Thursday, 4 March 2021

Introduction to Algorithms - MIT

Click Here to get link to the course.

Lecture 1 - Introduction; Analysis of Algorithms, Insertion Sort, Mergesort

• Introduction

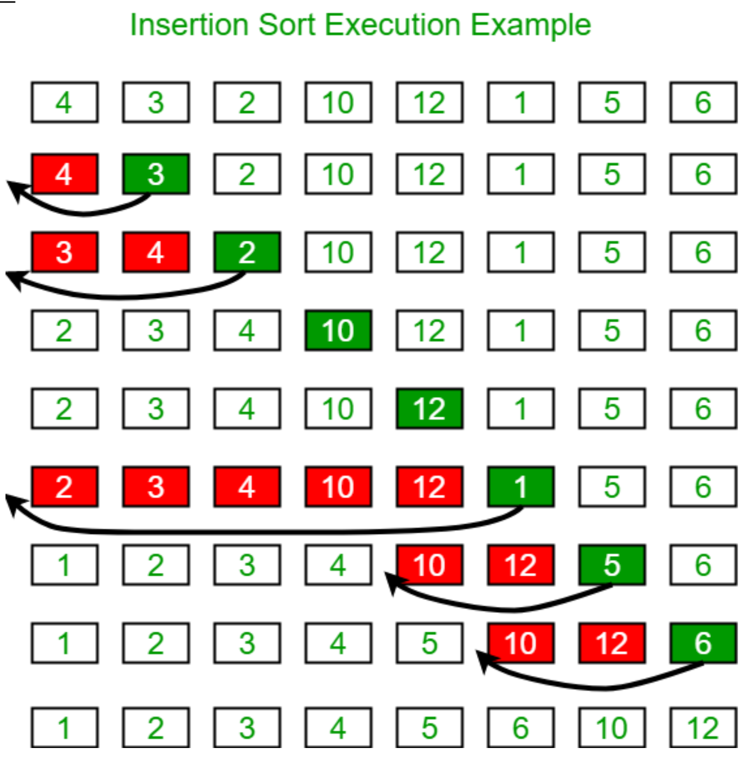
- For ***Performance***
- Algorithms are to make things faster so that any developer can compensate it with other features like security, functionality etc.

• Insertion Sort

- **Pseudo Code**
- To sort an array of size n in ascending order:
- 1: Iterate from array[1] to array[n] over the array.
- 2: Compare the current element (key) to its predecessor.
- 3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Worst case time complexity is O(n^2) - Moderately for small n but not at all for large n.

- **Example:**



- **<u>Source Code of insertion sort:</u>**

```
void sort(int[] arr)
{
    int n = arr.Length;

    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

- **Running Time Variables**

  - Input (Already Sorted)
  - Input size
  - Nant Upper bound (Maximum seconds/time it is gonna take to execute)

- **Kinds of Analysis**

  - Worst-Case: The maximum time to execute on any input of size n.
  - Average-Case: mean for all the probable numbers of n.
  - Best-Case(Bogus): Even a slower algorithm can do best.

All the above depending on the machine's hardware - Absolute or relative speed.

- **Asymptotic Analysis**

Ignore Machine dependance and see how it grows with T(n)

  - Theta O(n): ignore all the lower order terms along with the constants.

- **Merge Sort of [1..n]**

- If n ==1 : Done
- Recursively sort from [A[1] to A[n/2]] and [A[n/2] to A[n]].
- Merge two sorted lists

- **<u>Key sub-routine:</u>**

- Compare the starting of first two lists and compare to find to add to sorted list

- **Pseudo Code:**

```
// Merges two subarrays of []arr.
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int[] arr, int l, int m, int r)
{
    // Find sizes of two
    // subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temp arrays
    int[] L = new int[n1];
    int[] R = new int[n2];
    int i, j;

    // Copy data to temp arrays
    for (i = 0; i < n1; ++i)
        L[i] = arr[l + i];
    for (j = 0; j < n2; ++j)
        R[j] = arr[m + 1 + j];

    // Merge the temp arrays

    // Initial indexes of first
    // and second subarrays
    i = 0;
    j = 0;

    // Initial index of merged
    // subarry array
    int k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements
    // of L[] if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy remaining elements
    // of R[] if any
    while (j < n2) {
        arr[k] = R[j];
```

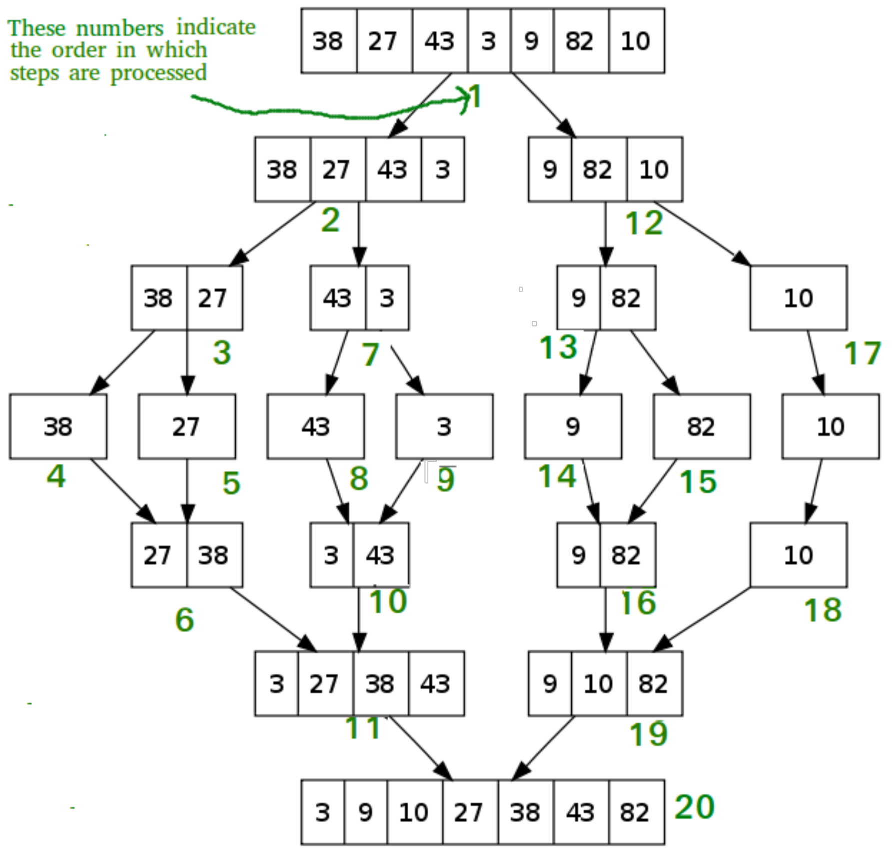
```
            j++;
            k++;
        }
    }

    // Main function that
    // sorts arr[l..r] using
    // merge()
    void sort(int[] arr, int l, int r)
    {
        if (l < r) {
            // Find the middle
            // point
            int m = l+ (r-l)/2;

            // Sort first and
            // second halves
            sort(arr, l, m);
            sort(arr, m + 1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }
```

- **Example:**


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20