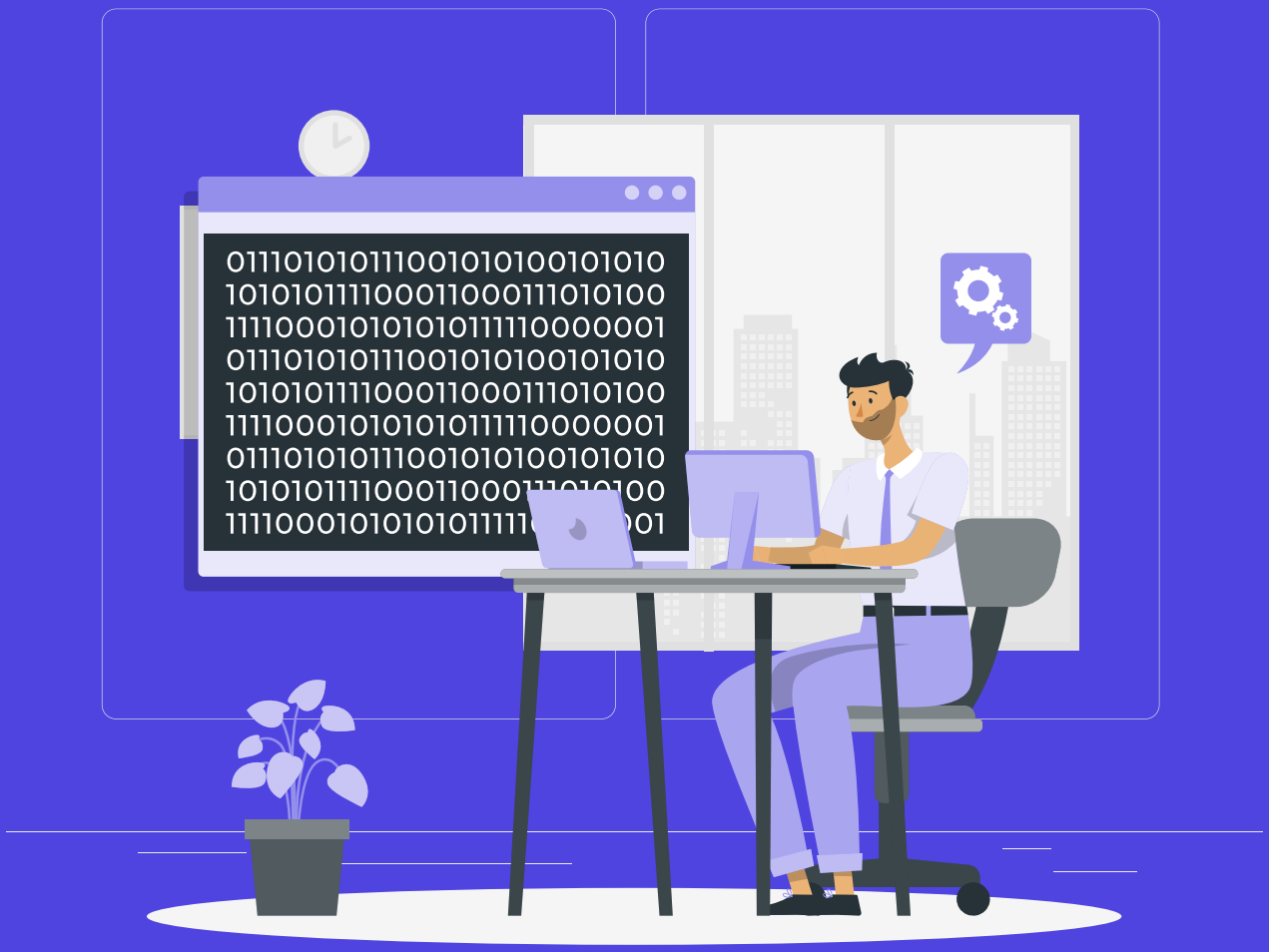
Class Plan

Advanced Sorting Algorithm - Count Sort, Bucket Sort -Part -2



Relevel
by Unacademy

Educator Introduction (5 mins)

- Greet Students
- Enquire students if they have revised previous class content

Concept to be covered in class (2 mins)

- Introduction to Advance Sorting
- Radix Sort
  - Definition
  - Algorithm
  - Code
  - Properties
- Count Sort on Negative Numbers
- Lexicographical Sorting

Advanced Sorting Algorithms (15 mins)

A technique to interpret data in some organized manner with the help of a comparator is known as sorting. Here comparator means a function that defines the order in which data should be present. For example: Ascending or Descending.

In our previous lecture, we have discussed many sorting algorithms and their complexities. For example, Insertion Sort, Selection Sort takes O(n^2). Also, we have optimized it further using Merge Sort, Quick Sort, which takes O(nlogn).

But sometimes, given some favorable conditions, we can improve the complexities to just linear time O(n) or O(n + k) means only one iteration is enough to sort the data.

Advanced sorting algorithms don't work on a comparison basis like other sorting algorithms. Here we sort with the help of some properties of the data given.

This lecture will discuss some advanced sorting algorithms and their complexities. Also, when we can use it and when it is impossible to use it.

Radix Sort

DEFINITION (10 mins)

Radix sort is a technique of sorting the data based on the digits. Here we sort the array with the least significant digit then move towards the most significant digit. It is also a non-comparison-based sorting algorithm.

Radix sort is again an integer-based sorting algorithm that works with the help of radix.

Here radix is meant by the base by which we will divide the array, like by base 10 or base 2.

Counting sort is used as a subroutine to sort the data in every transaction from least significant to the most significant digit.

ALGORITHM (30 mins)

Let's go through the implementation to understand better how radix sort works.

1.Given an unsorted array.
Example: givenArray = [129, 431, 234, 653, 232, 824, 2, 921, 54]

2.You have to first decide the base(the radix).
Example: base = 10

3.Now find the maximum element among them.
Example: maxElement = 921

4.So, our least to most significant value will iterate till 100's decimal place.
To calculate the decimal place, always multiply it by 10 in each pass.

5.Modify the counting sort to work with radix and decimal places.
a.Pass the decimal places with the array.

b.Wherever you are using the frequency array just use frequency[Math.floor((givenArray[i] / decimalPlace) %10)].
Example: Let's get 10's decimal of 129.
129 / 10 = 12.9 => 12.9 %10 = 2.

Let's get 100's decimal place of 921.
921 / 100 = 9.21 => 9.21 % 10 = 9.

c.The rest is the same as we discussed earlier.

Now, pass the sorting of the array using count sort based on the decimal place.

Pictorial view of radix sort:

[129, 431, 234, 653, 232, 824, 2, 921, 54]

129		431		002		002
431		921		921		054
234		232		824		129
653		002		129		232
232	=>	653	=>	431	=>	234
824	1's place	234	10's place	232	100's place	431
002		824		234		653
921		054		653		824
054		129		054		921

Hence, we get our sorted array.

An example of how count sort works from 1's place only will be helpful.

Value = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Frequency = [2, 2, 1, 3, 0, 0, 0, 0, 1]
So, output array will be
Result = [431, 921, 232, 002, 653, 234, 824, 054, 129]

Time Complexity:
Here it is not a direct calculation.

Let's take a list of n integers that are in base b (the radix).
So, the maximum d digits where d = floor(logb(maxElement) +1).
Example: 234 is in base 10 and 11101010 in base 2 are the same. But here decimals d may change.

So, with the help of base, you can control the complexity.

Radix sort works with any base since the running time of the algorithm, O(d(n+b)).

So, it depends on both size of data as well as the base. So, changing the base will also help you.

CODE (30 mins)

Let's check the code now to understand better:

```
const countSort = (givenArray, decimal) => {
   let frequency = new Array(10).fill(0);

   for (let i = 0; i < givenArray.length; i++)
       frequency[Math.floor((givenArray[i] / decimal) % 10)]++;
   console.log("Frequency");
    console.log(frequency);
    console.log();
    for (let i = 1; i < 10; i++)
       frequency[i] += frequency[i - 1];
    console.log("Frequency Sum");
    console.log(frequency);
    console.log();
   resultArray = new Array(givenArray.length).fill(0);
    for (let i = givenArray.length - 1; i >= 0; i--) {
       resultArray[frequency[Math.floor((givenArray[i] / decimal) %
10)] - 1] = givenArray[i];
       frequency[Math.floor((givenArray[i] / decimal) % 10)]--;
   }

    for (let i = 0; i < givenArray.length; i++)
       givenArray[i] = resultArray[i];

   console.log("sort with " + decimal + "'s decimal place.")
   console.log(givenArray);
}

const radixsort = (givenArray) => {
   let maxElement = Math.max(...givenArray);
   for (let decimal = 1; Math.floor(maxElement / decimal) > 0; decimal
*= 10){
       console.log("calling countSort with " + decimal + "'s decimal
place.")
       countSort(givenArray, decimal);
   }
}
let givenArray = [129, 431, 234, 653, 232, 824, 2, 921, 54];

radixsort(givenArray);

console.log("Result");
console.log(givenArray);
```

Output:

```
calling countSort with 1's decimal place.
Frequency
[
  0, 2, 2, 1, 3,
  0, 0, 0, 0, 1
]

Frequency Sum
[
  0, 2, 4, 5, 8,
  8, 8, 8, 8, 9
]

sort with 1's decimal place.
[
  431, 921, 232,  2,
  653, 234, 824, 54,
  129
]
calling countSort with 10's decimal place.
Frequency
[
  1, 0, 3, 3, 0,
  2, 0, 0, 0, 0
]

Frequency Sum
[
  1, 1, 4, 7, 7,
  9, 9, 9, 9, 9
]

sort with 10's decimal place.
[
    2, 921, 824, 129,
  431, 232, 234, 653,
   54
]
calling countSort with 100's decimal place.
Frequency
[
  2, 1, 2, 0, 1,
  0, 1, 0, 1, 1
]

Frequency Sum
[
  2, 3, 5, 5, 6,
  6, 7, 7, 8, 9
]

sort with 100's decimal place.
[
    2,  54, 129, 232,
  234, 431, 653, 824,
  921
]
Result
[
    2,  54, 129, 232,
  234, 431, 653, 824,
  921
]
```

Properties of Radix Sort (15 mins)

1. Time Complexity: O(d * (n+b)) where d is the maximum number of digits, n is the number of elements, and b is the base.
2. Space Complexity: O(n+k) same as counting sort.
3. Can be used for large datasets easily.
4. It will perform well till d is small means for less number of decimal places.
5. Used for suffix array construction in various algorithms like Manber's algorithm and DC3 algorithm.
6. It takes more space than any other sorting algorithm.
7. If you want to know how much decimal it will perform best is till we have log(n) base 2 decimal places for all elements. It will work best and even better than quicksort.

Count Sort on Negative Numbers (30 mins)

Problem Statement - Given an array having positive and negative numbers both. You need to sort it using Count Sort.

Input -> [-15, -10, 0, -13, 18, 5, -11, 10]
Output -> [-15, -13, -11, -10, 0, 5,10, 18]

Approach - In Counting sort, we use a specific range of given elements and then define a count array for those elements. In this case, since numbers are negative as well, we will find the minimum element and assign zero index to it.

Steps -

1. Create count array to store frequency of all elements
2. Keep minimum element index to be zero and so on
3. Modify count array in such a way that each element will have sum of previous frequencies
4. Rotate the array clockwise
5. Print the output

Code Link - https://jsfiddle.net/otzdkab7/

```
function countSort(arr)
{
var max = Math.max.apply(Math, arr);
var min = Math.min.apply(Math, arr);

var range = max - min + 1;
var count = Array.from({length: range}, (_, i) => 0);
var output = Array.from({length: arr.length}, (_, i) => 0);
for (i = 0; i < arr.length; i++) {
  count[arr[i] - min]++;
}

for (i = 1; i < count.length; i++) {
  count[i] += count[i - 1];
}

for (i = arr.length - 1; i >= 0; i--) {
  output[count[arr[i] - min] - 1] = arr[i];
  count[arr[i] - min]--;
}

for (i = 0; i < arr.length; i++) {
  arr[i] = output[i];
}
}

function print(arr)
{
  for (i = 0; i < arr.length; i++)
  {
    console.log(arr[i] + " ");
  }
}


var arr = [ -15, -10, 0, -13, 18, 5, -11, 10 ];
countSort(arr);
print(arr);
```

Lexicographical Sort (30 mins)

Problem Statement - Given a string. You need to sort it lexicographically

Input -> “unacademy”
Output -> “aacdemnuy”

Approach - We can use counting sort to sort the string.
Steps -

1. Create count array to store frequency of all characters
2. Modify count array in such a way that each element will have sum of previous frequencies
3. Rotate the array clockwise
4. Print the output

Code Link - https://jsfiddle.net/kfm48wxa/

```
function sortString(arr)
{
  var n = arr.length;

  var output = Array.from({length: n}, (_, i) => 0);

  var count = Array.from({length: 256}, (_, i) => 0);


  for (var i = 0; i < n; ++i)
    ++count[arr[i].charCodeAt(0)];

  for (var i = 1; i <= 255; ++i)
    count[i] += count[i - 1];


  for (var i = n - 1; i >= 0; i--) {
    output[count[arr[i].charCodeAt(0)] - 1] = arr[i];
    --count[arr[i].charCodeAt(0)];
  }


  for (var i = 0; i < n; ++i)
    arr[i] = output[i];
  return arr;
}


  var str = "unacademy";

  var arr = Array.from(str);

  arr = sortString(arr);
  document.write("Sorted string is ");
  for (var i = 0; i < arr.length; ++i)
    document.write(arr[i]);
```

MCQ

1) LSD radix sorting more efficient than other comparison sort algorithms?
a) true
b) False [Correct Answer]

2) What is the worst time complexity of LSD radix sort?
a) O(log n)
b) O(wn) [Correct Answer]
c) O(n log n)
d) None

3) What is the space complexity of LSD radix sort?
a) O(n+2^d) [Correct Answer]
b) O(nlogn)
c) O(logn)
d) None

4) Which are examples of comparison sort?
a) Bucket Sort
b) Counting Sort
c) Radix Sort
d) All of the above [Correct Answer]

5) Can we sort negative arrays using Count Sort?
a) True [Correct Answer]
b) false

Assignment Problem:

1) Implement MSD Radix Sort using LinkedList. MSD Radix Sort sort given input based on a most significant digit or index.

2) Write a Radix sort for negative numbers as well.