Tutorial -2

****

- In this tutorial, we will go through 3 sorting methods: **Quick Sort**, **Merge Sort,** and **Insertion Sort**.

****Note**: Visit the following link to better understand the sorting algorithms through visualization

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

****

Insertion Sort:

1. **Introduction:**

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part.

2. **Characteristics:**

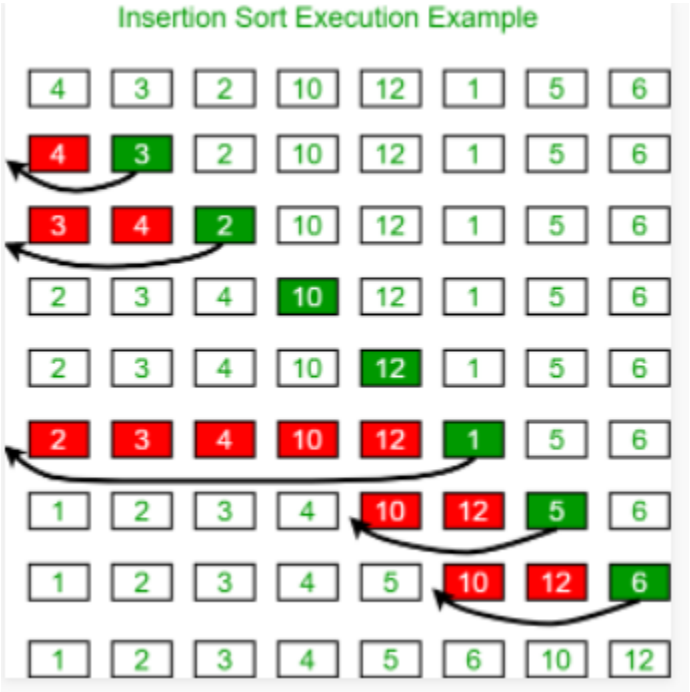
- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

3. **Pseudo-code:**

```
INSERTION-SORT(A)
  for i = 1 to n
        key ← A [i]
        j ← i – 1
```

```
        while j > = 0 and A[j] > key
                A[j+1] ← A[j]
                j ← j – 1
        End while
        A[j+1] ← key
End for
```

4. **Demonstration: (Explain below example through the pseudocode provided)**



5. **Time Complexity:**

Even though insertion sort is efficient, still, if we provide an already sorted array to the insertion sort algorithm, it will still execute the outer for loop, thereby requiring n steps to sort an already sorted array of n elements, which makes its best case time complexity a linear function of n.

Wherein for an unsorted array, it takes for an element to compare with all the other elements which mean every n element compared with all other n elements. Thus, making it for n x n, i.e., n2 comparisons.

Worst Case Time Complexity [Big-O]: $O(n^2)$

Best Case Time Complexity [Big-omega]: O(n)

Average Time Complexity [Big-theta]: $O(n^2)$

*****Few important points regarding Insertion Sort:**

- Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.
- It is an in-place sorting algorithm(no auxiliary space is required)
- It is a stable sorting algorithm(relative positions of elements in sorted and actual array are same)
- Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array

6. Questions:

Question 1)

Demonstrate the insertion sort results for each insertion for the following initial array of elements . 54 26 93 17 77 31 44 55 20

Question 2) (Very important)

Inversion Pair - In a permutation a1...an, of n distinct integers, an inversion is a pair (ai,aj) such that i<j and ai>aj.

What would be the worst case time complexity of the Insertion Sort algorithm, if the inputs are restricted to permutations of 1...n with at most n inversions?

A) Θ(n^2)
B) Θ(nlogn)
C) Θ(n^1.5)
D) Θ(n)

Quick Sort:

Question 1)

Suppose you are provided with the following function declaration in the C programming language.

int partition(int a[], int n);

The function treats the first element of a[ ] as a pivot and rearranges the array so that all elements less than or equal to the pivot are in the left part of the array, and all elements greater than the pivot are in the right part. In addition, it moves the pivot so that the pivot is the last element of the left part. The return value is the number of elements in the left part. The following partially given function in the C programming language is used to find the kth smallest element in an array a[ ] of size n using the partition function. We assume k≤n.

```
int kth_smallest (int a[], int n, int k)
{
   int left_end = partition (a, n);
   if (left_end+1==k) {
      return a[left_end];
   }
   if (left_end+1 > k) {
      return kth_smallest (_____);
   } else {
      return kth_smallest (_____);
   }
}
```

The missing arguments lists are respectively

A. (a,left_end,k) and (a+left_end+1,n-left_end-1,k-left_end-1)
B. (a,left_end,k) and (a,n-left_end-1,k-left_end-1)
C. (a+left_end+1,n-left_end-1,k-left_end-1) and (a,left_end,k)
D. (a,n-left_end-1,k-left_end-1) and (a,left_end,k)

Question 2)

Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Then what will be the pivot number?

Question 3)

Apply Quick sort on a given sequence 7 11 14 6 9 4 3 12. What is the sequence after first phase, pivot is first element?

Question 4)

Let P be a quicksort program to sort numbers in ascending order. Let t1 and t2 be the time taken by the program for the inputs [1 2 3 4] and [5 4 3 2 1], respectively. Which of the following holds?

A) t1=t2

B) t1>t2

C) t1<t2

D) t1=t2+5log5

Question 5)

In quick-sort, for sorting n elements, the (n/4)th smallest element is selected as pivot using an O(n) time algorithm. What is the worst case time complexity of the quick sort?

A)Θ(n)

B)Θ(nlogn)

C)Θ(n2)

D)Θ(n2logn)

Merge Sort:

Question 1)

Demonstrate the merge sort results for each iteration for the following initial array of elements . 54 26 93 17 77 31 44 55 20

Question 2)

For merging two sorted lists of sizes m and n into a sorted list of size m+n, we require comparisons ?

Answer:

O(m+n)

Question 3)

If one uses straight two-way merge sort algorithm to sort the following elements in ascending order:

20, 47, 15, 8, 9, 4, 40, 30, 12, 17

Then the order of these elements after second pass of the algorithm is:

A) 8, 9, 15, 20, 47, 4, 12, 17, 30, 40
B) 8, 15, 20, 47, 4, 9, 30, 40, 12, 17
C) 15, 20, 47, 4, 8, 9, 12, 30, 40, 17
D) 4, 8, 9, 15, 20, 47, 12, 17, 30, 40

Question 4)

Assume that a mergesort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

A)256
B)512
C)1024
D)2018

Question 5)

The worst case running times of Insertion sort , Merge sort and Quick sort, respectively are:

A)Θ(nlogn), Θ(nlogn) and Θ(n2)
B)Θ(n2), Θ(n2) and Θ(nlogn)
C)Θ(n2), Θ(nlogn) and Θ(nlogn)
D)Θ(n2), Θ(nlogn) and Θ(n2)