Some New Innovative Sorting Algorithms

George Michael

Abstract

In this research paper, some new innovative sorting algorithms are to be introduced as well as some development and enhancement for existing algorithms. for each algorithm, an evaluation is to be created. The main focus is to create efficient algorithms as well as developing new sorting algorithms that are designed to efficiently work under some conditions for specific problems.

1 Introduction

The paper has two types of categories, firstly, the new sorting algorithms, secondly, the development of some existing algorithms.

2 Innovative sorting Algorithms

2.1 NEMM Sorting Algorithm

The "Next Expected Minimum Maximum" sorting algorithm, lets consider having probably not sorted list of numbers L with length N lets denote any element in the list to be a_i where $a_i \in L$, lets define a_{min} to be the minimum element $in\ L$ and a_{max} to be the maximum element $in\ L$, a matter of fact, any element value $a_i\ in\ L$ will satisfy the following constrain $a_{min} \leq a_i \leq a_{max}$ by using a class having two attributes one for the value of the element i.e a_i and another attribute for the number of occurrences of a_i in L it is easy to keep track of the next expected element in the interval from both sides i.e moving in the positive direction looking for the next increasing element and moving backwards starting from the last element looking for the previous decreasing element if its occurrence is > 0, for illustration, lets consider having two arrays A_1 and A_2 having the same size, the first one will hold the class objects and the second one is the final sorted array that will holds only values starting from [a_0, a_0,...a_i, a_{i+1},...a_{N-1}] and lets consider having two pointers P_1 and P_2 will start at a_{min} and the other at a_{max}, for P_1 if the current Occurrence of $a_i > 0$ then fill A_2 with the value of a_i with the number of occurrence one after another, for P_2 the same will happen except that the direction of the pointer will be opposite to P_1 until one of the following conditions is satisfied:

-if N is an even number that is it can be represented by the formula (2*k) where k is some integer number, the length of the sorted array A_2 is equal to N then terminate i.e both pointers have no any other next element.

-both P_1 and P_2 are equal or point to same element i.e N is odd and can be represented by the formula (2*k+1)where k is some integer number.

-N is equal to 1 which is the base case then the array is sorted already.

Please note that this NEMM algorithm is different than the ordinary min-max algorithm, it does not look for the actual min and max elements it does look for the right next expected min and the right next expected max and if they already exist in the array A_1 then put its occurrences one after another, lets consider the following sample for how the "NEMM" algorithm will act for a list L holding numbers [5,5,3,7,9,2,9,4,1,3,6,8,8]:

By iterating over L with O(N) we have $a_{max} = 9$ and $a_{min} = 3$ then we create objects and add them to a temporary List T of size (a_{max} - a_{min})+1, starting by object = a_{min} to a_{max} inclusive. By using a class, data can be represented as a table as follow:

value: frequency

1 : 1
2 : 1
3 : 2
4 : 1

5	:	2
6	:	1
7	:	1
8	:	2
9	:	2

we then create the list that will hold the sorted numbers, let's denote that to be SL then,

we start $P_1 = 0$, P_2=Size(SL)-1

```
for i in Size(T):
//fill current number with its value in SL Frequency times.
for freq in number.frequency:
fill LS with number.valueAt($P_1$), frequency times
$P_1$ +=1
for freq in number.frequency:
fill LS with number.valueAt($P_2$), frequency times
$P_1$ -=1
if(Size(LS)==Size(L)):
break
if( $P_1$== $P_2$):
print last element
break
```

Over All complexity for the code O((((a_{max} - a_{min})+1)/2)*max(Frequency))

3 Optimized Algorithms

3.1 MP-NEMM Sorting Algorithm

"Multiple Pointer Next Expected Min Max" sorting algorithm is just a modification on the NEMM sorting algorithm where, instead of having two pointer, multiple number of pointers are created randomly in a set "no element two pointers points to the same element in the list T" lets denote the list as RP, each pointer will take a random direction weather left or right (0,1), using the same algorithm as NEMM but the only difference is whenever any two pointers P_i and P_j where i !=j and $a_{min} \leq$ i,j $\leq a_{max}$ point to the same element a_k where k lies between the interval from 0 to Size(T) inclusive.