Counting Sort

https://www.youtube.com/watch?v=wP_GoeEPB1w

```
CountingSort(array, n, k)
    # array: input array of length n
    # k: maximum value in array (range 0..k)

    # 1. Create count array of size (k + 1) initialized to 0
    let count[0..k] = {0}

    # 2. Count the occurrences of each element
    for i ← 0 to n - 1
        count[array[i]] = count[array[i]] + 1

    # 3. Compute cumulative counts (prefix sums)
    for i ← 1 to k
        count[i] = count[i] + count[i - 1]

    # 4. Create output array of size n
    let output[0..n - 1]

    # 5. Place elements into output array in sorted order
    # (go backwards to make the sort stable)
    for i ← n - 1 downto 0
        value = array[i]
        count[value] = count[value] - 1
```

```
    position = count[value]
    output[position] = value

# 6. Copy sorted elements back to original array
for i ← 0 to n - 1
    array[i] = output[i]
```

Radix sort

https://www.youtube.com/watch?v=gDF89ee79P0

Exercise important for CT+Term

1. Complete the algorithm and identify it

Algorithm 1

```
1:  procedure COUNTINGSORT(A, n, k)
2:      int Count[0..k-1] = {0};
3:      int Output[0..n-1];

4:      for i = 0 to n - 1 do
5:          Count[ ..... ] = Count[ ..... ] + 1;
6:      end for

7:      for i = 1 to k - 1 do
8:          Count[i] = Count[i] + Count[ ..... ];
9:      end for

10:     for i = n - 1 to 0 do
11:         Output[Count[A[i]] - 1] = A[i];
12:         Count[ ..... ] = Count[ ..... ] - 1;
13:     end for


14:     for i = 0 to n - 1 do
```

```
15:        A[i] = Output[i];
16:     end for
17: end procedure
```

2. You have a list of phone extensions in a company: Extensions = {678, 123, 890, 456, 234, 789, 12}. You are asked to sort this list by processing the numbers digit by digit, starting from the least significant digit to the most significant digit, without comparing the entire numbers directly. Also write the pseudocode.

3. You're sitting with a pile of playing cards. You pick up one card at a time and insert it into its correct position among the cards already in your hand.
   Each time you:
   - Look from **right to left** through your hand,
   - Find the correct spot for the new card,
   - Insert it carefully.
   - By the end, your hand is completely sorted.

4. You have a row of cards on the table.
   For each step:

   - You **look through all the remaining cards**,
   - Find the **smallest card**,
   - Swap it with the first card in the unsorted portion of the row.

5. Imagine a group of students standing in a line in **random order** of height.

   - You start at the **left end** of the line.
   - Compare the **first two students**:
     - If the one on the left is **taller**, they **swap places**.

Some more scenarios

Scenario	Best Sorting Algorithm	Why	Worst Algorithm	Why
Almost sorted array	Insertion Sort	O(n) when array is nearly sorted; very few shifts needed	Selection Sort	Always $O(n^2)$; scans entire unsorted part every time; no early stop
Reverse sorted array	Counting / Radix Sort	O(n + k) or O(d × (n + k)); not affected by input order	Insertion / Bubble Sort	Worst case $O(n^2)$ due to maximum number of shifts or swaps
Numbers with ≤ 5 digits	Counting / Radix Sort	Linear time; ideal for small digit or small range inputs	Selection / Bubble Sort	$O(n^2)$; inefficient compared to linear-time non-comparison sorts
Already sorted array (special case)	Insertion / Bubble Sort	Detects sorted order early → O(n)	Selection Sort	Still scans entire list even when no work is needed

- Why don't we always use counting sort or radix sort? why we use the rest three sometimes?