20186039

Week - 4 Exam

Non-uniform Data:

Write a client that generates test data that is not uniform,
including the following:

- Half the data is 0s, half 1s.
- Half the data is 0s, half the remainder is 1s, half the remainder is 2s, and so forth.
- Half the data is 0s, half random int values.

Develop and test hypotheses about the effect of such input on the performance of the algorithms in this section.

Aim : To sort the elements in the given array using selection and insertion sorts. And also to check the time taken by them for sorting different form of inputs.

Procedure :

Selection Sort :

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where **n** is the number of items.

How Selection Sort Works?

Consider the following depicted array as an example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in a sorted manner.

The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process –

Selection Sort algorithm:

Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted

Insertion Sort :

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

How Insertion Sort Works?

We take an unsorted array for our example.

Insertion sort compares the first two elements.

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

And finds that 33 is not in the correct position.

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

These values are not in a sorted order.

So we swap them.

However, swapping makes 27 and 10 unsorted.

Hence, we swap them too.

Again we find 14 and 10 in an unsorted order.

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

Insertion Sort algorithm:

Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

Results :

Test Cases :

1. Half the data is 0's, half 1's

Example :

Input :
01010101010101010101
Output :
00000000001111111111

Time taken to sort is:
For Selection Sort : 0.004 seconds
For Insertion Sort : 0.003 seconds

The input data contains only 0's and 1's. In selection as we know that it compares all the elements and takes the minimum element from the array. Where as in insertion sort, if the elements are in ascending order it leaves it. Otherwise keeps it in correct position.

2. Half the data is 0's half the remainder is 1's, half the remainder is 2's, and so forth.

Example :
Input :
02030100050003040501240120530 1
Output :
00000000000000001111222333445 55

Time taken to sort is :
For Selection Sort : 0.006 seconds
For Insertion Sort : 0.005 seconds

As we know selection sort checks all the elements it takes more time than insertion to sort.

3. Half the data is 0's and half random int values.

Example :
Input :
01060175216801430916063102070560401080230650108749302060704018605020 13
Output :
00000000000000000000000001111111111222223333344445555666666667777888899

Time taken to sort is :
For Selection Sort : 0.012 seconds
For Insertion Sort : 0.010 seconds

For the random case also insertion sort takes less time when compared to selection sort.

Conclusion : Comparing the three cases of input, the insertion sort algorithm gives the better time complexity results than selection sort.