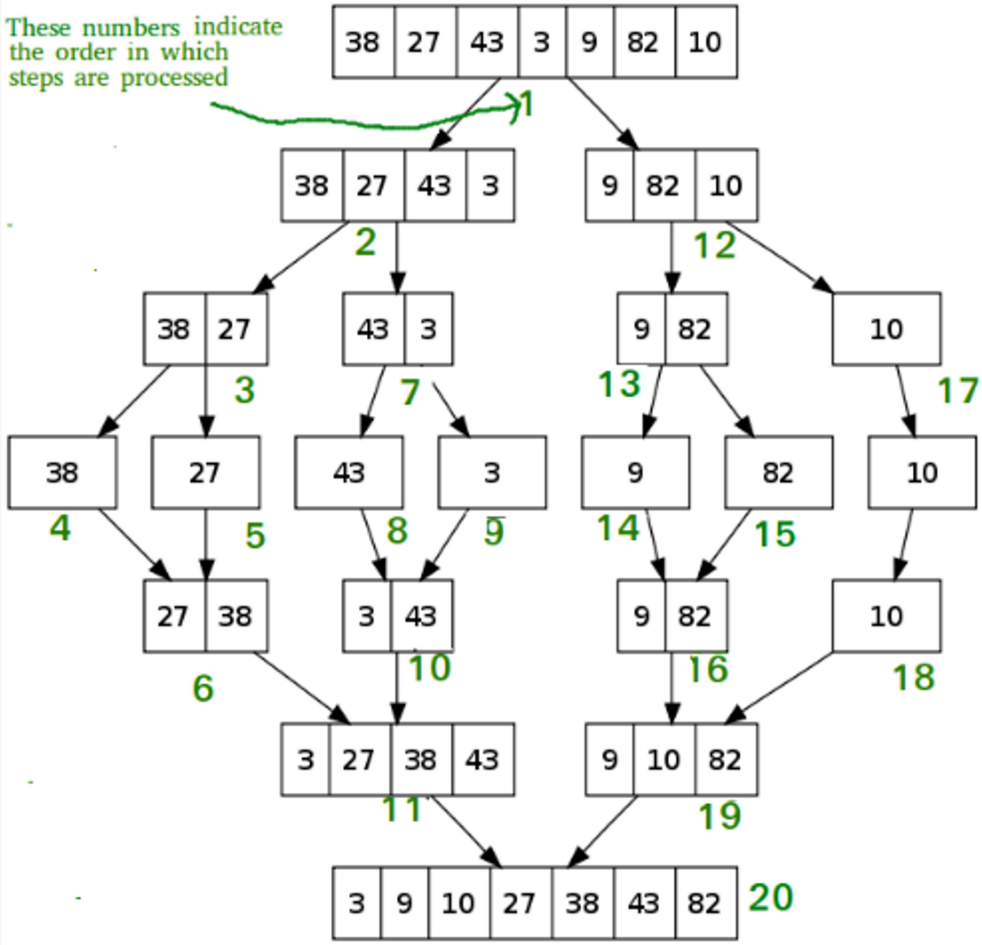
June4

03 June 2020 10:48

Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.



Time complexity of Merge Sort is

$$\Theta(nLogn)$$

in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and take linear time to merge two halves.

Auxiliary Space: O(n)

Algorithmic Paradigm: Divide and Conquer

Applications :

1. Merge Sort is useful for sorting linked lists in O(nLogn) time.In the case of linked lists,

the case is different mainly due to the difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory.

2. Inversion Count Problem.
3. Used in External Sorting.

- **Inversion Count**
  Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.
  Formally speaking, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

  Example

```
Input: arr[] = {8, 4, 2, 1}
Output: 6

Explanation: Given array has six inversions:
(8,4), (4,2),(8,2), (8,1), (4,1), (2,1).


Input: arr[] = {3, 1, 2}
Output: 2

Explanation: Given array has two inversions:
(3, 1), (3, 2)
```

Approach : Traverse through the array and for every index find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all index in the array and print the sum.

Algorithm :

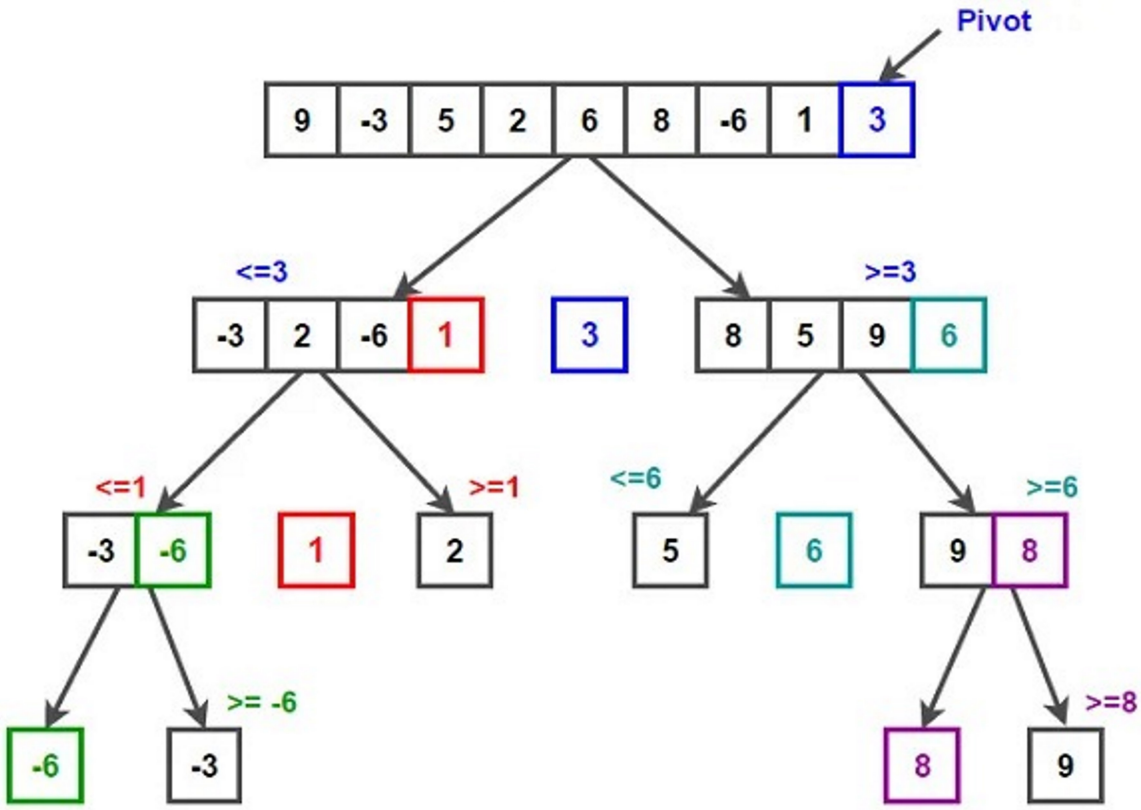
- Traverse through the array from start to end
- For every element find the count of elements smaller than the current number up to that index using another loop.
- Sum up the count of inversion for every index.
- Print the count of inversions.

Quick Sort

Like Merge Sort, Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick
Sort that pick pivot in different ways.

- Always pick first element as pivot.
- Always pick last element as pivot (implemented below)

- Pick a random element as pivot.
- Pick median as pivot.



✓ Quicksort is an in-place sorting algorithm, which means it does not require any extra/temporary list to perform sorting, everything will be done on the original list itself.

✓ Quicksort when implemented well it is one of the best sorting algorithms, In fact, the sort function provided in most of the language libraries is the implementation of Quicksort itself.

Algorithm:

- Pick an element, called a pivot, from the array.
- Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
- Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Pseudo Code for recursive Quick Sort function :

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Questions

1. **Explain inversion count. Write a python program to find inversion count.**
2. **What do you understand by External Sorting in context with merge sort?**
3. **Implement merge sort and sort a given link list.**
4. **Implement quick sort and discuss role of pivot.**