جامعة طرابلس ـ كلية تقنية المعلومات

Design and Analysis Algorithms

تصميم وتحليل خوارزميات

ITGS301

المحاضرة الرابعة: Lecture 4

خريف 2024

The Problem Of Sorting

Sorting

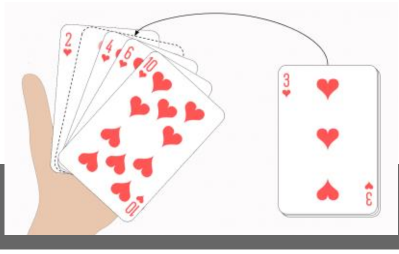
- Sorting is one of the most common data processing applications , the through which data are arranged according to their values.

- If data were not ordered , we would spend hours trying to find a single piece of information.

- Data may be sorted in either ascending sequence or descending sequence . and if the order of the sort is not specified its assumed to be ascending..

Example: Sorting Playing Cards

Let us start with a playing card example.
Imagine being handed one card at a time. You take the first card in your hand. Then you sort the second card to the left or right of it. The third card is placed to the left, in between or to the right, depending on its size. And also, all the following cards are placed in the right position.



Insertion Sort

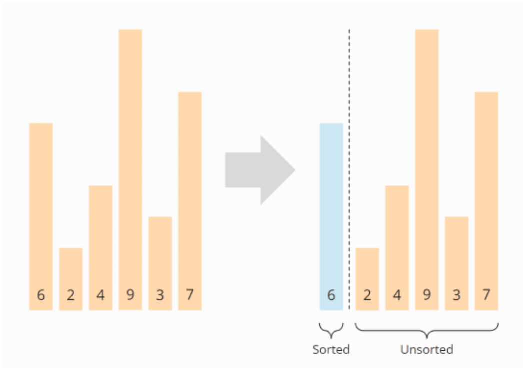
- In the insertion sort , the list is divided in two parts : sorted and unsorted.
- In each pass the first element of the unsorted sub list is transferred to the sorted sub list by inserting it at appropriate place.
- If we have a list of n elements , it will take at most n-1 passes to sort the data.

Insertion Sort Algorithm

Let's move from the card example to the computer algorithm. Let us assume we have an array with the elements [6, 2, 4, 9, 3, 7]. This array should be sorted with Insertion Sort in ascending order.

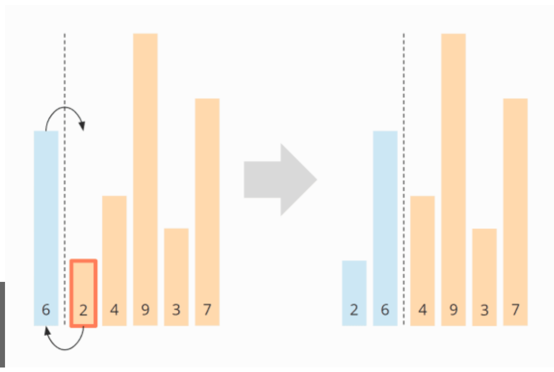
Step 1

First, we divide the array into a left, sorted part, and a right, unsorted part. The sorted part already contains the first element at the beginning, because an array with a single element can always be considered sorted.



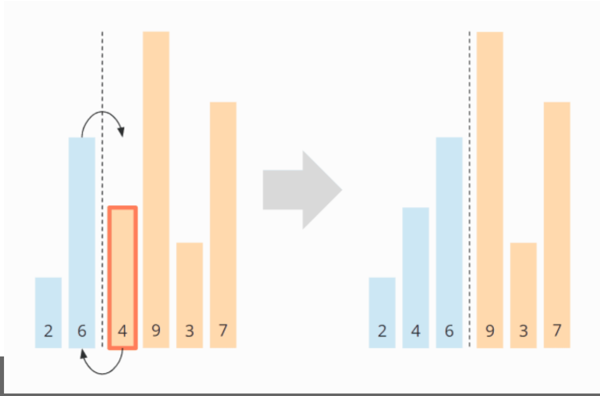
Step 2

Then we look at the first element of the unsorted area and check where, in the sorted area, it needs to be inserted by comparing it with its left neighbor.
In the example, the 2 is smaller than the 6, so it belongs to its left. In order to make room, we move the 6 one position to the right and then place the 2 on the empty field. Then we move the border between sorted and unsorted area one step to the right:



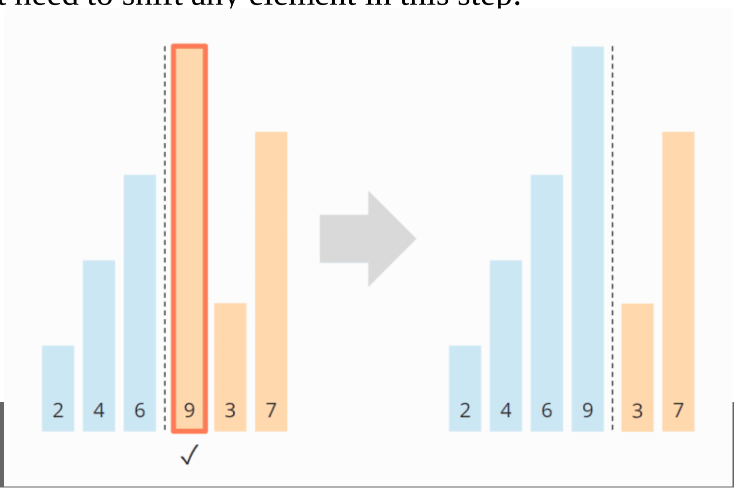
Step 3

We look again at the first element of the unsorted area, the 4. It is smaller than the 6, but not smaller than the 2 and, therefore, belongs between the 2 and the 6. So we move the 6 again one position to the right and place the 4 on the vacant field:



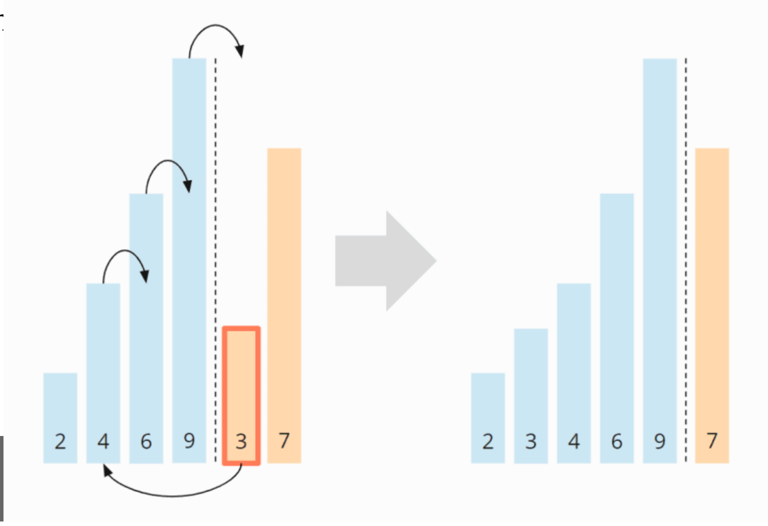
Step 4

The next element to be sorted is the 9, which is larger than its left neighbor 6, and thus larger than all elements in the sorted area. Therefore, it is already in the correct position, so we do not need to shift any element in this step:



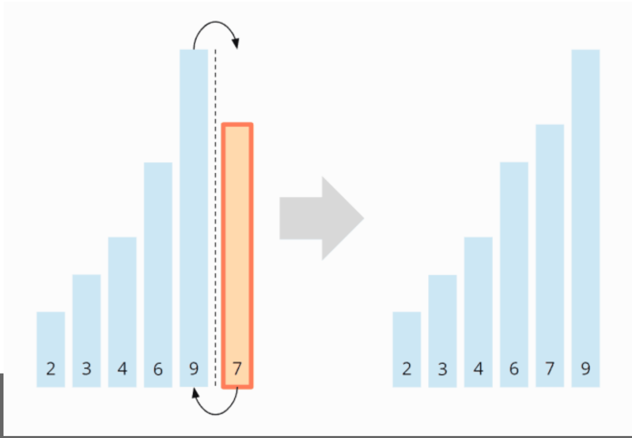
Step 5

The next element is the 3, which is smaller than the 9, the 6 and the 4, but greater than the 2. So we move the 9, 6 and 4 one position to the right and then put the 3 wh



Step 6

That leaves the 7 – it is smaller than the 9, but larger than the 6, so we move the 9 one field to the right and place the 7 on the vacant position:

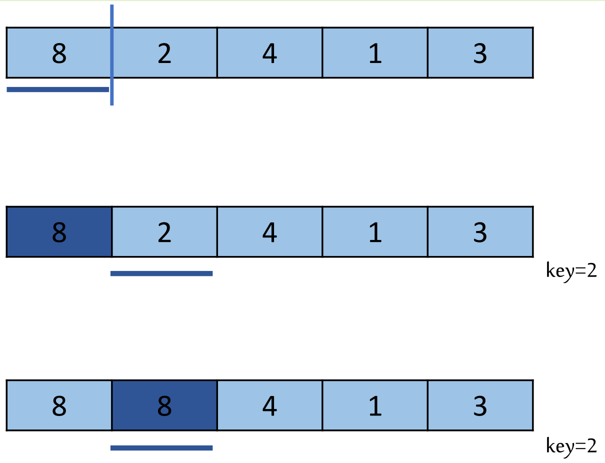


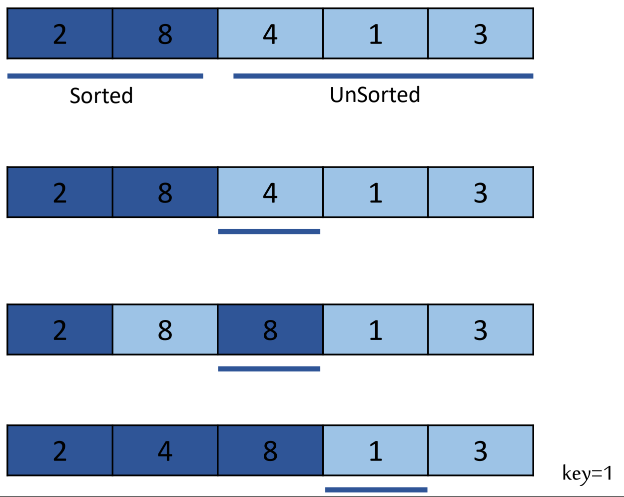
```
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j – 1
        }
        A[j+1] = key

    }
}
```

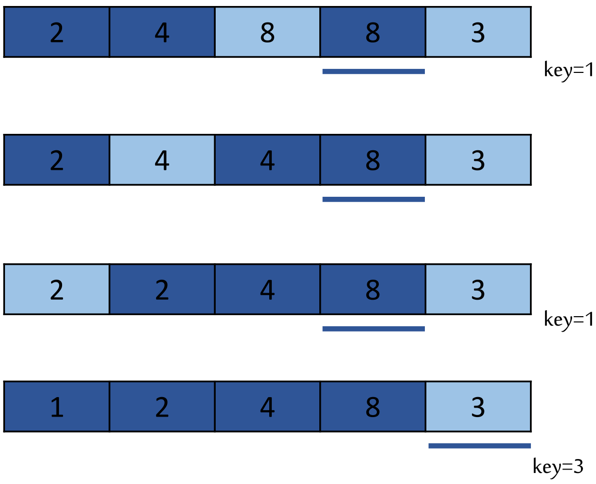
```
InsertionSort( A, n)
{
For i = 2 to n {

   Key = A[i]
   j= i - 1

While ( j > 0) and (A[j] > key)
 {
  A[j+1] = A[j]
  j = j – 1
 }
 A[ j+1] = key
 }
```

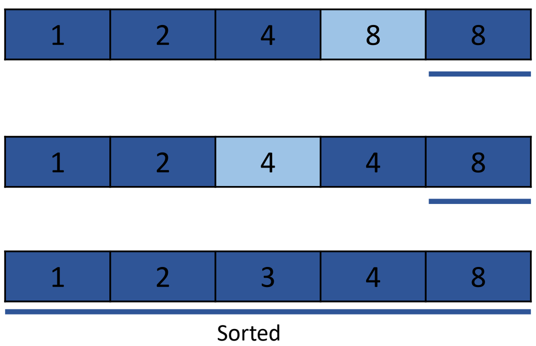
8	2	4	1	3

Example 1 :


8 2 4 1 3
8 2 4 1 3
key=2
8 8 4 1 3
key=2


2 8 4 1 3
Sorted
UnSorted
2 8 4 1 3
2 8 8 1 3
2 4 8 1 3
key=1


2 4 8 8 3
key=1
2 4 4 8 3
2 2 4 8 3
key=1
1 2 4 8 3
key=3


1 2 4 8 8
1 2 4 4 8
1 2 3 4 8
Sorted

Time Complexity

For i=2 to n {	n
Key= A [i]	n-1
j=i-1	n-1
While (j>0 & A [j] > key) {	$\sum_{i=2}^{n} ti$
A [j+1] = A [j]	$\sum_{i=2}^{n} (ti - 1)$
j=j-1	$\sum_{i=2}^{n} (ti - 1)$
}	
A [j+1] = key	n-1
}	

where t is the number of while tests

Number of times inner-loop is executed depends on the input

The worst case scenario for Insertion Sort is if the array is already sorted, but with the highest values first. That is because in such a scenario, every new value must "move through" the whole sorted part of the array.

These are the operations that are done by the Insertion Sort algorithm for the first elements:

- The 1st value is already in the correct position.
- The 2nd value must be compared and moved past the 1st value.
- The 3rd value must be compared and moved past two values.
- The 3rd value must be compared and moved past three values.
- And so on..

If we continue this pattern, we get the total number of operations for n values:

1+2+3+...+(n−1)

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Time Complexity

Statement	Time	Best case	Worst case
For i = 2 to n {	n	n	n
Key = A[i]	n-1	n-1	n-1
j= i - 1	n-1	n-1	n-1
While (j > 0) and (A[j] > key)	$\sum_{i=2}^{n}(ti)$	n-1	n(n+1)/2 -1
A[j+1] = A[j]	$\sum_{i=2}^{n}(ti-1)$	$\sum_{i=2}^{n}(1-1)$ = 0	n(n-1)/2
j = j – 1	$\sum_{i=2}^{n}(ti-1)$	$\sum_{i=2}^{n}(1-1)$ = 0	n(n-1)/2
A[j+1] = key	n-1	n-1	n-1

$\sum_{i=2}^{n} ti = n-1$

Best case running time:

$\sum_{i=2}^{n} (ti-1) = 0$

T(n) = n + (n-1) + (n-1) + (n-1) + 0 + 0 + (n-1)
= 5n – 4

T(n) = O (n)

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Worst case running time:

T(n) = n + (n-1) + (n-1) + (n(n+1)/2 – 1) + n(n-1)/n + n(n-1)/2 + (n-1)

T(n) = O (n^2)

$\sum_{i=2}^{n} ti = n(n+)1/2$

$\sum_{i=2}^{n} (ti-1) = n(n-1)/2$

Summary

Insertion Sort is an easy-to-implement, stable sorting algorithm with time complexity of $O(n^2)$ in the average and worst case, and $O(n)$ in the best case.

- It could be used in sorting small lists.
- It could be used in sorting "almost sorted" lists.
- It could be used to sort smaller sub problem in Quick Sort.

The End .