

8 Counting Sort

Theorem: Lower bound on comparison-based sorting

Sorting in the comparison model takes $\Omega(n \log(n))$ in the worst case. In other words, any comparison-based sorting algorithm can not run faster than $O(n \log(n))$ in the worst case. Recall that Ω notation denotes an asymptotic lower bound, i.e. the algorithm must take **at least** this long.

Can we get better time than $O(n \log n)$?

If we work in the Comparison Model, meaning the only valid operations performed on elements are $(<, \leq, >, \geq, =, \neq)$ then sorting can NOT possibly be done faster than $O(n \log n)$ time

Counting Sort:

arr: [3, 1, 0, 2, 1, 3, 2]
idx: 0 1 2 3 4 5 6



Step 1: count occurrences of each element in arr

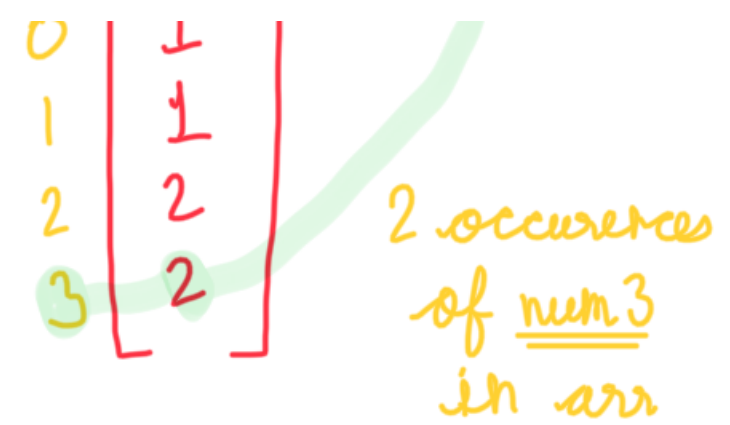
$$u = \max(arr) + 1 = 3 + 1 = 4$$

counter
[1]

```

counter = [0] * n
for num in arr:
    counter[num] += 1

```

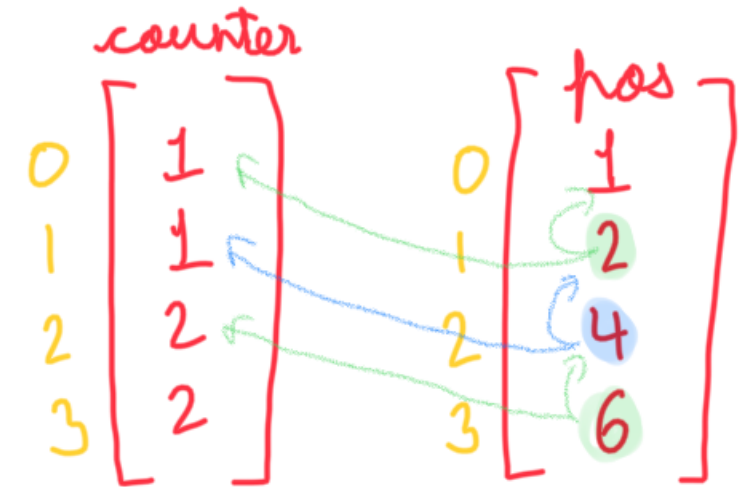


Step 2: calculate positions

```

pos = [0] * n
pos[0] = 1
for i in range(1, n):
    pos[i] = pos[i-1] + counter[i-1]

```



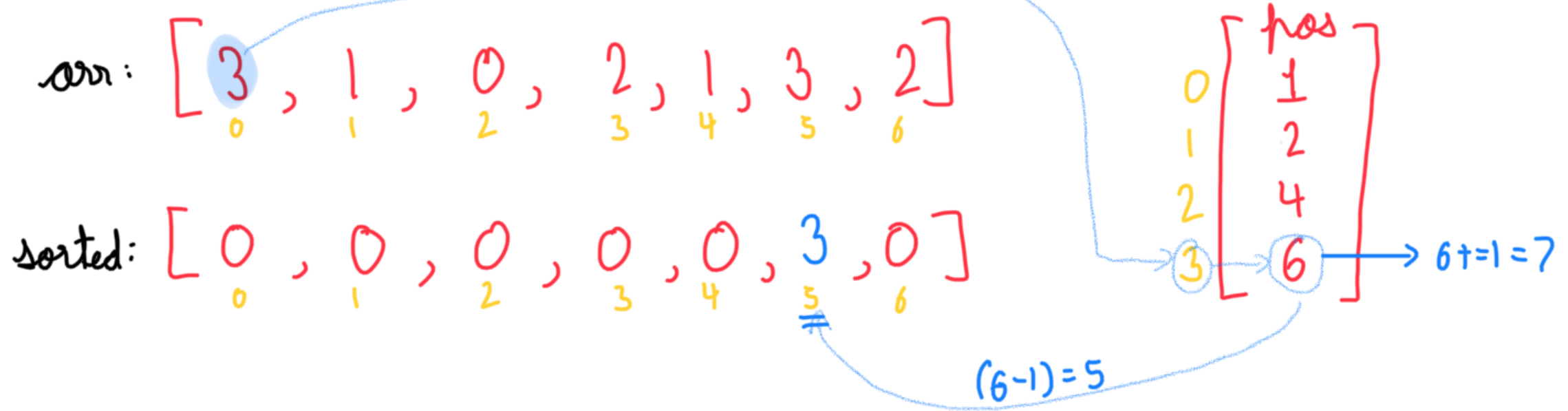
Step 3: sort the arr, using pos

```

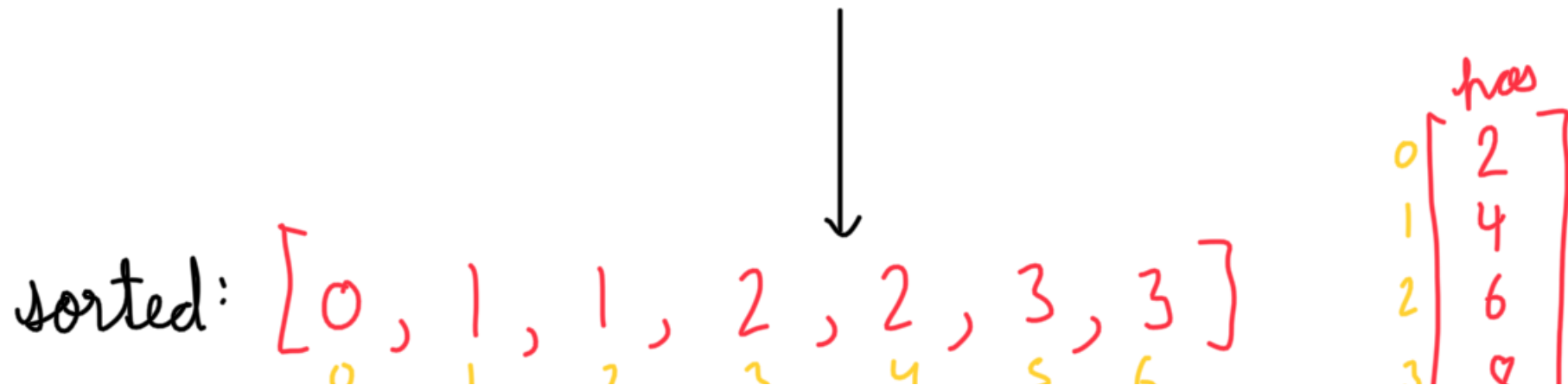
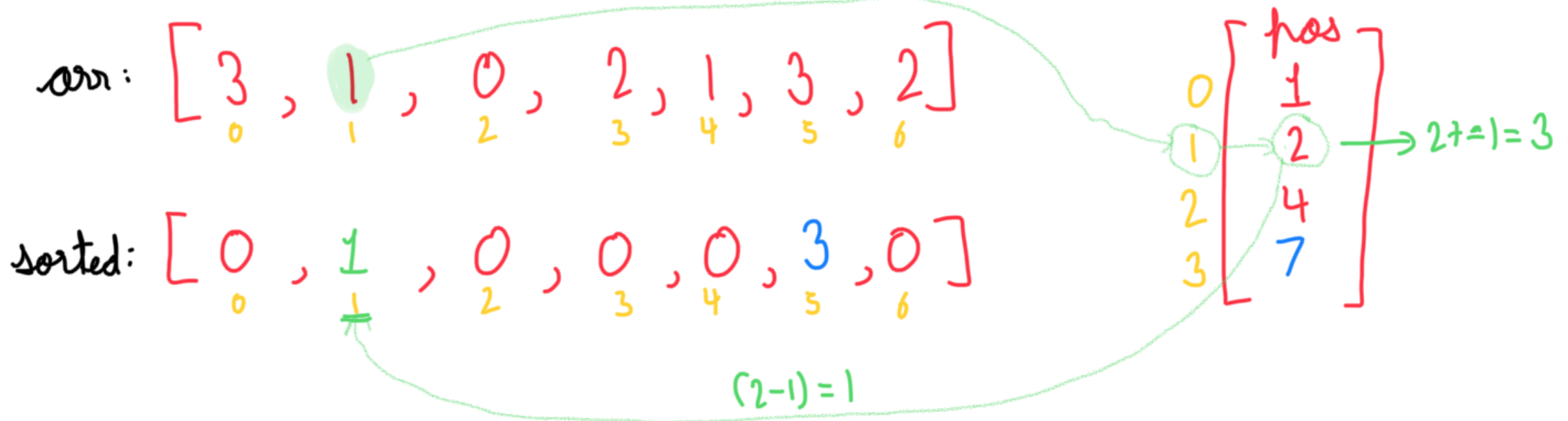
sorted = [0] * len(arr)
for num in arr:
    sorted[pos[num] - 1] = num
    pos[num] += 1

```

pos[3] - 1 shows the first position of num 3 in sorted arr.



pos $[1] - 1$ shows the first position of num1 in sorted arr.



- Not an in-place algorithm \rightarrow Aux-space = $O(n)$
- Stability is maintained

def counting_sort(arr[1...n], u):

 counter[0...u-1] = [0, 0, ...]

 for (i = 1 to n):
 counter[arr[i]] += 1

Time (Worst / Average / Best) = $O(n + u)$

space = $O(n + u)$

 pos[0...u-1] = [0, 0, ...]

 for (v = 1 to n):
 pos[v] = pos[v-1] + counter[v-1]

 temp[1...n] = [0, 0, ...]

 for (i = 1 to n):
 temp[pos[arr[i]]] = arr[i]
 pos[arr[i]] += 1

swap(arr, temp)

counting sort takes $O(n)$ time iff $u \leq n$ or $u = O(n)$

counting sort becomes inefficient when $u \gg n$.

An alternate way to see this:

This tells us that counting sort takes linear time in n if and only if $u = O(n)$. An alternative way to analyse integer sorting algorithms is to consider the *width* of the integers, rather than the maximum value u . The width of an integer is the number of bits required to represent it in binary. If we are sorting w -bit integers, then our universe size is $u = 2^w - 1$. The time complexity of counting sort on w -bit integers is therefore $O(n + 2^w)$, which is linear in n if and only if

$$w = \log(n) + O(1).$$

Code:

```
def countingSort(arr):
```

```
    u = max(arr) + 1
```

```
    n = len(arr)
```

```
    counter = [0] * u
```

```
    for num in arr:
```

```
        counter[num] += 1
```

```
    # Create a position array to store the starting position of each element in the sorted array
```

```
    pos = [0] * u
```

```
    for i in range(1, u):
```

```
        pos[i] = pos[i-1] + counter[i-1]
```

```
sorted = [0] * n
for num in arr:
    sorted[pos[num]] = num
    pos[num] += 1

return sorted
```

Time:

Best / Worst-case: $O(n + u)$

Space: $O(n + u)$

In-Place: No, aux-space is not $O(1)$.

Stable: Yes, pos array gets the first idxs to be placed in sorted array.