Study & Comparative Analysis of Various Sorting Techniques

HOMA FIRDAUS
Research Guide: Govind Prasad Arya

Abstract:-Sorting refers to the storage of data in a sorted manner or arranging data in an increasing or decreasing fashion so as to make searching purpose easy and fast. Sorting comes into picture with the term Searching in short we can say that it improves the efficiency of searching and hence reduces the complexity of problem . There are so many things in our real life that we need to search, like a particular record in database, arranging roll numbers of students , a particular telephone number, any particular page in a book etc ,all this can be easily done if we sort them in a particular order like for searching a word in a dictionary we sort them in alphabetical order Sorting techniques mainly depends on two parameters. First parameter is the execution time of program, which means time taken for execution of program. Second is the space, which means space taken by the program

Index Terms— Sorting Techniques and sorting algorithms with examples

1 INTRODUCTION

What is sorting?

Arranging data in a particular format is called sorting. Sorting algorithm specifies that how a data is to be arranged ,it provides a way to arrange data. There are two broad categories of sorting methods:

With the help sorting data searching is optimized to a very high level . Sorting is also used to represent data in more readable formats. Some of the examples of sorting in real life scenarios are following.

Telephone Directory − Telephone directory keeps telephone no. of people sorted on their names so that names can be searched.

Dictionary − Dictionary keeps words in alphabetical order so that searching of any work becomes easy.

2. DIFFERENT TYPES OF SORTING TECHNIQUES

2.1 INTERNAL SORTING

Internal sorting takes place in the main memory, where we can take advantage of the random access nature of the main memory.This is done when small amount of data is to be sorted.

Internal (In Memory)

- Insertion Sort
- Selection sort
- Bubble Sort
- Quick Sort
- Heap Sort
- Shell Sort

2.2 EXTERNAL SORTING

External sorting is necessary when the number and size of objects are prohibited to be stored in the main memory and external storage is required during sorting.

External (Appropriate For Secondary Storage)

- Merge Sort
- Polyphase Sort

Apart from all the above mentioned sorting there are some other types of sorting which are discussed below in detail:

2.1.1. INSERTION SORT

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists(data) and mostly sorted lists and is not efficient for large data. It takes elements from the list one by one and inserts them in their correct position into a new sorted list. New list obtained by sorting and the remaining elements can share the array's space in array, but insertion sort is expensive as it requires shifting all following elements over by one. Shell sort is a variant of insertion sort that is more efficient for larger lists.

ALGORITHM FOR INSERTION SORT

Insertion sort iterates by taking one input element at a time and checking with the other elements and finally generates a sorted output lists. In each iteration insertion sort removes one element from the input data finds the location which belongs within the sorted list and inserts it there This process is repeated until we get the desired set of sorted output .

1. Sorting is typically done in-place, by iterating up the array and growing the sorted list behind it i.e on the left side usually. At each array-position, it checks the value
there against the largest value in the sorted list if larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted list, shifts all the larger values up to make a space, and inserts into that correct position.

We take the below array for example

- Now checking the adjacent array position that that whether the element is smaller or not if it smaller than it is shifted back in the array and the array becomes

- As 7 is not smaller than 6 therefore it will remain in the same position and further checking is done within the array and 1 is found smaller ,and the array becomes

- The final resulted array after sorting and going through the same procedure is

2.1.2 SELECTION SORT

It is a comparison sort which is inefficient on large lists(data), and generally performs worse than the similar insertion sort. Selection sort uses fewer writes, and thus is used when write performance is a limiting factor.Selection sort is known for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

The algorithm finds the minimum value, swaps it with the value in the first position, and repeats these steps for the remainder of the list.It does no more than *n* swaps, and thus is useful where swapping is very expensive.

ALGORITHM FOR SELECTION SORT

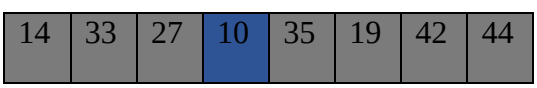
1 The list is divided into two parts, sorted part is at left end and unsorted part is at right end.

2. Initially sorted part is empty and unsorted part is entire list.

3. Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues by moving unsorted element to the right.

4. This algorithm is not suitable for large data sets.

We take the below depicted array for our example.



5 For the first position in the sorted list, the whole list is scanned . The first position where 14 is stored presently, we search the whole list and find that

10 is the lowest value in the whole list. So we replace 14 with 10. After one iteration 10, appears in the first position of sorted list.

10	33	27	14	35	19	42	44

For the second position, where 33 is placed, we start scanning the rest of the list and find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two data with leat values are positioned at the the beginning in the sorted manner.

10	14	27	33	35	19	42	44

6 For the first position in the sorted list, the whole list is scanned . The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value in the whole list. So we replace 14 with 10. After one iteration 10, appears in the first position of sorted list.

2.1.3 BUBBLE SORT

This algorithm works by comparing each item in the list with the item next to it, and swapping them if required. In other words, the largest element has bubbled to the top of the array and the largest element is swapped with the smallest element by comparing each element with the top bubbled element . The algorithm repeats this process until the bubbled element is placed at its position .

Example. Here is one step of the algorithm. The largest element - 20- is bubbled to the top:

20,5, 4,15,18
5,20,4,15,18
5,4,20,15,18
5,4,15,20,18
5,4,15,20,18

After 20, 5is the bubbled element .The whole process repeats for each element until finally sorted list of elements is obtained.

2.1.4 QUICK SORT

Quick sort is a highly efficient sorting algorithm which is quite efficient for large sized data and is based on partitioning in which the array is portioned in two parts one contains smaller value than pivot element and other contains larger value than pivot element .

ALGORITHM FOR QUICK SORT

Step 1. Choosing the Pivot Element a. Normally we choose the first, last or the middle element as pivot. This can harm us badly as the pivot can

the smallest or the largest element, thus leaving one of the partitions empty.
b. We should choose the Median of the first, last and middle elements. If there are N elements, then the ceiling of N/2 is taken as the pivot element.
Example:

7	3	25	6	15	17	1	2	20	10

First Element: 7
Middle Element: 15
Last Element: 10
Therefore the median on [7,15,10] is 7 which is taken as the pivot element.
Step 2. Partitioning
a. First thing is to get the pivot out of the way and swapping it with the last number.
Example: (shown using the above array elements)

7	3	2	6	15	17

b. Now we want the elements greater than pivot to be on the right side of it and similarly the elements less than pivot to be on the left side of it.
For this we define 2 pointers, namely i and j. i being at the first index and j being and the last index of the array.

7 While i is less than j . i is incremented until we find an element greater than pivot.

8 Similarly, while j is greater j is decremented until we find an element less than pivot.

9 After both i and j stop we swap the elements at the indexes of i and j respectively.

c. Pivot is restored

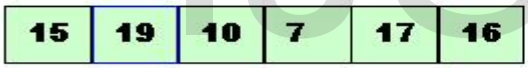
After performing the above steps we will get this as our output:
[3, 2, 6, 1] [7] [15.10,25, 18, 17]

Step 3. Recursively Sort the left and right part of the pivot

2.16. HEAP SORT

Heap is a way to implement a priority queue .Heaps have same complexity as a balanced search tree but:

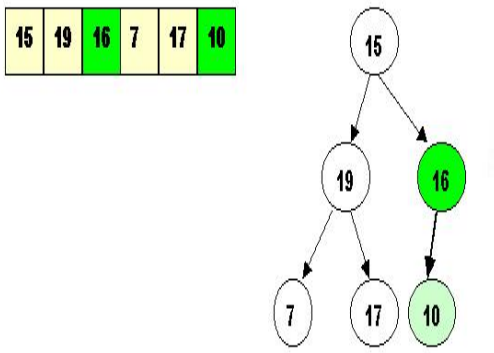
- they can easily be kept in an array
- they are much simpler than a balanced search tree
- they are cheaper to run in practice



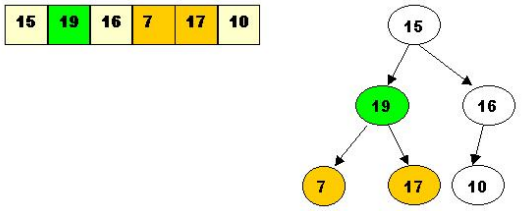
Start with the rightmost node at height 1 - the node at position 3 = Size/2.

It has one greater child.

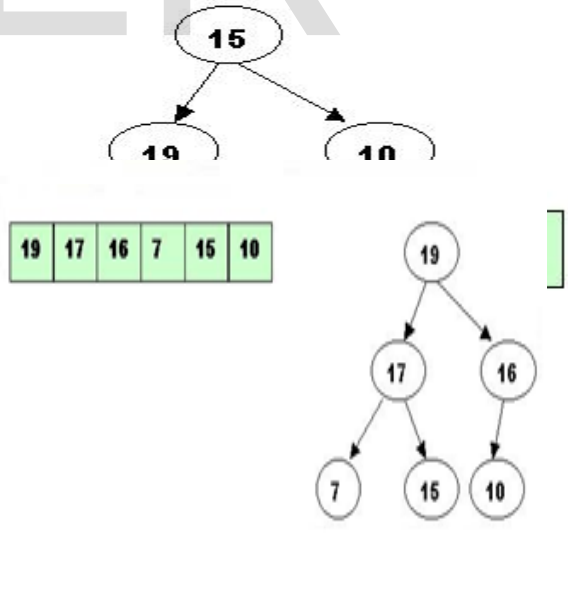
After processing array[3] the situation is



Next comes array[2]. Its children are smaller, so no rearranging is needed.



The whole process runs and sorting is done as mentioned in the above way



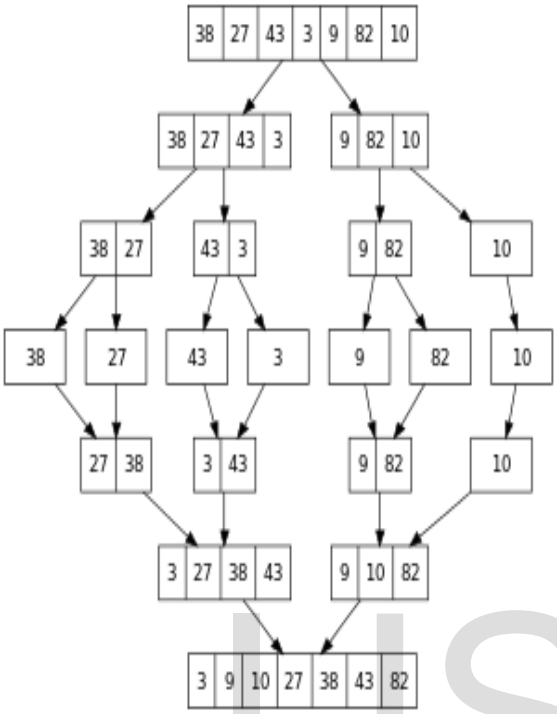
Finally sorted heap tree

2.2.1 MERGE SORT

Merge-sort is based on the divide-and-conquer paradigm. It involves the following three steps:

1.

2. Divide the array into two (or more) subarrays
3. Sort each subarray (Conquer)
4. Merge them into one to get the final sorted array



In the above given example the whole array is broken in two parts in such a way that single element is found at the last.Now these two elements for ex 40 and 28 are compared with each other and placed in the array and at the same time the other two elements are compared and with each other then with the previous elements i.e with 40 and 28 .Lastly we get an array of elements which are sorted.In this way sorting is done by splitting the array first and then combining them together

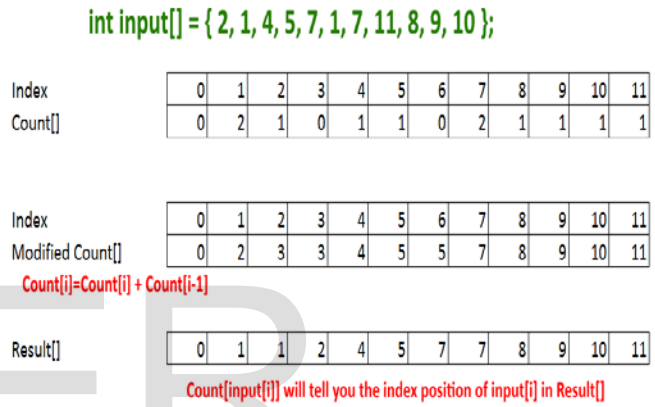
2.3 SORTING TECHNIQUES WITH ADDITIONAL STORAGE REQUIREMENT

COUNTSORT

Counting Sort is an sorting algorithm, which sorts the integers(or Objects) given in a specific range. In this numbering is done in such a way that it shows the occurrence of that number.

ALGORITHM OF COUNT SORT

- Take two arrays, Count[] and Result[] and given array is input[].
- Count[] will store the counts of each numbers in the given array.
- Update the Count[] so that each index will store the sum till previous step. (Count[i]=Count[i] + Count[i-1]). Now updated Count[] array will reflect the actual position of each integer in Result[].
- Now check the input array taking one element at a time, Count[input[i]] will tell you the index position of input[i] in Result[].



2.3.1 RADIX SORT

Radix sort was developed for sorting large integers, but it treats an integer as a string of digits, so it is really a string sorting algorithm unlike count sort. It compares the digits of the numbers and then it arranges the data according to tha basis.

The working of radix sort is explained below with the help of given example

Example:

Consider a group of numbers. It is given by the list:

123, 002, 999, 609, 111

STEP1:

Sort the list of numbers according to the ascending order of least significant bit. The sorted list is given by:

111, 002, 123, 999, 609

STEP2:

Then sort the list of numbers according to the ascending order of 1st significant bit. The sorted list is given by:
609, 002, 111, 123, 999

STEP3:

Then sort the list of numbers according to the ascending order of most significant bit. The sorted list is given by:

002 ,111, 123, 609 ,99

3. ANALYSIS

3.1 TIME COMPLEXITY COMPARISON OF SORTING ALGORITHMS

Algorithm	Time Complexity		
	Best	Average	Worst
Quick sort	O(n log(n))	O(n log(n))	O(n^2)
Merge sort	O(n log(n))	O(n log(n))	O(n log(n))
Heap sort	O(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	O(n)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Select Sort	O(n^2)	O(n^2)	O(n^2)
Count Sort	O(n+k)	O(n+k)	O(n^2)
Radix Sort	O(nk)	O(nk)	O(nk)

3.2 SPACE COMPLEXITY COMPARISON OF SORTING ALGORITHMS

Algorithm	Worst Case Auxiliary Space Complexity
Quick sort	O(n)
Merge sort	O(n)
Heap sort	O(1)
Bubble Sort	O(1)
Insertion Sort	O(1)
Select Sort	O(1)
Count sort	O(n+k)
Radix Sort	O(n+k)

4 CONCLUSION

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists and mostly-sorted list and is good only for sorting small arrays (usually less than 100 items). In fact, the smaller the array, the faster insertion sort is compared to any other sorting algorithm. However, being an $O(n^2)$ algorithm, it becomes very slow when the size of the array increases. Heap sort is not the fastest possible in all (nor in most) cases it also makes sure that it will not take extra memory, which is often a nice feature. Mege sort is Given that it is always $O(n\log n)$, it is a very good alternative. Its main problem is that it requires a second array with the same size as the array to be sorted, thus doubling the memory requirements.Quicksort is the most popular sorting algorithm and it is usually very fast. The main problem with quicksort is that it's not trustworthy: Its worse-case scenario is $O(n^2)$ (in the worst case it's as slow, if not even a bit slower than insertion sort) .Bubble sort is a straightforward and simplistic method of sorting data While simple, this algorithm is highly inefficient and is rarely used except in education.

5 REFERENCE

https://www.google.co.in/
http://faculty.simpson.edu/
http://scanftree.com/