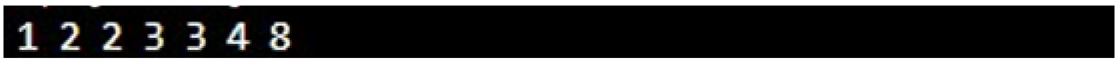
LAB 3

Counting Sort

Count Sort is a non-comparative, integer-based sorting algorithm that works by counting the occurrences of each element in the input array. It creates a count array to store frequencies, then modifies it to determine positions, and finally constructs the sorted array. It is efficient for small-range integer values but not suitable for large-value or floating-point numbers. The time complexity is O(n + k), where n is the number of elements and k is the range of values.

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

void countingSort(vector<int> &nums)
{
    int largest = INT_MIN;

    for (int num : nums)
    {
        largest = max(num, largest);
    }

    vector<int> count(largest + 1, 0);

    for (int num : nums)
    {
        count[num]++;
    }

    int index = 0;
    for (int i = 0; i <= largest; i++)
    {
        while (count[i] > 0)
        {
            nums[index++] = i;
            count[i]--;
        }
    }
}

int main()
{
    vector<int> arr = {4, 2, 2, 8, 3, 3, 1};

    countingSort(arr);

    for (int num : arr)
    {
        cout << num << " ";
    }

    return 0;
}

```



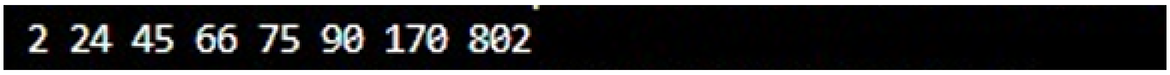
```
1 2 2 3 3 4 8
```

Radix Sort

Radix Sort is a non-comparative sorting algorithm that sorts numbers digit by digit, starting from the least significant to the most significant. It uses a stable sub-sorting algorithm like Counting Sort at each digit level. It is efficient for sorting large integers with a fixed number of digits, achieving a time complexity of

O(nk), where n is the number of elements and k is the number of digits in the largest number. Radix Sort is best suited for scenarios where data has a uniform length and range.

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
void countingSort(vector<int> &nums, int place) {
int n = nums.size();
vector<int> output(n);
vector<int> count(10, 0);
for (int num : nums) {
int digit = (num / place) % 10;
count[digit]++;
}
for (int i = 1; i < 10; i++) {
count[i] += count[i - 1];
}
for (int i = n - 1; i >= 0; i--) {
int digit = (nums[i] / place) % 10;
output[count[digit] - 1] = nums[i];
count[digit]--;
}
nums = output;
}
void radixSort(vector<int> &nums) {
int maxNum = INT_MIN;
for (int num : nums)
{
maxNum = max(num, maxNum);
}
for (int j = 1; maxNum / j > 0; j *= 10) {
countingSort(nums, j);
}
}
int main() {
vector<int> arr = {170, 45, 75, 90, 802, 24, 2, 66};
radixSort(arr);
for (int num : arr) {
cout << num << " ";
}
return 0;
}


```



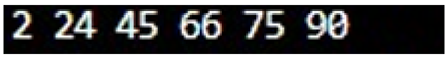
```
2 24 45 66 75 90 170 802
```

TASK 2

Suppose all the marks secured in each subject in a semester by a student is stored in an array, Assume that all the marks are distinct. Write an algorithm to find all the marks in the array that have at-least two greater marks than themselves. For example, Input: {50, 40, 10, 20, 30} Output: 10 20 30

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
void countingSort(vector<int> &nums)
{
    int largest = INT_MIN;
    for (int num : nums)
    {
        largest = max(num, largest);
    }
    vector<int> count(largest + 1, 0);
    for (int num : nums)
    {
        count[num]++;
    }
    int index = 0;
    for (int i = 0; i <= largest; i++)
    {
        while (count[i] > 0)
        {
            nums[index++] = i;
            count[i]--;
        }
    }
}
vector<int> function(vector<int>&nums){
    countingSort(nums);
    int n = nums.size();
    vector<int>result;
    if(n==0 || n==1 || n==2){
        return result;
    }
    else{
        for(int i=0;i<n-2;i++){
            result.push_back(nums[i]);
        }
    }
    return result;

}
int main() {
    vector<int> arr = {170, 45, 75, 90, 802, 24, 2, 66};
    vector<int>main = function(arr);
    for (int num : main) {
        cout << num << " ";
    }
    return

```

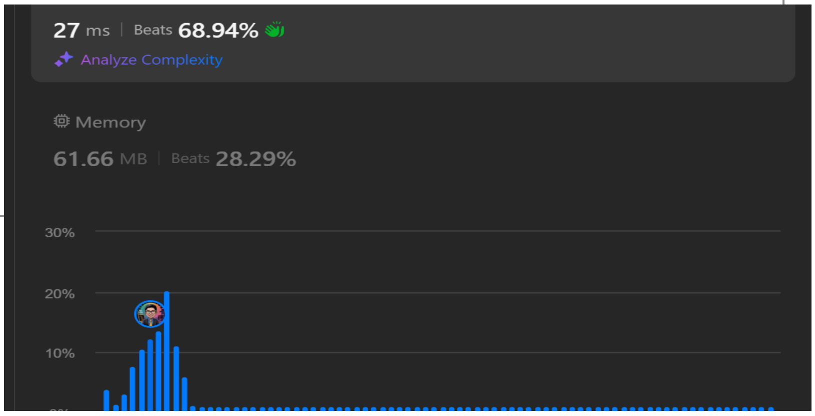
Output



```
2 24 45 66 75 90
```

215. Kth Largest Element in an Array
Medium O Topics Companies
Solved
Given an integer array mms and an integer k. retum the kth' largest element in the array.
Note that it is the kth largest element in the sorted order, not the kth distinct element.
Can you solve it without sorting?

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        priority_queue<int>maxHeap(nums.begin(), nums.end());
        for(int i=0;i<k-1;i++){
            maxHeap.pop();
        }
        return maxHeap.top();

    }
};

```



27 ms | Beats 68.94%
Analyze Complexity

Memory
61.66 MB | Beats 28.29%