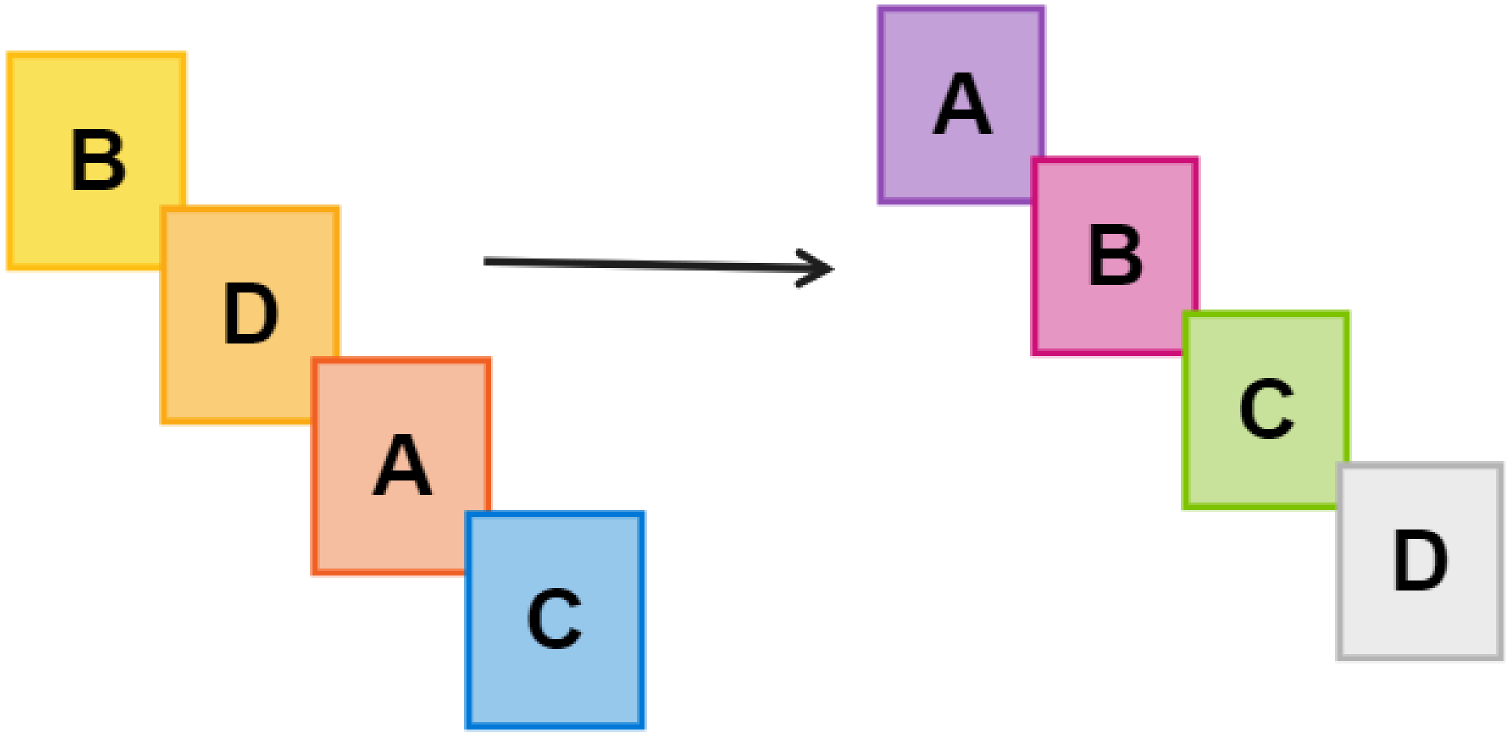
ALGORITHM STEPS FOR RADIX SORT

SORTING Presentation



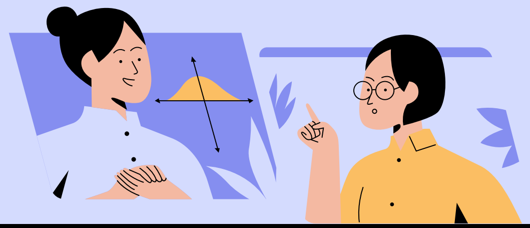
UNDERSTAND THE OBJECTIVE

We have been given an array and we have to sort the Array using Radix Sort

170	45	75	90	802	24	2	66

0	1	2	3	4	5	6	7

ALGORITHM



1. Identify the maximum number in the array:
   - Find the largest number in the array to determine the number of digits `d`.
2. Initialize sorting by digits:
   - Start sorting from the least significant digit (LSD) to the most significant digit (MSD).
3. Counting sort on each digit:
   - Initialize a counting array: Create a count array to store the occurrences of each digit (0–9).
   - Count occurrences: For each element in the array, extract the current digit and increment the corresponding count in the count array.
   - Calculate positions: Convert the count array to represent the cumulative position of each digit.
   - Sort based on the current digit: Build a temporary sorted array by placing elements at their correct position based on the cumulative count of the current digit.
   - Copy to the original array: Copy the sorted elements back into the original array.
4. Repeat for each digit:
   - Move to the next significant digit and repeat the steps 2 and 3 until all digits are processed.
5. End: The array will be sorted once all digits from LSD to MSD have been processed.

PSEUDO CODE

```
procedure radixSort(A: array of n elements)
    maxElement = findMax(A)
    exp = 1
    while maxElement / exp > 0 do
        countingSort(A, exp)
        exp = exp * 10
    end while
end procedure
```

```
procedure findMax(A: array of n elements)
    maxElement = A[0]
    for i from 1 to n-1 do
        if A[i] > maxElement then
            maxElement = A[i]
        end if
    end for
    return maxElement
end procedure
```

PSEUDO CODE

```
procedure countingSort(A: array of n elements, exp)
 n = length(A)
 output = array of size n
 count = array of size 10, initialized to 0

 for i from 0 to n-1 do
      index = (A[i] / exp) % 10
      count[index] = count[index] + 1
 end for

 for i from 1 to 9 do
       count[i] = count[i] + count[i - 1]
 end for

 for i from n-1 downto 0 do
        index = (A[i] / exp) % 10
        output[count[index] - 1] = A[i]
        count[index] = count[index] - 1
 end for

for i from 0 to n-1 do
         A[i] = output[i]
end for
end procedure
```

DRY RUN

arr =

170	45	75	90	802	24	2	66
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

n = 8

current_significant_bit(exp) = 1

max_element = 170

count =

2	0	2	0	1	2	1	0	0	0
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

```
for i from 0 to n-1 do
  index = (arr[i] / current_significant_bit) % 10
  count[index] = count[index] + 1
end for
```

comm_count =

2	2	4	4	5	7	8	8	8	8
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

output =

0	0	0	0	0	0	0	0
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

DRY RUN

arr =

170	45	75	90	802	24	2	66

- *Start Traversing your array from n-1 to 0 in variable **i***
    - i = 7
    - j = (arr[i] / current_significant_bit) % 10 = 6
    - output[commu_count[j] - 1] = arr[i]
    - commu_count[j] = commu_count[j] - 1

output =

0	0	0	0	0	0	0	66
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

comm_count =

2	2	4	4	5	7	7	8	8	8
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 6
j = (arr[i] / current_significant_bit) % 10 = 2
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	2	0	0	0	66
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

comm_count =

2	2	3	4	5	7	7	8	8	8
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 5
j = (arr[i] / current_significant_bit) % 10 = 4
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	2	24	0	0	66
0	1	2	3	4	5	6	7

comm_count =

2	2	3	4	4	7	7	8	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 4
j = (arr[i] / current_significant_bit) % 10 = 2
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	802	2	24	0	0	66
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

comm_count =

2	2	2	4	4	7	7	8	8	8
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 4
j = (arr[i] / current_significant_bit) % 10 = 2
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	1	2	3	4	5	6	7
0	90	802	2	24	0	0	66

comm_count =

0	1	2	3	4	5	6	7	8	9
1	2	2	4	4	7	7	8	8	8

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 3
j = (arr[i] / current_significant_bit) % 10 = 5
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	1	2	3	4	5	6	7
0	90	802	2	24	0	75	66

comm_count =

0	1	2	3	4	5	6	7	8	9
1	2	2	4	4	6	7	8	8	8

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 2
j = (arr[i] / current_significant_bit) % 10 = 5
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

comm_count =

1	2	2	4	4	5	7	8	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	45	75	90	802	24	2	66

i = 2
j = (arr[i] / current_significant_bit) % 10 = 5
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	1	2	3	4	5	6	7
170	90	802	2	24	45	75	66

comm_count =

0	1	2	3	4	5	6	7	8	9
0	2	2	4	4	5	7	8	8	8

DRY RUN

- Now we will sort the array on the basis of second least significant bit

current_significant_bit = 10

arr =

0	1	2	3	4	5	6	7
170	90	802	2	24	45	75	66

count =

0	1	2	3	4	5	6	7	8	9
2	0	1	0	1	0	1	2	0	1

```
for i from 0 to n-1 do
  index = (arr[i] / current_significant_bit) % 10
  count[index] = count[index] + 1
end for
```

comm_count =

0	1	2	3	4	5	6	7	8	9
2	2	3	3	4	4	5	7	7	8

output =

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

DRY RUN

arr =

0	1	2	3	4	5	6	7
170	90	802	2	24	45	75	66

- *Start Traversing your array from n-1 to 0 in variable* ***i***
  - i = 7
  - j = (arr[i] / current_significant_bit) % 10 = 6
  - output[**commu_count[j] - 1**] = arr[i]
  - commu_count[index] = commu_count[index] - 1

output =

0	1	2	3	4	5	6	7
0	0	0	0	66	0	0	0

comm_count =

0	1	2	3	4	5	6	7	8	9
2	2	3	3	4	4	4	7	7	8

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 6
j = (arr[i] / current_significant_bit) % 10 = 7
output[commu_count[j] - 1] = A[i]
commu_count[index] = commu_count[index] - 1

output =

0	0	0	0	66	0	75	0
0	1	2	3	4	5	6	7

comm_count =

2	2	3	3	4	4	4	6	7	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 5
j = (arr[i] / current_significant_bit) % 10 = 4
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	45	66	0	75	0
0	1	2	3	4	5	6	7

comm_count =

2	2	3	3	3	4	4	6	7	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 4

j = (arr[i] / current_significant_bit) % 10 = 2

output[commu_count[j] - 1] = A[i]

commu_count[j] = commu_count[j] - 1

output =

0	0	24	45	66	0	75	0
0	1	2	3	4	5	6	7

comm_count =

2	2	2	3	3	4	4	6	7	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 3
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	2	24	45	66	0	75	0
0	1	2	3	4	5	6	7

comm_count =

1	2	2	3	3	4	4	6	7	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 2
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

802	2	24	45	66	0	75	0
0	1	2	3	4	5	6	7

comm_count =

0	2	2	3	3	4	4	6	7	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

0	1	2	3	4	5	6	7
170	90	802	2	24	45	75	66

i = 1
j = (arr[i] / current_significant_bit) % 10 = 9
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	1	2	3	4	5	6	7
802	2	24	45	66	0	75	90

comm_count =

0	1	2	3	4	5	6	7	8	9
0	2	2	3	3	4	4	6	7	7

DRY RUN

arr =

170	90	802	2	24	45	75	66
0	1	2	3	4	5	6	7

i = 1
j = (arr[i] / current_significant_bit) % 10 = 7
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

comm_count =

0	2	2	3	3	4	4	5	7	7
0	1	2	3	4	5	6	7	8	9

DRY RUN

- Now we will sort the array on the basis of third least significant bit

current_significant_bit = 100

arr =

0	1	2	3	4	5	6	7
802	2	24	45	66	170	75	90

count =

0	1	2	3	4	5	6	7	8	9
6	1	0	0	0	0	0	0	1	0

```
for i from 0 to n-1 do
  index = (arr[i] / current_significant_bit) % 10
  count[index] = count[index] + 1
end for
```

comm_count =

0	1	2	3	4	5	6	7	8	9
6	7	7	7	7	7	7	7	8	8

output =

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

- *Start Traversing your array from n-1 to 0 in variable **i***
    - i = 7
    - j = (arr[i] / current_significant_bit) % 10 = 0
    - output[commu_count[j] - 1] = arr[i]
    - commu_count[index] = commu_count[index] - 1

output =

0	0	0	0	0	90	0	0
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

comm_count =

5	7	7	7	7	7	7	7	8	8
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 6
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	0	75	90	0	0
0	1	2	3	4	5	6	7

comm_count =

4	7	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 5
j = (arr[i] / current_significant_bit) % 10 = 1
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	0	75	90	170	0
0	1	2	3	4	5	6	7

comm_count =

4	6	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 4
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	0	66	75	90	170	0
0	1	2	3	4	5	6	7

comm_count =

3	6	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 3
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	0	45	66	75	90	170	0
0	1	2	3	4	5	6	7

comm_count =

2	6	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 2
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

0	24	45	66	75	90	170	0
0	1	2	3	4	5	6	7

comm_count =

1	6	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 1
j = (arr[i] / current_significant_bit) % 10 = 0
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

2	24	45	66	75	90	170	0
0	1	2	3	4	5	6	7

comm_count =

0	6	7	7	7	7	7	7	8	8
0	1	2	3	4	5	6	7	8	9

DRY RUN

arr =

802	2	24	45	66	170	75	90
0	1	2	3	4	5	6	7

i = 0
j = (arr[i] / current_significant_bit) % 10 = 8
output[commu_count[j] - 1] = A[i]
commu_count[j] = commu_count[j] - 1

output =

2	24	45	66	75	90	170	807
0	1	2	3	4	5	6	7

comm_count =

0	6	7	7	7	7	7	7	7	8
0	1	2	3	4	5	6	7	8	9

Now our Array has become completely Sorted exit the algorithm

2	24	45	66	75	90	170	807
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Time Complexity and Space Complexity:

- **Time Complexity:**
  - Worst-case: $(O(d(n+k))$, where `d` is the number of digits in the maximum number, n is the number of elements, and k is the range of the digit (typically 0-9, so k = 10).
  - Average-case: $O(d(n+k)$
  - Best-case: $O(d(n+k))$
- **Space Complexity:** O(n + k) (for the output array and the count array)

Sorting Algorithm	Best Case Time Complexity	Average Case Time Complexity	Worst Case Time Complexity	Space Complexity
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)