Advanced Sorting Algorithm

Created	@July 30, 2022 1:18 PM
Class	
Type	
Materials	
Reviewed	☐

We already know about comparison based sorting algorithms (Bubble, selection, insertion, merge, quick). Now there are a set of sorting algorithms that are based on counting.

Counting Sort

Let’s say we have the following array - [2,3,1,1,2,4,56,7,3,2,8,1,2] and we need to sort it. Let’s say we don’t have knowledge of any previous sorting algorithm. Then how we could have sorted it ?

The most intuitive approach should be, get all occurrences of the minimum element and arrange it to the start of array. Then take the second min element and all of its occurrences and arrange it after the min element and so on and so forth.
So technically, this approach is called as counting sort. We can just count the frequency of all the elements and store it in a frequency map. Then place all the occurrences of the smallest element in the start, then second smallest , then third smallest and so on and so forth.

```
[2,3,1,1,2,4,56,7,3,2,8,1,2]
frequency map - [0,3,4,2,0,0,1,1,0,0,0,0,0, ........ 1, 0,0,0,,,,]
1-3
2-4
3-2
4-1
7-1
8-1
56-1

[1,1,1,2,2,2,2,3,3,4,7,8,56]
```

How to prepare frequency map here ?

Let’s assume we have only positive elements, then we can use an array to prepare the frequency map ?

`arr[i]` → will store frequency of `i`

```
function countingSort(arr) {
  let maxElement = Math.max(...arr); // O(n)
  let freq = Array(maxElement + 1).fill(0); // O(1)
  for(let i = 0; i < arr.length; i++) { //O(n)
    freq[arr[i]]++;
  }
  let k = 0;
  for(let i = 0; i < freq.length; i++) { // O(maxElement + n)
    while(freq[i] > 0) {
      arr[k] = i;
      k++;
      freq[i]--;
    }
  }
}
```

```
freq = [0,0,0,0,0,0,1,1,0,0,0,0,0, ........ 1, 0,0,0,,,,]
arr = [1,1,1,2,2,2,2,3,3,2,8,1,2]
i = 5, k = 9

[2,1,9999999];
```

Disadvantages

- If the max element is huge, counting sort will perform very bad for even very small arrays
- Space is also extra
- Unstable → this current implementation is unstable. We can modify the implementation without hampering the time and space and make counting sort stable.

How to make it stable ?

```
arr = [2,1,3,2,1,4,5,3,2,6]
freq = [0,2,3,2,1,1,1] // convert this freq array to prefix sum array
pre-freq = [0,2,4,6,8,9,9]
final output = [0,0,0,0,2,0,3,0,0,6]
indexes -       1,2,3,4,5,6,7,8,9,10
```

Assume 1 based indexing in the final output array. If we carefully observe, then `pre-freq[i]` it is denoting the last index in the final output array, where `i` should be placed.

> Example: `pre-freq[1] = 2` → and in the final output array the last occurrence of 1 is at position `2`

Example: `pre-freq[2] = 5` → and in the final output array the last occurrence of 2 is at position `5`

Example: `pre-freq[5] = 9` → and in the final output array the last occurrence of 5 is at position `9`

Now because we know where the last occurrence of any element should be placed, can we start reading the array from behind, so that for any element we first encounter the last occurrence and then with the position of the last occurrence we calculated, we can place it.

```
function countingSortStable(arr) {
// Time: O(n+k) Space: O(n+k)
  let maxElement = Math.max(...arr); // O(n)
  let freq = Array(maxElement + 1).fill(0); // O(1)
  for(let i = 0; i < arr.length; i++) { //O(n)
    freq[arr[i]]++;
  }
// freq -> prefix sum array
  for(let i = 1; i < freq.length; i++) { // O(k)
    freq[i] = freq[i] + freq[i-1];
  }
//  console.log(freq);
  let output = Array(arr.length);
  for(let i = arr.length - 1; i >= 0; i--) { //O(n)
    let currelement = arr[i];
    let index = freq[currelement]; // index- 1 based
    output[index - 1] = currelement; // stored based on 0 based indexing
    freq[currelement]--;
  }
  return output;
}
```