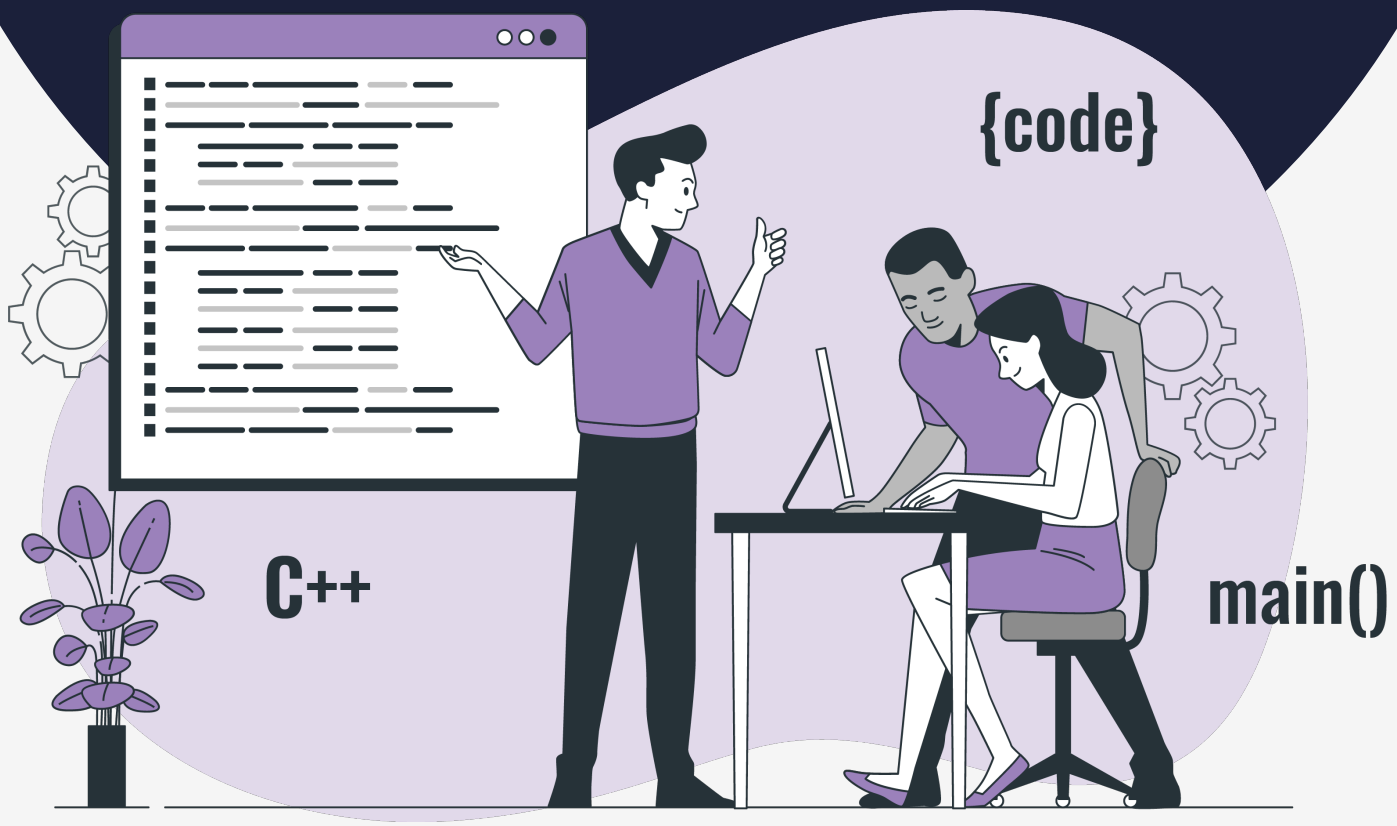
Lesson:

Advanced Sorting Algorithms



PW SKILLS

Pre-Requisites

- Cpp loops
- Cpp array

List of Concepts Involved

- Count Sort
- Radix Sort

Topic 1: Count sort

This sorting technique is used when the range of input is limited.This sort is a sorting technique which is based on the range of input value. For example, if we use the array arr = {1, 2, 1, 3, 4, 5}. All the values of the array lie between 1 and 5. So we can use this method.

Steps

1. Find out the maximum element (let it be max) from the given array.
2. Initialize an array of length at least max+1 with all elements 0. This array is used for storing the frequency of the elements in the array.
3. Store the frequency of each element at their respective index in freq array
4. We will be using our original array only to store the sorted values.
5. Start iterating from 0 till max element and if an index has value greater than zero then store that particular index in the original array as our sorted value and reduce the count of that index by 1. Repeat this process till that element value decreases to 0. Once done you can move to next index.
6. Since indices in the array are marked in sorted order from 0 to last index this will always ensure the array obtained will be in sorted order.

Illustration:
Consider the data in the range of 0 to 7.
Input data: {1, 4, 1, 2, 6, 5, 2}

- Find out the maximum element (let it be max) from the given array.
  max = 6
- Take a count array of length max+1 with all elements 0 to store the count of each unique object.

Count array:

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

- Store the count of each unique element in the count array, if any element repeats itself, simply increase its count.

Count array:

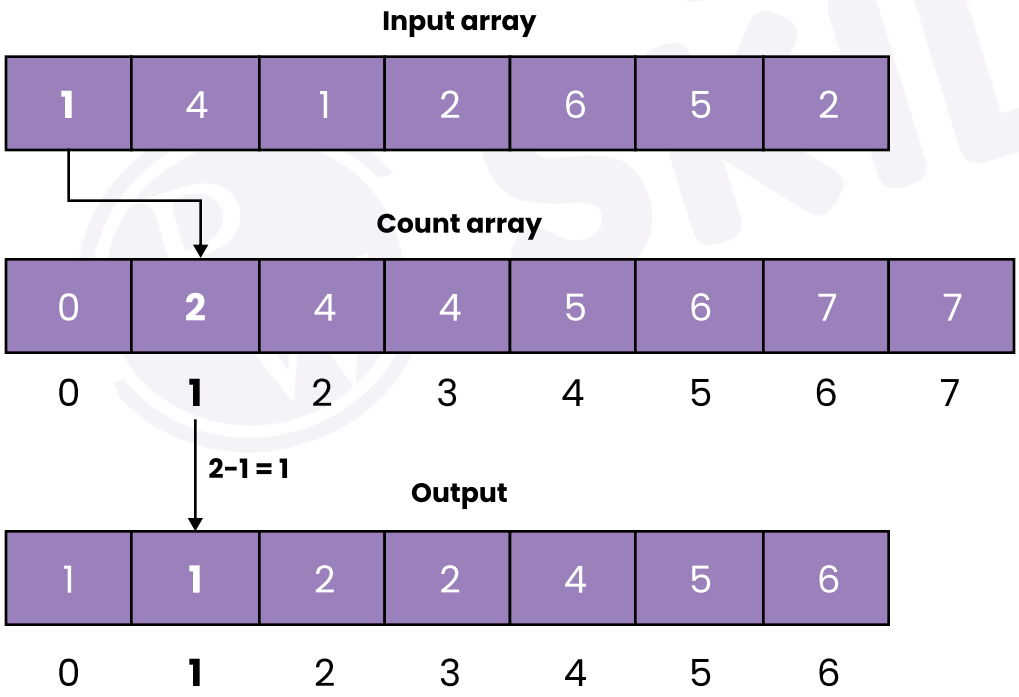
0	2	2	0	1	1	1	0
0	1	2	3	4	5	6	7

- Modify the count array such that each element at each index stores the sum of previous counts.

Count array:

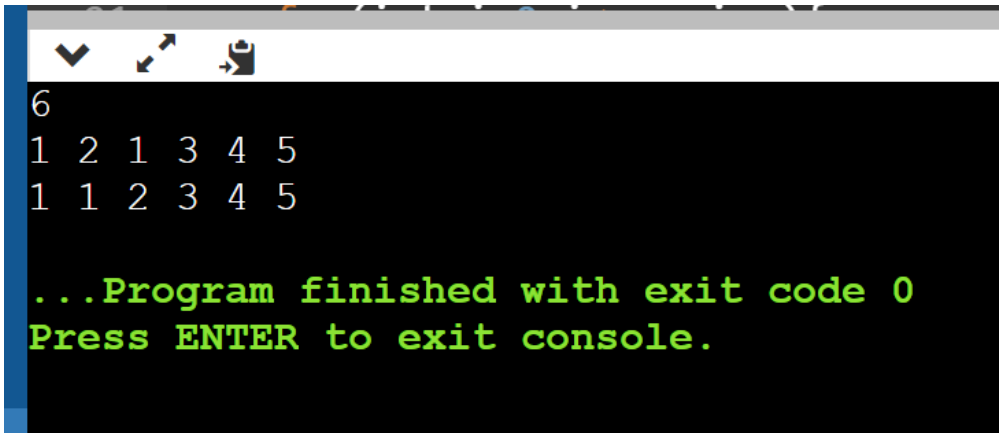
0	2	4	4	5	6	7	7
0	1	2	3	4	5	6	7

- Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated as shown in figure below.



- After placing each element at its correct position, decrease its count by one.

Code link - https://pastebin.com/tSssw5Wr



TIME COMPLEXITY:
Best case: When all items are in the same range, or when k is equal to 1, k is the time required to find the maximum number.
O(n)

Average case: k computes to (k+1/2), and the time complexity is N+(K+1)/2. = O(n+k)

Worst case: The largest element is much larger than the other elements.
O(n+k)

SPACE COMPLEXITY:
O(k)

Note:
Maximum element in an array or vector can be found with the help of *max_element() function provided in STL.

```
*max_element (first_index, last_index);
```

Topic 2: Radix sort

In Radix sort, digit by digit sorting is performed that is started from the least significant digit to the most significant digit.
Suppose, we have an array of 5 elements. First, we will sort elements based on the value of the unit place. Then, we will sort elements based on the value of the tenth place. This process goes on until the last significant place

Steps:

1. Find the maximum element in the array, i.e. max to calculate the no of digits because we have to go through all the significant places of all elements.
2. Now, go through each significant place one by one.
3. Any stable sorting technique can be used to sort the digits at each significant place.Count sort has been used here.
4. Now, sort the elements based on digits at the unit's place.
5. Now, sort the elements based on digits at tens place.
6. Finally, sort the elements based on the digits at hundreds place.

Illustration:

- Consider this input array:

170	45	75	90	802	2

- Sort on ones place.

170	90	802	2	45	75

Note, here 170 comes before 90 and 802 comes before 2 because it has appeared in this way in the original array.

- Sort on tens place.

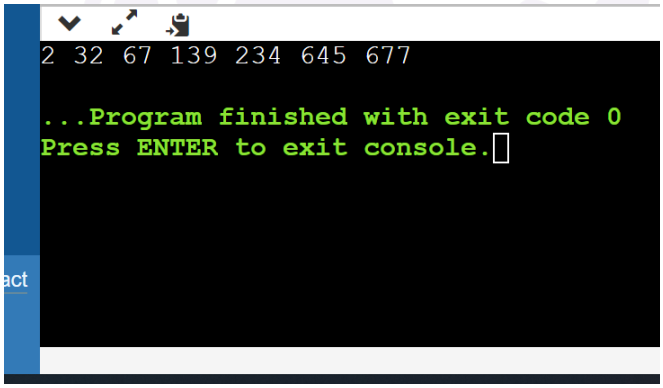
_2	802	45	170	75	90

- Sort on hundreds place.

_2	_45	_75	_90	170	802

- The array is now sorted.

Code link - https://pastebin.com/gp1ebCqd



TIME COMPLEXITY:
n = number of digits in the largest element

Best case: When all elements have the same number of digits.
O(a(n+b))

Average case: There are 'p' passes, and each digit can have up to 'd' different values. O(p*(n+d))

Worst case: When all elements have the same number of digits except one, which has a significantly larger number of digits.
$O(n^2)$

SPACE COMPLEXITY:
O(n+k)

Upcoming Class Teasers

- Bucket Sort