++;
}
// Remaining Elements from 2nd Array
while(j < b.length){
    ans[k] = b[j];
    j++;
    k++;
}
return ans;
}
```

5. Merge Sort

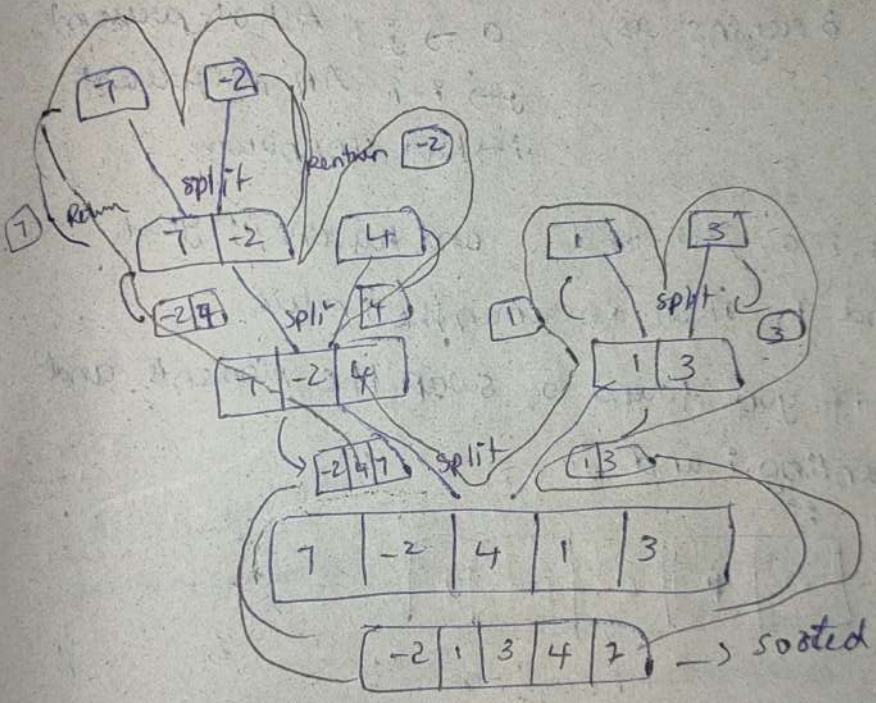
Sample Input	Output	Constraints
5	Merging these two arrays	$1 <= N <= 100000$
7	left array -> 7	$-10^9 <= arr[i] <= 10^9$
-2	right array -> 2	
4	Merging these two arrays	
1	left array -> 2 7	
3	right array -> 4	
	Merging these two arrays	
	left array -> 1	
	right array -> 3	
	Merging these two arrays	
	left array -> 2 4 7	
	right array -> 1 3	
	Sorted Array -> -2 1 3 4 7	

Time Complexity: $n \log n$

steps:

1. Split the Array into halves by mid
2. Sort the two Arrays by using merging two sorted Arrays [previous problem].
3. If low == high, then there will be single element return the single element with array

Dry Run:



Program:

```
public static int[] mergeSort(int[] arr, int low, int hi)
{
    if(lo == hi){
        int[] bs = new int[1];
        bs[0] = arr[low];
        return bs;
    }
    int mid = (lo+hi)/2;
    int[] a = mergeSort(arr, lo, mid);
    int[] b  mergeSort(arr, mid+1, hi);
    int[] ans = mergeTwoSortedArrays(a,b);
    return ans;
}
```

6. Sort 01

Sample Input

```
5
0
1
0
.
0
```

Sample Output

```
Swapping index 0 and index 0
Swapping index 2 and index 1
Swapping index 4 and index 2
0
0
0
.
.
```

Constraints

$1 <= N <= 10000$

$arr[i] = 0, 1$

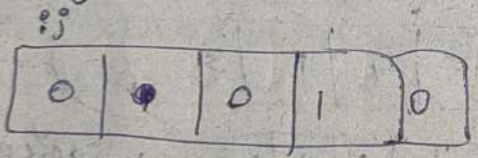
Steps;

1) Consider 3 regions as; $0 \to j-1$ All 0's present
$j \to i-1$ All 1s present
$i \to$ last unknown

2) Initially $i=0, j=0$ and and traverse, if you found 1, then increment the i value

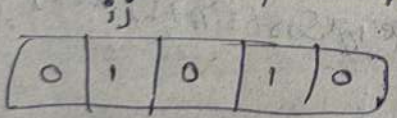
3) Else if you found 0, swap the elements and incrementing i and j.

Dry Run;

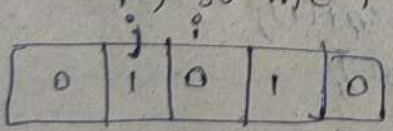


$i=0, j=0$

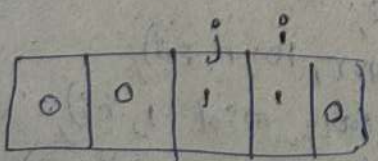
$arr(i)=0$, swap $\Rightarrow$ inc i, j

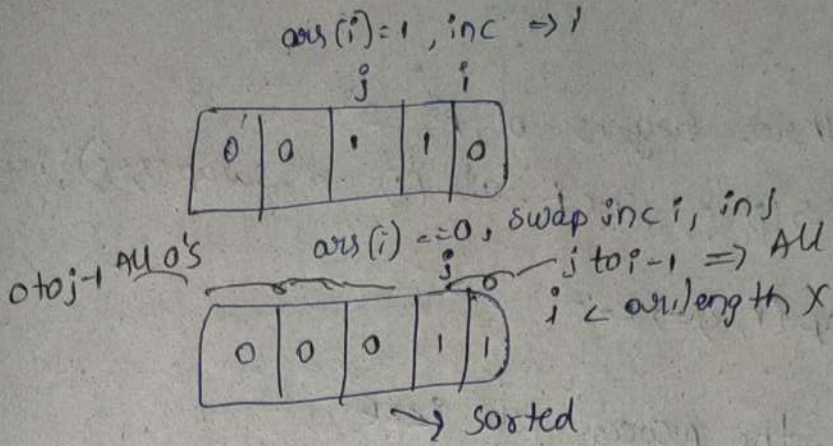


$1 \Rightarrow$ so inc i



$arr(i)=0$, swap $\Rightarrow$ inc i, j





Program;

```
public static void sort01(int[] arr) {
    int i=0;
    int j=0;
    while(i<arr.length) {
        if(arr[i]==1)
            i++;
        else {
            swap(arr,i,j);
            i++;
            j++;
        }
    }
}
```

7. Sort 012

Sample Input

```
10 → No of elements
1
0
2
2
1
0
2
1
0
2
```

Sample Output

```
Swapping index 1 and index 0
Swapping index 2 and index 9
Swapping index 2 and index 8
Swapping index 2 and index 1
Swapping index 3 and index 7
Swapping index 5 and index 2
Swapping index 6 and index 6
0
0
0
1
1
1
2
2
2
2
```

Constraints

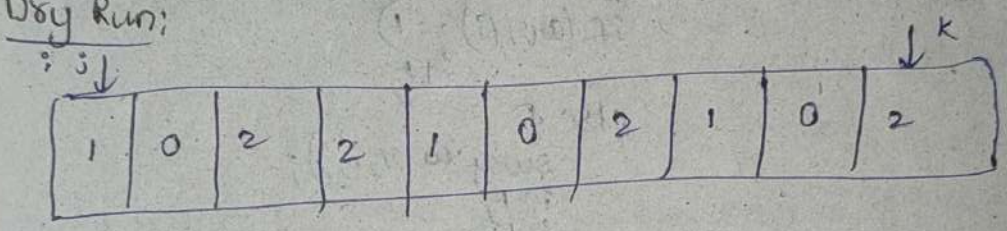
$1 \leq N \leq 10000$

$arr[i] = 0, 1, 2$

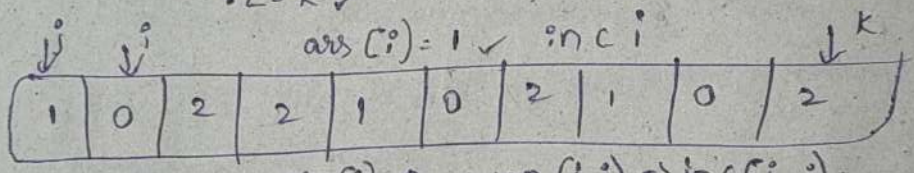
Steps:

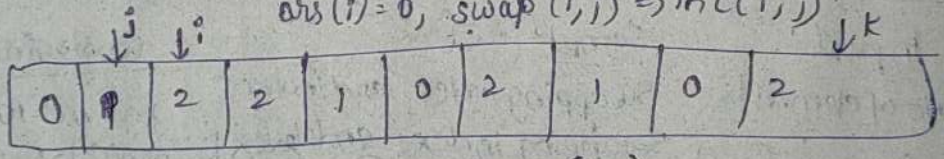
1. Make three regions 0's, 1's, 2's.
2. If arr(i) == 1, inc i, arr(i) == 0, swap(i, j), i++, j++
3. Else swap(i, k) => k--
4. Continue the process till i <= k

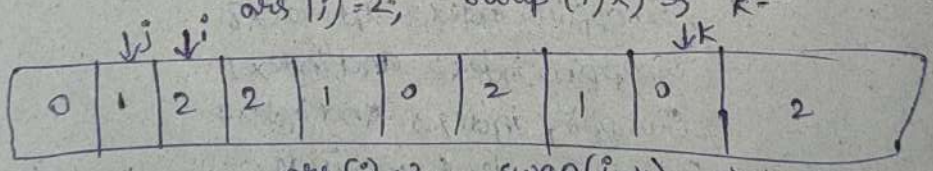
Dry Run:

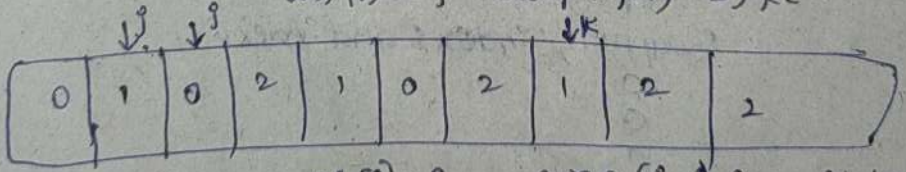


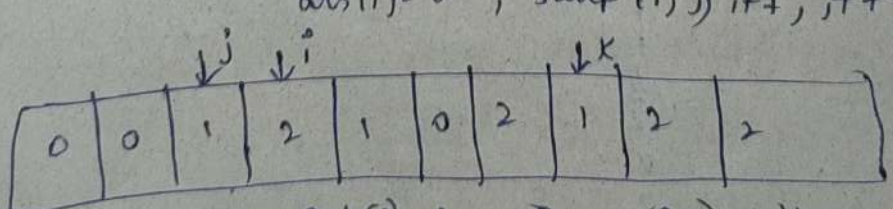
i <= k ✓



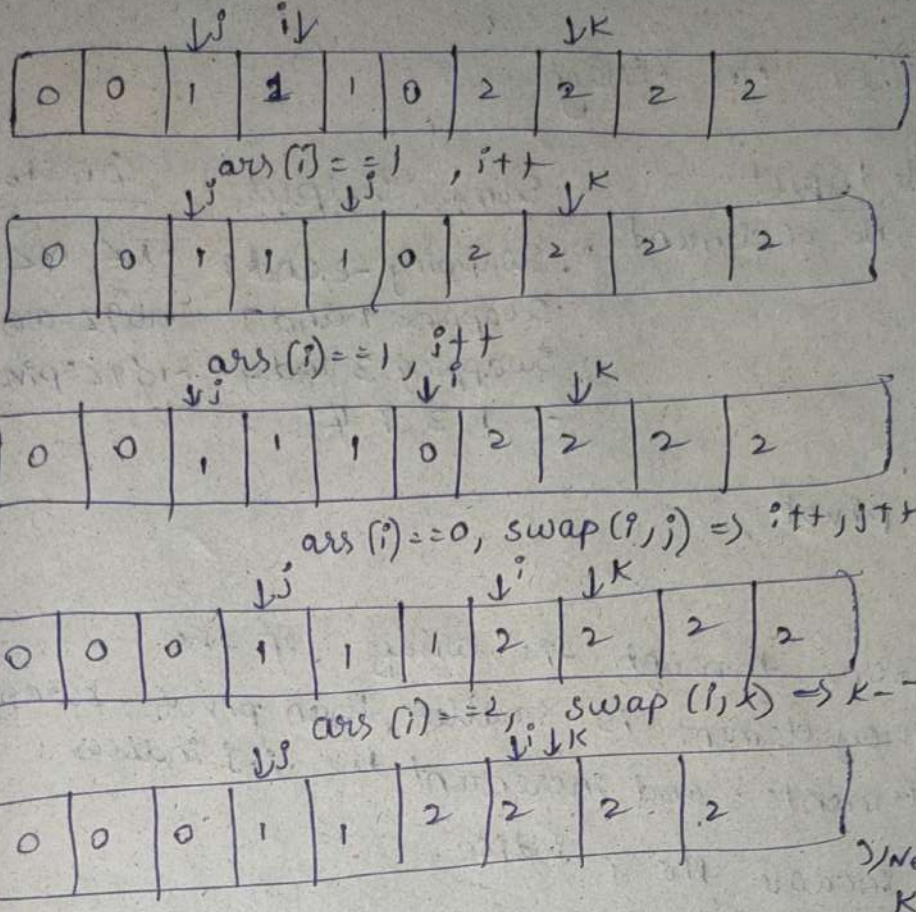








arr(i) = 2, swap(i, k) => k--



Program:

```
public static void sortOIC(int[] arr) {
    int i=0;
    int j=0;
    int k=arr.length-1;
    while(i<=k) {
        if(arr[i]==0) {
            swap(arr,i,j);
            i++;
            j++;
        }
        else if(arr[i]==1) {
            i++;
        }
        else{
            swap(arr,i,k);
            k--;
        }
    }
}
```

8. Partition An Array

Sample Input

5 → No of elements
7
-2
4
1
3
3 → pivot

Sample Output

Swapping -2 and 7
Swapping 1 and 7
Swapping 3 and 4
-2 1 3 7 4

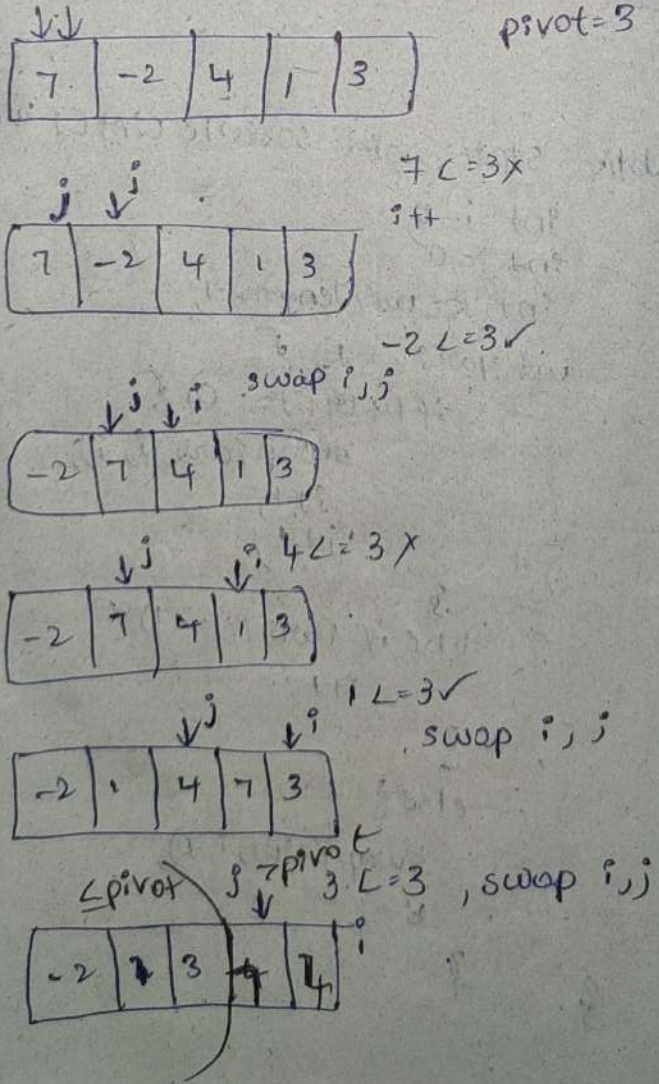
Constraints

$1 <= N <= 1000000$
$-10^9 <= arr[i] <= 10^9$
$-10^9 <= pivot <= 10^9$

Steps:

1. Traverse through the array, if the Array element is smaller than pivot, swap the elements and increment the i, j indices
2. Else increase the j indice

Dry Run:



Program:

```
public static void partition(int[] arr, int pivot){
    int i=0;
    int j=0;
    while(i<arr.length){
        if(arr[i]<=pivot){
            swap(arr,i,j);
            i++;
            j++;
        }
        else{
            i++;
        }
    }
}
```

9. Quick Sort

Sample Input

```
5
7
-2
4
1
3
```

Sample Output

```
pivot -> 3
swapping -2 and 7
swapping 1 and 7
swapping 3 and 4
pivot index -> 2
pivot -> 1
swapping -2 and -2
swapping 1 and 1
pivot index -> 1
pivot -> -2
swapping -2 and -2
pivot index -> 0
pivot -> 4
swapping 4 and 7
pivot index -> 3
pivot -> 7
swapping 7 and 7
pivot index -> 4

-2 1 3 4 7
```

Constraints

$1 <= N <= 100000$

$-10^9 <= arr[i] <= 10^9$

Steps:

1. Get the pivot element, by assigning to arr[hig]
2. Then Partition the Array based on pivot index which Returns the pivot index
3. All elements lesser than pivot index will goto Recursive calls of quick sort and lly the other half.
4. The base condition will be as lo>hi till then the process continues

Time Complexity:

$$T(n) = 2T(n/2) + n + K$$
$$2T(n/2) = 2T(n/4) + 2n/2 + 2K \quad \times 2$$
$$2^2T(n/4) = 2^3T(n/8) + 4\frac{n}{4} + 4K \quad \times 4$$
$$2^3T(n/8) = 2^4T(n/16) + 16\times\frac{n}{16} + 8K \quad \times 8$$

$\frac{n}{2^x} = 1 \quad n = 2^x$

$x = \log_2 n$

$$T(n) = \underbrace{n+n+\cdots+n}_{x \text{ times}} + \underbrace{K+2K+4K+\cdots}_{x \text{ times}} + 2T(0) + n + K$$

$$= n\log_2 n + K(1+2+\cdots+2^{x-1})$$
$$= n\log_2 n + K(2^x - 1) \approx 2^x$$
$$= n\log_2 n + n$$
$$T(n) = O(n\log n)$$

Program:

```
public static void quickSort(int[] arr, int lo, int hi) {
    if (lo > hi)
        return;
    int pivot = arr[hi];
    int pi = partition(arr, pivot, lo, hi);
    quickSort(arr, lo, pi-1);
    quickSort(arr, pi+1, hi);
}
```

10. Quick Select

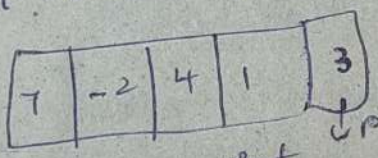
Sample Input	Sample Output	Constraints
5 → no of elements	pivot → 3	1 <= N <= 100000
7	swapping 2 and 7	-10^9 <= arr[i] <= 10^9
-2	swapping 1 and 7	1 <= K <= N
4	swapping 3 and 4	
1		
3	pivot index → 2	
3 → K	3	

Steps:

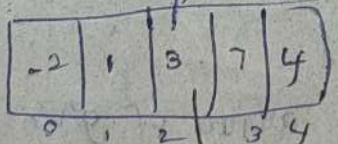
1) Assign pivot as hi (last element)
2) Then do the partition, Get the pivot index
3) Now compare the pivot index with k
4) If k is smaller than pivot index search in the first half.
5) If k is greater than pivot index, search in the second half. (3)
6) Else return the arr[pi] → which gives the k^{th} [k-1 elements gives kth smallest]
   smallest element in the array.

k = 3 smallest
↓
2nd index

Dry Run:



After partition



(3) ⇒ satisfies 3rd smallest is 3

Program:

```
public static int quickSelect (int[] arr, int lo, int hi, int k){
    int pivot = arr[hi];
    int pi = partition (arr, pivot, lo, hi);
    if (k < pi)
        return quickSelect (arr, lo, pi-1, k);
    else if (k > pi)
        return quickSelect (arr, pi+1, hi, k);
    else return arr[pi];
}
```

Worst case; Smallest, last before index $\Rightarrow O(n^2)$

$T(n) = n + k + T(n-1)$

Best/Avg case; $T(n) = n + k + T(n/2)$

$= n + n/2 + T(n/4)$

$= n + n/2 + n/4 + \cdots + n/n$

$= n\left(n + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log_2(n)}}\right)$ ⟩ By G.Series

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

Note;

For Array.sort() Small Elements Insertion sort Large Elements Dual Pivot Quick Sort

11. Count Sort

Input:

5
7
-2
4
1
3

Sample Output

-2
1
3
4
7

Constraints

$1 <= N <= 100000$

$-10^8 <= arr[i] <= 10^8$

Steps:

1. Take the Given Array
2. Construct the frequency array of given Array. [by Max, Min Element from Array + 1]
3. Construct the prefix sum array from frequency array
4. Construct the Ans Array by traversing in reverse order from orginal array, with subtracting element from min [As index starts from 0).
5. Get the position, and place the element in

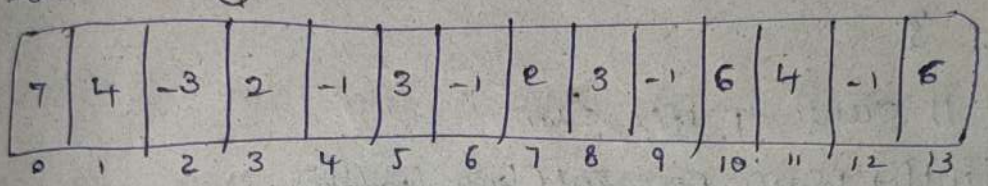
ans array with pos-1.

6. Decrement the value of prefixsum at that index.

7. Copy the entire ans Array → Orginal Array.

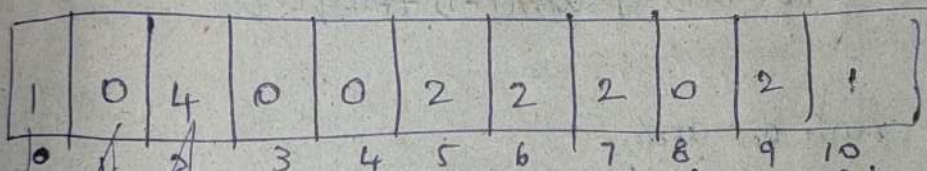
DryRun;

Orginal Array :



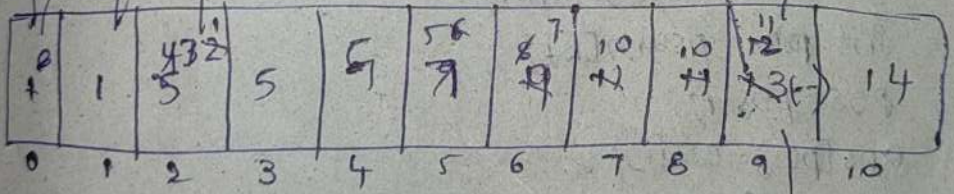
Construct Freq Array; length = (max - min + 1)

= (7 - (-3) + 1) = 10



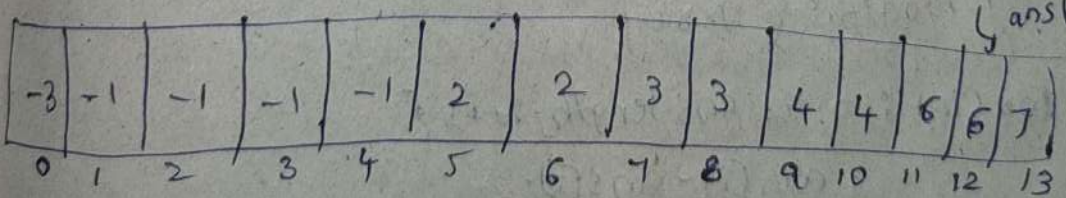
Construct PrefixSum Array

psum[k] - ans[i] = 6, 6 - min = 6 - (-3) = 6 + 3 = 9



Construct Ans Array

pos = pos - 1
pos = 12
ans[pos] = ans[i]



Program:

```
public static void countSort(int[] arr, int min, int max){

    int[] freq = new int[max-min+1];

    // Create freq array
    for(int i=0; i<arr.length; i++){
        freq[arr[i]-min]++;
    }

    // create prefix sum
    int[] psum = new int[freq.length];
    psum[0] = freq[0];
    for(int i=1; i<freq.length; i++){
        psum[i] = psum[i-1] + freq[i];
    }

    // create ans array
    int[] ans = new int[arr.length];
    for(int i=arr.length-1; i>=0; i--){
        int k = arr[i]-min;
        int pos = psum[k];
        pos = pos-1;
        ans[pos] = arr[i];
        psum[k]--;
    }

    // Modifying Array
    for(int i=0; i<ans.length; i++){
        arr[i] = ans[i];
    }

}
```

12. Radix Sort

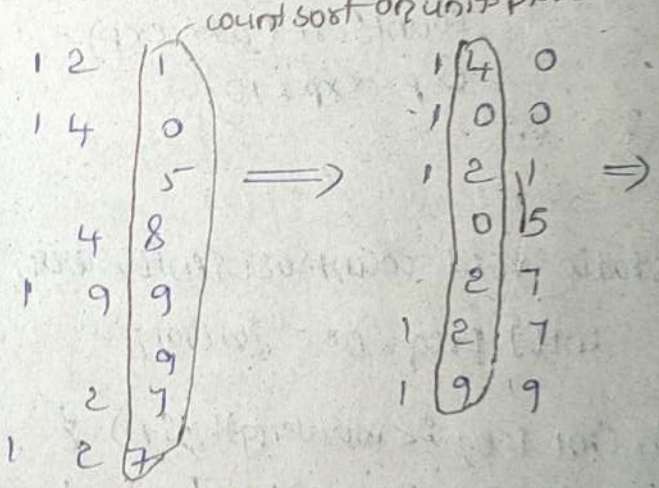
Sample Input:	Sample Output	Constraints
5 7 2 4 1 3	After sorting on 1 place → 1 2 3 4 7 1 2 3 4 7	0 <= N <= 10000 0 <= arr(i) <= 10^8

Steps:

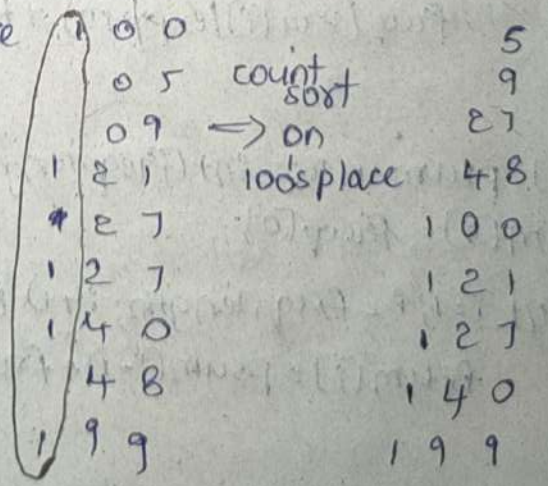
1. Get the maximum element from Array
2. Now pass Array to countsort in multiples of exponent, till it doesn't exceed the max value of array.
3. Modify the count sort code

Dry Run:

Eg:



count sort on 100's place



For getting units place

(arr[i]/1)%10, tensplace: (arr[i]/10)%10

Hundreds place: (arr[i]/100)%10

Program:

```
public static void radixSort(int[] arr) {
        int max = arr[0];
        for(int i=1; i<arr.length; i++) {
            if(max<arr[i]) {
                max = arr[i];
            }
        }
        int exp=1;
        while(exp<=max) {
            countSort(arr, exp);
            exp = exp*10;
        }
}
public static void countSort(int[] arr, int exp) {
        int[] freq = new int[10];
        for(int i=0; i<arr.length; i++) {
            freq[(arr[i]/exp)%10]++;
        }
        int[] psum = new int[freq.length];
        psum[0] = freq[0];
        for(int i=1; i<freq.length; i++) {
            psum[i] = psum[i-1]+freq[i];
        }
}
```

```
int[] ans = new int[arr.length];
for(int i = arr.length-1; i>=0; i--) {
    int k = arr[i];
    int pos = psum[(k/exp)%10];
    pos = pos-1;
    ans[pos] = arr[i];
    psum[(k/exp)%10]--;
}
for(int i=0; i<ans.length; i++) {
    arr[i] = ans[i];
}
System.out.print("After sorting on "+ exp + "place -> ");
print(arr);
}
```

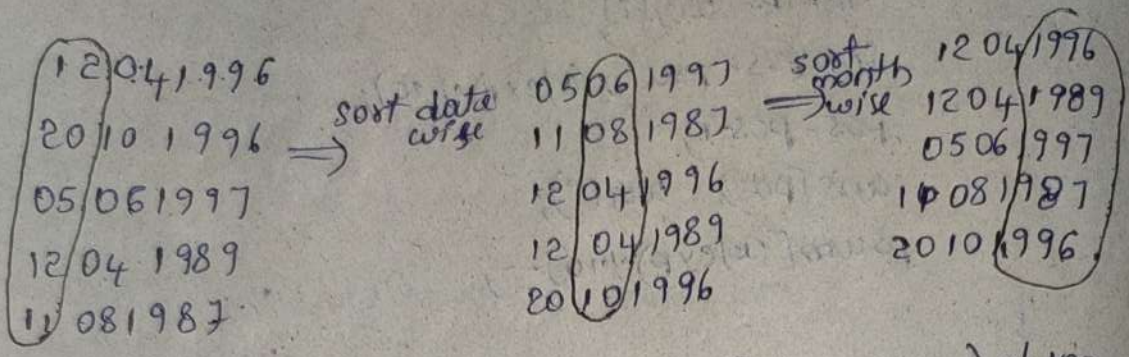
13. Sort Dates

Sample Input	Sample Output	Constraints
5	11081987	1<=N<=10000
12041996	12041989	All dates are between year 0 to year 2500.
20101996	12041996	
05061997	20101996	
12071989	05061997	
11081987		

Steps:

1. Apply the count sort through Days wise then month's wise, then years wise
2. As input is in string format, pare them into Integer format. (use paramder as 10, to get in decimal format).

Dry Run;



sort year wise ⟹
11/08/1987
12/04/1989
12/04/1996
20/10/1996
05/06/1997

To get Date; (arr/1000000) %100
month; (arr/10000) %100
years; (arr/1) %10000

Freq Array Range should be greater than days (32)
As Array starts with 0, [1-32), 1y Months,
1y for Years

Program;

```
public static void sortDates (String[] arr){
    countSort(arr, 1000000, 100, 32);   // days
    countSort (arr, 10000, 100, 13);    // months
    countSort (arr, 1, 10000, 2501);    // years
}
public static void countSort(String [] arr, int div, int
  mod, int range){
    int[] freq = new int[range];
    For(int i=0; i<arr.length; i++){
        freq[Integer.parseInt(arr[i], 10)/div %mod]++;
    }
}
```

```
int[] psum = new int[freq.length];
    psum[0] = freq[0];
for(int i=1; i<freq.length; i++){
    psum[i] = psum[i-1] + freq[i];
}
String[] ans = new String[arr.length];
for(int i=arr.length-1; i>=0; i--){
    int k = Integer.parseInt(arr[i], 10);
    int pos = psum[(k)div).%.mod]-1;
    pos = pos-1;
    ans[pos] = arr[i];
    psum[(k)div).%.mod]--;
}
for(int i=0; i<ans.length; i++){
    arr[i] = ans[i];
}
}
}
```