Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Sorting of Data/5a12e62546765b2b63e34716

TUTORIAL

Sorting of Data

Topics

Sorting refers to arranging data in a particularly defined format. User may specify the way to arrange data in a particular order. Most common orders are in numerical (Ascending or Descending) or String (lexicographical) order. The importance of sorting is that data searching can be optimized to a very high level, if data is stored in a sorted manner. Also the sorted data will be in more readable format. Following are some of the examples of sorting in real-life scenarios –

Telephone Directory: – The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.

Dictionary: – The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

Tournament Teams: The tournament points table list the teams in descending order of their points, so that the position of each team can be determined.

So given a collection of items, and required ordering, arranging those items in that particular order is called sorting. For example, following are some numbers to be sorted,

```
4, 7, 1, 6, 12, 23, 54, 87, 2
```

There sorted arrangement is as below: -

```
1, 2, 4, 6, 7, 12, 23, 54, 87 (Ascending Order)
87, 54, 23, 12, 7, 6, 4, 2, 1 (Descending Order)
```

Various sorting algorithms exist and it is estimated that 90 % of the time spent by all computers is on sorting data. One of algorithm design principle says – "When in doubt, Sort the data". So it is really a problem which requires attention, requires consideration and so we need to develop good algorithms for it.

Before moving to sorting algorithms, we need to understand the terminology of sorting so that we can understand the things in a better way. Sorting algorithms can be classified in different categories based on following factors: -

1. **Time Complexity**: Based on the time taken by the algorithms.
2. **Space Complexity**: Based on the memory usage by the algorithms.
3. In-Place Sorting: If the space required by the algorithm is constant, that is called in-place sorting.
4. Out-Place Sorting: If algorithm requires another array of same size to sort an array, that is known as out-place sorting.
5. **Stability**: If two elements with same key appear in the same order in sorted output as they appear in the input array to be sorted, that is called stable sorting. For example, if we are sorting a set of cards having 5 of Heart and 5 of Diamond in the sequence such that Heart is coming before Diamond. Then in sorted order, obviously they come together, but if their relative ordering remains as it is, that is called stable sorting, otherwise sorting becomes unstable. Many times we need a stable sorting only.
6. **Internal or External**: If the algorithm is sorting the data in main memory of the CPU, it is called internal sorting. Some time algorithm may require to store the data at external memory during sorting, in that case it is called external sorting. External Sorting is used for massive amount of data. Merge Sort and its variations are typically used for external sorting.
7. **Recursive / Non-Recursive**: Many sorting algorithms use recursion to sort the data, whereas many algorithms can be implemented non-recursively also.
8. **Adaptive / Non-Adaptive Sorting**: A sorting algorithm is said to be adaptive, if it takes advantage of already 'sorted' elements in the list. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them. Whereas Non-Adaptive algorithms will force each element to be re-ordered to confirm their position.

Points needs to consider while sorting any particular data

- **How much data to sort**? This will help you know whether you need to look for an algorithm with a very low time complexity.

- **How sorted will your data be already**? Will it be partly sorted? Randomly sorted? This can affect the time complexity of the sorting algorithm. Most algorithms will have worst and best cases - you want to make sure you're not using an algorithm on a worst-case data set.
- **Time complexity is not the same as running time**: Remember that time complexity only describes how the performance of an algorithm varies as the size of the data set increases. An algorithm that always does one pass over all the input would be O(n) - it's performance is linearly correlated with the size of the input. But, an algorithm that always does two passes over the data set is also O(n) - the correlation is still linear, even if the constant (and actual running time) is different.
- **Space is not free**: Space complexity describes how much space an algorithm needs to run. For example, a simple sort such as insertion sort needs an additional fixed amount of space to store the value of the element currently being inserted. This is an auxiliary space complexity of O(1) - it doesn't change with the size of the input. However, merge sort creates extra arrays in memory while it runs, with an auxiliary space complexity of O(n). This means the amount of extra space it requires is linearly correlated with the size of the input.

Of course, algorithm design is often a trade-off between time and space - algorithms with a low space complexity may require more time, and algorithms with a low time complexity may require more space

Sorting of data can be done in two manners: comparison based sorting and Non-comparison based sorting. All the comparison based sorting algorithms use comparisons to determine the relative order of elements. e.g., insertion sort, merge sort, quick sort, heap sort. The best worst-case running time that we've seen, for comparison sorting is O(n log n). It is also possible to sort the numbers without comparing them, instead by making certain assumption about data. The process is known as non-comparison sorting and algorithms are known as the non-comparison based sorting algorithms.

There are sorting algorithms that run faster than this but they require special assumptions about the input sequence to be sort. Examples of sorting algorithms that run in linear time are counting sort, radix sort and bucket sort. Counting sort and radix sort assume that the input consists of integers in a small range. Whereas, bucket sort assumes that the input is generated by a random process that distributes elements uniformly over the interval. Linear-time sorting algorithms use operations other than comparisons to determine the sorted order.

A comparison sort algorithm compares pairs of the items being sorted and the output of each comparison is binary (i.e. smaller than or not smaller than). Radix sort considers the digits of the numbers in sequence and instead of comparing them it groups numbers in buckets with respect to the value of the digit (in a stable manner). Note that the digit is not compared to anything - it is simply put in a bucket corresponding to its value. So a non-comparison sort algorithm uses the internal character of the values to be sorted. It can only be applied to some particular cases, and requires particular values. And the best complexity is probably better depending on cases, such as O(n).

Some real life experiments for sorting

Following are some results executed at server side with same input on different sorting algorithms, the results here are the averages of 10 runs to show you a glance of time taken by these algorithms:

```
    N        Bubble   Selection   Insertion   Heap   Merge
  Quick   Radix10
   10,000     .454      .687        .354       *      *       *
       *
  100,000     25.6      47.0        27.4      .059   .050
    .031     .093
1,000,000       +         +           +       1.359   .850
    .431    1.933
               S,I      U,I         S,I       U,I     S,N
     U,I     S,N

*   = Cannot be timed: <.001 seconds
+   = Not timed      : > 1 minute
U/S = Unstable/Stable
I/N = In place (requires < O(N) more space: O(1) or O(Log N))/Not
in place
```

For more details, look at the Wikipedia article on Sorting Algorithms:

http://en.wikipedia.org/wiki/Sorting_algorithm