# 9 Radix Sort

# Radix Sort

Sort an array of numbers, assuming each number has k digits (why is this often reasonable?)

- Use stable sort to sort them on the k-th digit
- Use stable sort to sort them on the (k-1)-th digit
- …
- Use stable sort to sort them on 1st digit

| | Sort on $4^{th}$ digit | Sort on $3^{rd}$ digit | Sort on $2^{nd}$ digit | Sort on $1^{st}$ digit |
|---|---|---|---|---|
| 7 5 5 5 | 1 1 9 1 | 3 5 1 2 | 1 1 8 2 | 1 1 8 2 |
| 4 6 4 2 | 4 6 4 2 | 5 4 1 2 | 1 1 9 1 | 1 1 9 1 |
| 3 5 1 2 | 3 5 1 2 | 1 3 2 3 | 6 2 8 4 | 1 3 2 3 |
| 1 3 2 3 | 6 6 8 2 | 9 5 2 3 | 1 3 2 3 | 3 5 1 2 |
| 3 7 8 4 | 5 4 1 2 | 4 6 4 2 | 9 3 5 6 | 3 7 8 4 |
| 6 2 8 4 | 1 1 8 2 | 7 5 5 5 | 5 4 1 2 | 4 6 4 2 |
| 6 6 8 2 | 1 3 2 3 | 9 3 5 6 | 3 5 1 2 | 5 4 1 2 |
| 9 5 2 3 | 9 5 2 3 | 6 6 8 2 | 9 5 2 3 | 6 2 8 4 |
| 1 1 9 1 | 3 7 8 4 | 1 1 8 2 | 7 5 5 5 | 6 6 8 2 |
| 9 3 5 6 | 6 2 8 4 | 3 7 8 4 | 4 6 4 2 | 7 5 5 5 |
| 5 4 1 2 | 7 5 5 5 | 6 2 8 4 | 6 6 8 2 | 9 3 5 6 |
| 1 1 8 2 | 9 3 5 6 | 1 1 9 1 | 3 7 8 4 | 9 5 2 3 |

The simplest invariant we can see is Least Significant digit (LSD) radix sort, which works by sorting an arr of elements 1 digit at a time, from least to most significant.
Sort the array using a stable sorting algorithm, specifically, counting sort.

We will use counting sort for $\underline{\underline{k}}$ times ($k = \#$ of digits)

Time $= k \cdot O(n+u) = \boxed{O(k(n+u))}$

$u = 10$ because a single digit max num is 9. So, $u = 9+1 = 10$

Time $= O(k(n+b))$

b can reflect any base.

$k = \#$ of digits

since, we are using nums, $b = 10$. If say we are using alphabets $b = \#$ of letters $+ 1$

Lets say a num is 100. $k = 3$, cuz 100 has 3 digits

OR

$$\lfloor \log_{10}(100) \rfloor + 1 = 2 + 1 = 3$$

So,

$$k = \lfloor \log_b(M) \rfloor + 1$$

$$k = O(\log_b M)$$

$$\boxed{\text{Time} = O(\log_b M (N + b))}$$

- Each count sort will be $O(N)$ as long as $\overset{u \le N}{b \le N}$ OR $b = O(N)$

Lets say $b = N$, then $\text{Time} = O(\log_N M (N + N)) = O(\log_N M \cdot N)$

What can the value of M be so time is $O(N)$?

$O(\log_N M \cdot N)$, if $(\log_N M = 1) \Rightarrow N^1 = M$ OR $M = N^c$, then $\text{Time}: O(N)$

So, when $M = N^c$, then time $= O(\log_N(N^c) \cdot N) = O(c \cdot N) = O(N)$

NOTE: M = max num in list, so $M \le N^c$ means radix sort can sort a large range of values in O(n) time, when compared to counting sort, which sort in O(n) time when max num is $\le N$

**Algorithm 20** Radix sort

```
function RADIX_PASS(array[1..n], base, digit)
    counter[0..base-1] = [0,0,...],
    for i = 1 to n do
        counter[GET_DIGIT(array[i], base, digit)] += 1
    position[0..base-1] = [1,0,...]
    for v = 1 to base−1 do
        position[value] = position[v−1] + counter[v−1]
    temp[1..n] = [0,0,...]
    for i = 1 to n do
        digit = GET_DIGIT(array[i],base,digit)
        temp[position[digit]] = array[i]
        position[digit] += 1
    swap(array, temp)

function RADIX_SORT(array[1..n], base, digits)
    for digit = 1 to digits do
        RADIX_PASS(array[1..n], base, digit)
```

O(n) Adds # of occurrences of a num into counter

O(b) Finds the pos for each num

O(n) Sorts the arr using pos

$K \cdot O(n+b) = O(K \cdot (n+b))$ From LST to MSD, arr is sorted.

| | Best case | Average Case | Worst Case |
|---|---|---|---|
| Time | $O(k(n+b))$ | $O(k(n+b))$ | $O(k(n+b))$ |
| Auxiliary Space | $O(n+b)$ | $O(n+b)$ | $O(n+b)$ |

NOT an

Note: The # of digits (k) and the base (b) are NOT independent, BUT related.

$K = \log_b M$ OR $K = \frac{w}{\log b}$

$O(\log_b M \cdot (n+b))$ $O(\frac{w}{\log b} \cdot (n+b))$

A larger base (b) means no. of count sorts go DOWN, Because of $K = \log_b M$

cost of each count sort goes UP, Because of $(n+b)$

- $O\left(\frac{w}{\log b} \cdot (n+b)\right)$

- Lets say we are sorting integers and choose base to be constant $(b=10)$ and max. value is $O(n)$, i.e. integers with $\log(n) + O(1)$ bits.

- So, time $= O\left(\frac{\log n + 1}{\log(10)} (n+10)\right) = O(n \log n)$

To make a radix sort faster we need to choose b's val. more cleverly:

Counting sort is effecient till $b \leq n$ OR $b = O(n)$.
So, if we pick $b=n$, the time complexity of counting sort $= O(n+n) = O(n)$

- So, Radix sort time $= O\left(\frac{w}{\log n} \cdot n\right)$

- We know that $w = \log n + O(1)$ bits $\Rightarrow O(\log n)$

- So, Radix sort Time $= O\left(\frac{\log n}{\log n} \cdot n\right) = \underline{O(n)!}$

Space $: O(n+b)$
$: O(n)$, $b=n$

- Notice that integers with $w = c \cdot \log n$ bits have max size of $O(2^{c \cdot \log n}) = O(n^c)$, which is better than counting sort, which can only handle ingers of size $O(n)$.

Radix sort is better becaus it can handle a larger size of nums that is $O(n^c)$,
whereas, counting can only handle $O(n)$ size of nums

---

## Radix Sort for strings:

$$\begin{bmatrix} A & B & C \\ Z & B & O \\ P & Q & R \end{bmatrix}$$

Simpliy sort for each char left to right using ASCII values.

In ASCII encoding (like UTF-8), there are $2^8 = 256$ possible chars, so <u>base = 256</u> !

---

```
def get_digit(num, base, digit):
    # Get the digit at the specified position
    return (num // (base ** (digit - 1))) % base

def radix_pass(arr, base, digit):
    n = len(arr)
```

```
    counter = [0] * base
    # Count occurrences of each digit
    for i in range(n):
        counter[get_digit(arr[i], base, digit)] += 1

    position = [0] * base
    # Calculate positions of each digit in the sorted array
    for v in range(1, base):
        position[v] = position[v - 1] + counter[v - 1]

    temp = [0] * n
    # Place elements in temporary array according to their digit
    for i in range(n):
        d = get_digit(arr[i], base, digit)
        temp[position[d]] = arr[i]
        position[d] += 1

    # Copy elements back to original array
    for i in range(n): arr[i] = temp[i]


def radix_sort(arr, base, digits):
    for digit in range(1, digits + 1):
        radix_pass(arr, base, digit)


# Example usage:
arr = [170, 45, 75, 90, 802, 24, 2, 66]
radix_sort(arr, 10, 3)  # Sorting base 10 integers with maximum of 3 digits
print("Sorted array:", arr)




def radix_pass_strings(arr, digit):
    n = len(arr)
    b = 256
```

```
    counter = [0] * b  # Assuming ASCII characters, you can adjust this based on your character set
    # Count occurrences of each character
    for string in arr:
        if digit < len(string):
            counter[ord(string[digit])] += 1

    position = [0] * b
    # Calculate positions of each character in the sorted array
    for v in range(1, b):
        position[v] = position[v - 1] + counter[v - 1]

    temp = [''] * n
    # Place strings in temporary array according to their character
    for string in arr:
        if digit < len(string):
            d = ord(string[digit])
            temp[position[d]] = string
            position[d] += 1

    # Copy strings back to original array
    for i in range(n): arr[i] = temp[i]


def radix_sort_strings(arr, max_length):
    # Perform Radix Pass for each character position in strings (starting from right most)
    for digit in range(max_length - 1, -1, -1):
        radix_pass_strings(arr, digit)


# Example usage:
arr = ['ACB', 'ABC', 'BZA', 'ZAP', 'BAC']
max_length = max(len(s) for s in arr)  # Find the maximum length of strings
radix_sort_strings(arr, max_length)
print("Sorted array:", arr)
```

---

**Algorithm 20** Radix sort

```
function RADIX_PASS(array[1..n], base, digit)
    counter[0..base-1] = [0,0,...],
    for i = 1 to n do
        counter[GET_DIGIT(array[i], base, digit)] += 1
    position[0..base-1] = [1,0,...]
    for v = 1 to base−1 do
        position[value] = position[v − 1] + counter[v − 1]
    temp[1..n] = [0,0,...]
    for i = 1 to n do
        digit = GET_DIGIT(array[i],base,digit)
        temp[position[digit]] = array[i]
        position[digit] += 1
    swap(array, temp)

function RADIX_SORT(array[1..n], base, digits)
    for digit = 1 to digits do
        RADIX_PASS(array[1..n], base, digit)
```

| | **Best case** | **Average Case** | **Worst Case** |
|---|---|---|---|
| Time | $O(k(n+b))$ | $O(k(n+b))$ | $O(k(n+b))$ |
| Auxiliary Space | $O(n+b)$ | $O(n+b)$ | $O(n+b)$ |