International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6S4, November 2019

A Novel Sorting Method for Real and Integer Numbers: An Extension of Counting Sort

Kshitij Kala, Sandeep Kumar Budhani, Rajendra Singh Bisht, Dhanuli Kokil Bisht, Kuljinder Singh Bumrah

Abstract: ***Sorting is an essential conceptin the study of data structures. There are many sorting algorithms that can sort elements in a given array or list. Counting sort is a sorting algorithm that has the best time complexity. However, the counting sort algorithm only works for positive integers. In this paper, an extension of the counting sort algorithm is proposed that can sort real numbers and integers (both positive and negative).***

Index Terms: Counting Sort, Sorting, Algorithm.

I. INTRODUCTION

In the field of computer science, **sorting**denotes the process of arranging items in a sequence ordered by a given criterion. A sorting procedure is aprocedure that places elements of a list or array in a certain order. Acapable sorting algorithm helps optimize the competence of other procedures, such as search and merge algorithms, which require input data in the form of sorted lists. In fact, **search preprocessing** is perhaps the single most important application of sorting algorithms. Sorting is also useful for **canonicalizing** data (i.e converting it into a standard representation) and for producing human-readable output. Sorting algorithms function as a basic building block for a wide variety of applications, including problems of closest pair finding, element uniqueness, frequency distribution and selection.Commonly used sorting algorithms include **quick sort, counting sort, heap sort, bubble sort, selection sort, insertion sort, merge sort** and **radix sort.**

The efficiency of an algorithm is measured as a function of the computational cost it exacts on the system, in relations of its space and time complexity. **Time complexity** is a measure of the time it takes for an expression to be executed, as a function of the input size. It is generally expressed in **big O notation (O(n)),** a mathematical notation that is used to describe the limiting manners of a function when its argument tends towards a given value or infinity (asymptotic complexity). This representationdescribes functions according to their **growth rate**, or the rate at whichthe time taken to execute a function increases with increase in input size.

Space complexity is a measure of the volume of memory space mandatory to solve an instance of a problem, as a function of input size.

Manuscript published on 30 November 2019.
***** Correspondence Author (s)
Kshitij Kala, Student, GEHU, Bhimtal Campus, Uttarakhand, India.
Sandeep Kumar Budhani (Corresponding Author), Associate Professor, GEHU, Bhimtal Campus, Uttarakhand, India. E-mail: sandeepbudhani13@gmail.com
Rajendra Singh Bisht, Research Scholar, GEU, Dehradun, India.
Dhanuli Kokil Bisht, Student, GEHU, Bhimtal Campus, Uttarakhand, India.
Kuljinder Singh Bumrah, Assistant Professor, GEHU, Dehradun, India.

It is also often expressed asymptotically in big 0 notation O(n), where n is the input size in bytes.

Given below is a comparison of the space and time complexity of various popular sorting algorithms:

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

Fig. 1.

In this paper, we will focus on a discussion of counting sort, a unique sorting algorithm. Unlike other sorting algorithms, it does not compare elements to arrange them in a particular order. We will test the working of the proposed extension of the counting sort algorithm with large data sets, and compare its space and time complexity with that of other popular sorting algorithms.

II. WORKING OF COUNTING SORT ALGORITHM

Let us assume that an array A of size N is to be sorted.

Algorithm: CountingSort(A, N , SortedA, Count, S)
// A is the array to be sorted
// The result is stored in SortedA
// N is the size of arrays A and SortedA
// Count is an array whose size is equal to the value of maximum element of A
// S is the maximum element of array A
// Each element of Count is initialized to 0

Step 1: Set i← 0
Step 2: Repeat this step while i< N
  a. Set Count[A[i]] ← Count[A[i]] + 1
  b. Set i←i + 1
  End of Loop
Step 3: Set i← 0
Step 4: Set j ← 0
Step 5: Repeat this step while i< S:
  a. Repeat while Count[A[i]] > 0:
    a. Set SortedA[j] ← A[i]
    b. Set j ←j + 1
    c. Set Count[A[i]] ← Count[A[i]] – 1
  End of Loop

b. Set i←i + 1
End of Loop
Step 6: Print SortedA

Lets take an example.
Suppose the input array is [4, 8, 4, 2, 9, 9, 6, 2, 9].

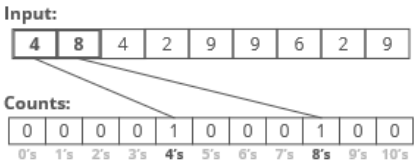

Fig. 2.

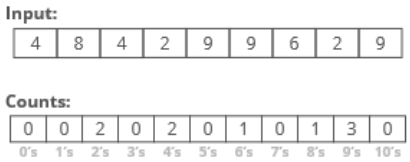

Fig. 3.

The above figure shows how the operations in counting sort are done.

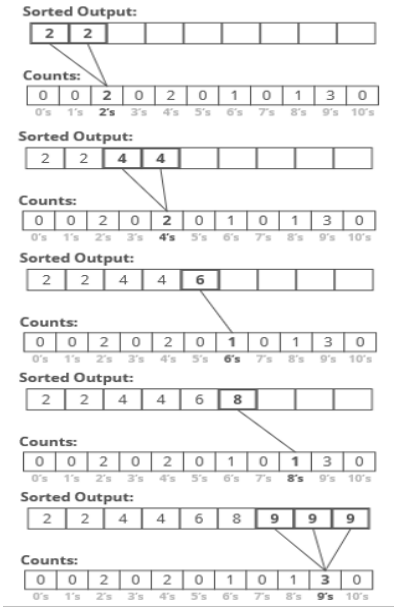

Fig. 4.

III. LIMITATIONS OF COUNTING SORT

1. The counting sort algorithm has high space complexity.
2. The counting sort algorithm can only operate on positive integers.

In this research paper, we have worked on removing the second limitation of the counting sort algorithm.

IV. NEGATIVE INDEX CONCEPT

Python is a very popular programming language, in which there is a concept of negative indices. For example, let us say an array A = [4, 5, 6, 1, 2]. In Python, there is no such thing as an array; Python uses lists. This means that A =[4, 5, 6, 1, 2] is a list in Python. If we try to access A[2], we will get 6 as output. If we try to access A[-3], this will also give us 6 as the output.

This concept is illustrated in the table below:

Positive Index	0	1	2	3	4
Values	4	5	6	1	2
Negative Index	-5	-4	-3	-2	-1

Fig. 5.

From the above diagram, it is clear that index 0 is equivalent to index (-5). Index 4 is equivalent to index (-1).

Positive Index	**Negative Index**	**Value**
A[0]	A[-5]	4
A[1]	A[-4]	5
A[2]	A[-3]	6
A[3]	A[-2]	1
A[4]	A[-1]	2

Fig. 6.

So we can access an array (a list in Python) with negative indices as well as positive, which can solve the problem of sorting negative integers using the counting sort algorithm.

V. EXTENSION OF COUNTING SORT ALGORITHM FOR NEGATIVE INTEGERS

Algorithm: CountingSort(A, M, N, SortedA, Count, S, T)
// A is the array to be sorted
// M is the size of array A
// The result is stored in SortedA
// Count is an array of size N with each of its elements initialized to 0
// S is the maximum value of any positive integer in array A
// T is the minimum value of any negative integer in array A
// N is the size of arrays A and SortedA,
N = S + ((-1) * T)
// Each element of Count is initialized to 0

Step 1: Set i← 0
Step 2: Repeat this step while i< M
a. Set Count[A[i]] ← Count[A[i]] + 1
b. Set i←i + 1
End of Loop
Step 3: Set i← T
Step 4: Set j ← 0
Step 5: Repeat this step while i<S:
a. Repeat while Count[A[i]] > 0:
a. Set SortedA[j] ← A[i]
b. Set j ←j + 1
c. Set Count[A[i]] ← Count[A[i]] – 1

Retrieval Number: F10031186S419/2019©BEIESP
DOI:10.35940/ijeat.F1003.1186S419
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

End of Loop
b. Set i←i + 1
End of Loop
Step 6: Print SortedA

VI. CONVERSION OF REAL NUMBERS TO INTEGERS

Thus far, we have modified the counting sort algorithm to operate on negative integers as well as positive. Now we shall discuss how to adapt the algorithm for floating point numbers.

The counting sort algorithm uses a concept of memory addressing (array indices) to store the values of elements present in the array to be sorted.Memory address or array index cannot be represented by a floating point value. Therefore, if we want to sort floating point numbers using the counting sort algorithm, we need to convert them to integers.

Now we must address the question of how we can achieve this conversion.

We consider the number 6.78. If we multiply this number by 100 then it becomes 678.

Convertor: In the above example 100 acts as a convertor. Taking the example 56.2, if we multiply it by 10 then it becomes 562. In this case 10 is the convertor. Thus, we see that the value of the convertor depends on how many digits are present after the decimal point.

For example, to convert 10.222 into an integer, the convertor must be 1000.

If the value of the convertor is 10000, it can convert only those floating point numbers which have number of digits after the decimal point less than or equal to 4.

\Here is a table which shows how floating values are converted to integers if the convertor is 10000.

Floating Point Value	Convertor Value	Integer Value
5.4345	10000	54345
4.6	10000	46000
3.21	10000	32100
0.001	10000	10
0.4315	10000	4315
10.001	10000	100010

Fig. 7.

VII. PROPOSED ALGORITHM

Before beginning the sorting process, we need to convert floating point numbers to integers. Given below is the algorithm of the function that will perform the required conversion.

Algorithm: toIntegers(A, N, convertor)
// A is the input array
// N is the size of A
Step 1: Set i← 0
Step 2: Repeat while i< N:
a. Set A[i] ← A[i] * convertor
b. Set i←i + 1
End of Loop
Step 3: Return A

After converting, we need to find the maximum size of the required count array.

Algorithm: size Of Count Array(limit, dec)
// limit is the maximum number of digits before the decimal point in any of the values to be sorted
// dec is the maximum number of digits after the decimal point
Step 1: Set h ← 1
Step 2: Set i← 0
Step 3: Set size ← 0
Step 3: Repeat while i< limit + dec
a. Set size ←size = (9 * h)
b. Set h ←h * 10
c. Set i←i + 1
End of Loop
Step 4: Set size ← (size * 2) + 1
Algorithm: ExtendedCountingSort(A, M, SortedA, limit, dec, convertor)
// limit is the maximum number of digits before decimal point
// dec is maximum number of digits after decimal point
// A is array to be sorted of size M
// Result will be stored in SortedA of size M
Step 1: Set s←sizeOfCountArray(limit, dec)
Step 2: Declare array count [ ] of size s
Step 3: Set A ←toInteger(A)
Step 5: Repeat this step while i< s
a. Set Count[A[i]] ← Count[A[i]] + 1
b. Set i←i + 1
End of Loop
Step 5: Set i← (-1) * (s/2)
Step 6: Repeat while i< s/2
a. Repeat while Count[A[i]] > 0:
a. Set SortedA[j] ← A[i]/convertor
b. Set j ←j + 1
c. Set Count[A[i]] ← Count[A[i]] – 1
End of Loop
b. Set i←i + 1
End of Loop
Step 7: Print SortedA

VIII. TIME AND SPACE COMPLEXITY

The time and space complexity of this suggestedprocedure depends on the range of numbers. Time complexity also be influenced by on the number of input. Here is a graph of time and space complexities. Time complexity of this procedure depends more on range rather than input size.

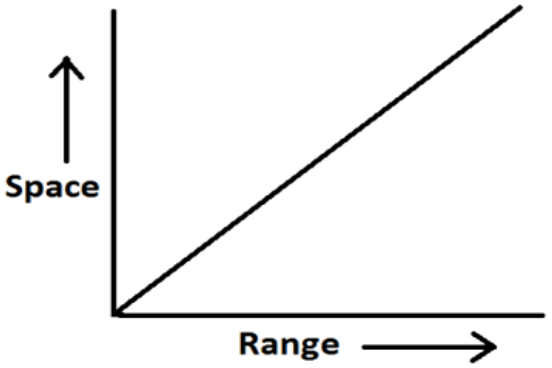

Fig. 8. Space and Range relation

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

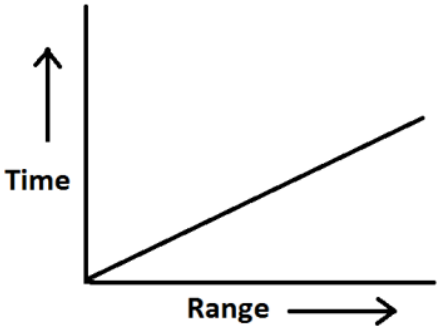

Fig. 9. Time and range relation

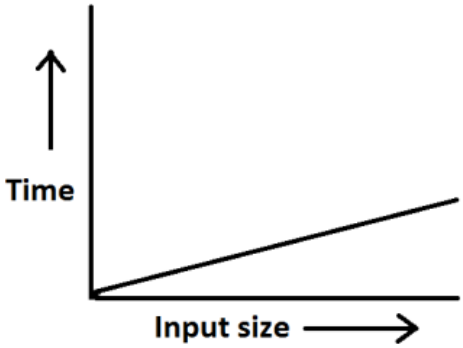

Fig. 10. Time and input size relation

IX. COMPARISON

This algorithm was tested with large data sets. The Fig. 11 shows the results of comparison when 1000000 elements were sorted using different algorithms.

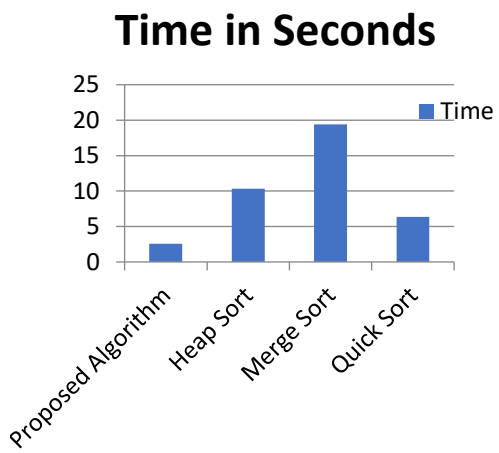

Fig. 11 Comparison of Algorithms on the basis of time

X. CONCLUSION

As we can see in the above figure, the proposed algorithm had the lowest time complexity of any of the algorithms tested (in this case, its execution time was 2.57 seconds, appreciably faster than the next fastest algorithm, quick sort, which took 6.34 seconds).

Negative integers and floating point numbers are successfully sorted using the proposedalgorithm. The speed of the algorithmis very high as compared to other popular algorithms, as seen above. This algorithm is also very efficient for large data sets where the range of numbers is known.

REFERENCES

1. J. Hammad, "A comparative study between various sorting algorithms," International Journal of Computer Science and Network Security (IJCSNS), vol. 15, no. 3, p. 11, 2015.
2. T. H. Cormen, Introduction to Algorithm, MIT Press, Cambridge, MA, USA, 2009.
3. K. S. Al - Kharabsheh, I. M. AlTurani, A. M. I. AlTurani, and N. I. Zanoon, "Review on sorting algorithms a comparative study," International Journal of Computer Science and Security(IJCSS), vol. 7, no. 3, pp. 120–126, 2013.
4. A. Jehad and M. Rami, "An enhancement of major sorting algorithms," International Arab Journal of Information Technology, vol. 7, no. 1, 2010
5. Hoare, C.A.R. "Algorithm 64: Quick sort". Comm. ACM 4, 7 (July 1961), 321.
6. Flores, I. "Analysis of Internal Computer Sorting". J.ACM 7, 4 (Oct. 1960), 389-409.
7. Y. Han "Deterministic Sorting In O(nlog Log n) Time And Linear Space", Proceedings of the thirty-fourth annual ACM symposium on Theory of computing, Montreal, Quebec, Canada, 2002, p.602-608.
8. MacLaren, M.D. "Internal Sorting By Radix Plus Sifting". J. ACM 13, 3 (July 1966), 404-- 411.
9. J. Larriba-Pey, D. Jim´enez-Gonz´alez,and J. Navarro. "An Analysis of Superscalar Sorting Algorithms on an R8000 processor". In International Conference of the Chilean Computer Science Society pages 125–134, November 1997.

AUTHOR'S PROFILE

Kshitij Kala received a Bachelor's degree in Computer Science and Application from Graphic Era Hill University, India in 2018. He is currently pursuing a Master's degree in Computer Applications from Graphic Era Hill University, India. His research interests include algorithms, cryptography and artificial intelligence.

Sandeep Kumar Budhanir received his Ph.D in Computer Science. He is Microsoft Certified Professional and Microsoft Certified Database Administrator also. He is having more than 15 year of teaching experience. His area of interest is data mining, machine learning and neural network.

Rajendra Singh Bisht is a research scholar in Department of Computer Science and Engg., GEU, Dehradun. His area of interest is MANET and Data Mining.

Dhanuli Kokil Bishtis currently puruing a Bachelor's degree in Computer Science and Engineering from Graphic Era Hill University, India. Her research interests include logic, philosophy and cryptography.

Kuljinder Singh Bumrah currently working as an Assistant Professor in CSE department, GEHU Dehradun. His area of intereset is Algorithms and Machine learning.