All sorting algorithims

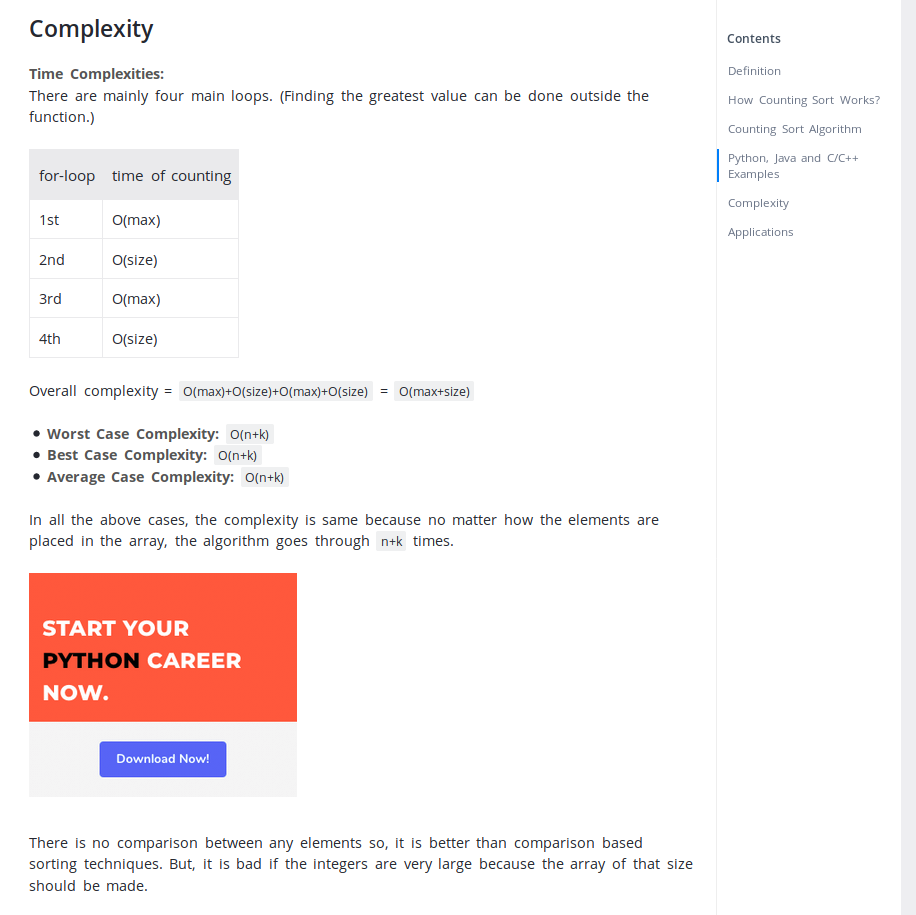
* // Counting sort in C++ programming
* #include <iostream>
* using namespace std;
* void countSort(int array[], int size)
* {
* // The size of count must be at least the (max+1) but
* // we cannot assign declare it as int count(max+1) in C++ as
* // it does not support dynamic memory allocation.
* // So, its size is provided statically.
* int output[10];
* int count[10];
* int max = array[0];
* for (int i = 1; i < size; i++)
* {
* if (array[i] > max)
* max = array[i];
* }
* for (int i = 0; i <= max; ++i)
* {
* count[i] = 0;
* }
* for (int i = 0; i < size; i++)
* {
* count[array[i]]++;
* }
* for (int i = 1; i <= max; i++)
* {
* count[i] += count[i - 1];
* }
* for (int i = size - 1; i >= 0; i--)
* {
* output[count[array[i]] - 1] = array[i];
* count[array[i]]--;
* }
* for (int i = 0; i < size; i++)
* {
* array[i] = output[i];
* }
* }
* void printArray(int array[], int size)
* {
* for (int i = 0; i < size; i++)
* cout << array[i] << " ";
* cout << endl;
* }
* int main()
* {
* int array[] = {4, 2, 2, 8, 3, 3, 1};
* int n = sizeof(array) / sizeof(array[0]);
* countSort(array, n);
* printArray(array, n);
* }

Space Complexity:

The space complexity of Counting Sort is O(max). Larger the range of elements, larger is the space complexity.

## Counting Sort Applications

Counting sort is used when:

* the are smaller integers of multiple counts.
* linear complexity is the need.

# Radix Sort Algorithm

In this tutorial, you will learn how radix sort works. Also, you will find working examples of radix sort in C, C++, Java and Python.

Radix sort is a sorting technique that sorts the elements by first grouping the individual digits of same place value. Then, sort the elements according to their increasing/decreasing order.

Suppose, we have an array of 8 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.

How Radix Sort Works?

1. Find the largest element in the array, i.e. max. Let X be the number of digits in max. X is calculated because we have to go through all the significant places of all elements.  
     
   In this array [121, 432, 564, 23, 1, 45, 788], we have the largest number 788. It has 3 digits. Therefore, the loop should go up to hundreds place (3 times).
2. Now, go through each significant place one by one.  
     
   Use any stable sorting technique to sort the digits at each significant place. We have used counting sort for this.  
     
   Sort the elements based on the unit place digits (X=0).

// Radix Sort in C++ Programming

#include <iostream>

using namespace std;

int getMax(int array[], int n)

{

int max = array[0];

for (int i = 1; i < n; i++)

if (array[i] > max)

max = array[i];

return max;

}

void countingSort(int array[], int size, int place)

{

const int max = 10;

int output[size];

int count[max];

for (int i = 0; i < max; ++i)

count[i] = 0;

for (int i = 0; i < size; i++)

count[(array[i] / place) % 10]++;

for (int i = 1; i < max; i++)

count[i] += count[i - 1];

for (int i = size - 1; i >= 0; i--)

{

output[count[(array[i] / place) % 10] - 1] = array[i];

count[(array[i] / place) % 10]--;

}

for (int i = 0; i < size; i++)

array[i] = output[i];

}

void radixsort(int array[], int size)

{

int max = getMax(array, size);

for (int place = 1; max / place > 0; place \*= 10)

countingSort(array, size, place);

}

void printArray(int array[], int size)

{

int i;

for (i = 0; i < size; i++)

cout << array[i] << " ";

cout << endl;

}

int main()

{

int array[] = {121, 432, 564, 23, 1, 45, 788};

int n = sizeof(array) / sizeof(array[0]);

radixsort(array, n);

printArray(array, n);

}

## Complexity

Since radix sort is a non-comparative algorithm, it has advantages over comparative sorting algorithms.

For the radix sort that uses counting sort as an intermediate stable sort, the time complexity is O(d(n+k)).

Here, d is the number cycle and O(n+k) is the time complexity of counting sort.

Thus, radix sort has linear time complexity which is better than O(nlog n) of comparative sorting algorithms.

If we take very large digit numbers or the number of other bases like 32-bit and 64-bit numbers then it can perform in linear time however the intermediate sort takes large space.

This makes radix sort space inefficient. This is the reason why this sort is not used in software libraries.

## Radix Sort Applications

Radix sort is implemented in

* DC3 algorithm (Kärkkäinen-Sanders-Burkhardt) while making suffix array.
* places where there are numbers in large range.