INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 5)
Available online at: www.ijariit.com

Adaptive approach for sorting the list: Top hill method

Chaudhary Ravi Singh
ravirbmi244@gmail.com
Rakshpal Bahadur College of Engineering and Technology,
Bareilly, Uttar Pradesh

ABSTRACT

The reason behind finding this method is to decrease the time of insertion. Sort by using the pivot method of quicksort, binary searching and doubly linked list. By using all these methods in a combination this method can sort real-time data and increase the size of the list with the worst case of O(n).

Keywords*— Insertion sort, Complexity, Algorithm, Top sort hill method*

1. INTRODUCTION

The key component behind solving computational problems is an algorithm, the algorithm is a well -defined procedure to do a certain task on the computer. They were on existence even before the existence of computers.

There are different kinds of problems are there for which different kinds of the algorithm were made through various methodologies and techniques, out of all these algorithms, let us talk about the class of algorithm which is based on sorting.The main aim of these type algorithm is to sort the given data in given order (less than greater than or alphabetical).

Sorting is considered a fundamental operation in computer science as it is used as an intermediate step in many programs. Due to this, the number of sorting algorithms are available and it is pretty large in number. Out of these available algorithms, which one is considered as the best one for a particular application further has its dependency on various other factors which include:

- The size of the sequence to be sorted.
- The extent up to which the given input sequence is already sorted.
- The type of storage devices to be used: main memory or disks.

Almost all the available sorting algorithms can be categorized into two categories based on their difficulty. The complexity of an algorithm and its relative effectiveness are directly correlated. A standardized notation that is Big O(n), is used to describe the complexity of an algorithm. In this notation, the O represents the complexity of the algorithm and n represents the size of the input data values.

Almost all the available sorting algorithms can be categorized into two categories based on their difficulty. The complexity of an algorithm and its relative effectiveness are directly correlated. A standardized notation that is Big O(n), is used to describe the complexity of an algorithm. In this notation, the O represents the complexity of the algorithm and n represents the size of the input data values.

Two categories of sort algorithms were classified according to the records whether stored in the main memory or secondary memory. One category is the internal sort which stores the records in the main memory. Another is the external sort which stores the records in the hard disk because of the records' large space occupation.

The basic idea of insertion algorithm is to divide the array into an unsorted and sorted region. Initially, the first item is the only member of the sorted region, and the remaining items make up the unsorted region. The idea then is to sequentially pick out each item of the unsorted region and place it in the proper position in the sorted region.

This algorithm provides several advantages such as it is easy to implement, can sort a list as it receives it, does not change the relative order of elements with equal keys and more efficient than any other sorting algorithms. But it has a problematic disadvantage that is it is much less efficient on large lists than more advanced algorithms such as quick sort, heap sort, or merge sort. With n-squared steps required for every n element to be sorted, the insertion sort does not deal well with a huge list. Therefore, the insertion sort is particularly useful only when sorting a list of few items.

In this paper, we present a new implementation of Insertion Sort using Doubly Linked List. A linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence. The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while an array has to be declared in the source code, before compiling and running the program. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal. And because of its top sort hill method, it is much faster in real-time data.

2. GENERAL INSERTION SORT

2.1 Algorithm

Every repetition of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list until no input elements remain. The choice of which element to remove from the input is arbitrary and can be made using almost any choice algorithm.

Sorting is typically done in-place. The resulting array after k iterations has the property where the first k + 1 entries are sorted. In each iteration the first remaining entry of the input is removed, inserted into the result at the correct position, thus extending the result.

Below is the pseudo-code for insertion sort:

```
1. for j ←1 to length(A)-1
2. key ← A[ j ] //insert A[j] is added in the sorted sequence
   A[1, ..  j-1]
3. i ← j - 1
4. while i >= 0 and A [ i ] > key
5. A[ i +1 ] ← A[ i ]
6. i ← i -1
7. A [i +1] ← key
```

2.2 Performance

The best case input is an array that is already sorted. In this case insertion sort has a linear running time that is O(n). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

The worst case input is an array sorted in reverse order. In this case, every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. For this case insertion sort has a quadratic running time that is O(n^2).

The average case is also quadratic, which makes insertion sort impractical for sorting large arrays. However, insertion sort is one of the fastest algorithms for sorting very small arrays, even faster than quicksort; indeed, good quicksort implementations use insertion sort for arrays smaller than a certain threshold, also when arising as subproblems; the exact threshold must be determined experimentally and depends on the machine, but is commonly around ten.

3. INSERTION SORT WITH TOP SORT HILL METHOD

3.1 Algorithm

This method works on a real-time basis. And due to the use of dynamic linked list its size is not fixed. During an insert, the process number is inserted as a node into the link list by breaking the links between the node and then reconnecting them.

Below is the pseudo-code for insertion sort with top sort hill method:

```
struct node
        {data
        leftlink
        rightlink
        }
node *top,*tmp,*aj
top=null
aj=create a new node
enter aj->data
aj->leftlink=null
aj->rightlink=null
 if top==null
   top=aj;
else
   {if top->data>tmp->data
     // the searching will be done on the left side of
      top and insert the 'aj' node in the right
     place of the list
     top=aj
   else //the searching will be done on the right side of
      top and insert the 'aj' node in the right
     place of the list
     top=aj
   }
```

3.2 Performance

The best case input in a linked list is when a node is to be entered just side to pivot right/left position. In this case, insertion sort time does not depend on the size of list O(1).

The worst case input in a linked list is when the pivot is one of the endpoints of the linked list and node is to be entered at opposite endpoint. Due to this reason it had to do a linear search from one point to another. In this case insertion sort has a linear running time that is O(n).

The average case is also quadratic, which makes insertion sort impractical for sorting large arrays. However, insertion sort is one of the fastest algorithms for sorting very small arrays, even faster than quicksort; indeed, good quicksort implementations use insertion sort for arrays smaller than a certain threshold, also when arising as subproblems; the exact threshold must be determined experimentally and depends on the machine, but is commonly around ten.

3.3 Comparison

Simple insertion sort (Array)

						temp
	5	4	1	2	3	
shifting1	4	5	1	2	3	
shifting2	4	4	5	2	3	1
copying ten	1	4	5	2	3	
shifting3	1	4	4	5	3	2
copying ten	1	2	4	5	3	
shifting 4	1	2	4	4	5	3
copying ten	1	2	3	4	5	

Insertion sort with Top sort hill method

Input			Top node		
			NULL		
5			5		
4			4	5	
1			1	4	5
2		1	2	4	5
3	1	2	3	4	5

4. CONCLUSION

We can see that new method required fewer steps in comparison to simple insertion sort. In simple insertion sort we have to maintain an unsorted list before sorting it but in Top sort hill method, we don't have to maintain a predefined sorted list. We can enter a new number whenever we want and sorting is done at the real time.

5. REFERENCES

[1] Pankaj Sareen. Comparison of Sorting Algorithms (On the Basis of Average-Case). Department of Computer Applications. International Journal of Advanced Research in Computer Science and Software Engineering, 2013. ISBN: 2277 128X

[2] Kenneth A. Berman and Jerome L. Paul. Algorithms: Sequential, Parallel, and Distributed. Computer Science and Engineering. Thomason course technology, 2002. ISBN: 0-534-42057-5.

[3] Khalid Suleiman Al-Kharabsheh et al. Review on Sorting Algorithms A Comparative Study. Computer Science. International Journal of Computer Science and Security (IJCSS), 2013.

[4] https://en.wikipedia.org/wiki/Seymour_Lipschutz

[5] Ramesh Chand Pandey, Comparison Of various sorting algorithms, Computer Science and Engineering 2008, ISBN 80632019.

[6] H. Deitel and P. Deitel, C++ How to Program, Prentice Hall, 2001, pp.150-170.

[7] M. Sipser, Introduction to the Theory of Computation, Thomson, 1996, pp.177-190.