21118
Name: Shubham Chemate
(SE-I-E-1)

19-oct

Assignment 05

Sorting Algorithms - Insertion Sort, Shell Sort

# Problem Statement
Write a python program to store second year percentages of students in an array. Write function for sorting array of floating point numbers in ascending order using
a> Insertion sort
b> shell sort
and display top five scores.

# Objectives:
1> To understand working of insertion sort & shell sort algorithm.
2> To able to use list in python to implement this algorithms.

# Software requirement:
Operating system: Windows 10 home, single language
Python version: 3.8.5
vs code (text editor) : Sept 2020 Version.

# Hardware requirement:
Manufacturer: Acer
Processor: Intel (R) Core i5 - 8265U CPU @ 1.6GHz 1.8GHz.
System Type: 64-bit operating system, x-64 based processor

# Theory:
a> Insertion Sort: In this, array is sorted by inserting one element from unsorted part to sorted part one at a time.
Initially we have all elements in unsorted part & finally we have all elements in sorted part.
The algorithm works by inserting element in sorted

part, hence it is called insertion sort algorithm.

b) Shell Sort:

i) It is a modification of insertion sort algorithm.

ii) The insertion sort algorithm is improved to work better in some cases, & this is called shell-sort.

iii) The shell sort works by breaking the original list into number of smaller sublists after which each one is sorted using insertion sort.
Instead of breaking list into sublist of contiguous item, it uses an increment which is called gap to create sublist by choosing all items that are i apart.

Pseudo Code:

1) Insertion Sort:

```
Algorithm   InsertSort(list, n):
    for i ← 1 to n:
        val ← list[i]
        hole ← i
        while (hole > 1 and list[hole-1] > val)
            list[hole] ← list[hole-1]
            hole ← hole - 1
        list[hole] = val.
```

2) Shell Sort (~~using~~ gap will be halved after each pass)

```
Algorithm   ShellSort(list, n)
    gap ← n/2
    while (gap > 0):
        j = gap
        while (j < n):
            i = j - gap
            while (i >= 0):
```

```
            if ( list [i+gap] < list [i] ) : swap(list[i+gap], list[i])
            i ← i-gap
        j ← j+1
    gap ← gap/2
    end.
```

ADT for class:

```
class Scoresheet:
    def constructor-function():
        //initialize empty list
        list = [], n=0

    def insertionSort():
        // logic of insertion sort

    def shellSort ():
        // logic of shell sort

    def PrintTop5 ():
        // prints top 5 students marks.
```

Time & space complexity Analysis.

1) Insertion sort:

~~Bestcase~~: Space complexity: The algorithm doesn't require any additional space. Hence it's a constant space algorithm. Space complexity: $O(n)$

Time complexity:

Best case: In case of sorted array: $O(n)$

Worst case: In case of reverse-sorted array: $O(n^2)$

$\therefore$ time complexity of insertion sort is $O(n^2)$

⇒ Shell sort
→ Space complexity: The algorithm doesn't require additional space. O(n) space is required to store the array
→ Time complexity: The time complexity of algorithm is $O(n^2)$ but the algorithm will have less number of comparison compared to insertion sort and it is a improvement over insertion sort.

Testcases

Testcase no	Testcase Description	Expected o/p		Actual o/p.	
1.	Insertion sort (No duplicates)	rank	marks	rank	marks
	list = {90.22, 85.23, 96.28,	1	96.28	1	96.28
	53.55, 82.91, 93.26}	2	93.26	2	93.26
		3	90.22	3	90.22
		4	85.23	4	85.23
		5	82.91	5	82.91
2.	Shell Sort (No duplicates)	Rank.	marks	Rank	marks
	list = {90.22, 85.23,	1	96.28	1	96.28
	96.28, 53.55, 82.91, 93.26}	2	93.26	2	93.26
		3	90.22	3	90.22
		4	85.23	4	85.23
		5	82.91	5	82.91
3.	Insertion Sort (with dupli:)	Rank	marks	Rank	marks
	list: {90.21, 88.26, 90.21,	1	93.83	1	93.83
	92.53, 88.26, 93.83}	2	92.53	2	92.53
		3	90.21	3	90.21
		4	88.26	4	88.26
4.	Shell Sort (with duplicates)	Rank	marks	Rank	marks
	list: {90.21, 88.26, 90.21,	1	93.83	1	93.83
	92.53, 88.26, 93.83}	2	92.53	2	92.53
		3	90.21	3	90.21
		4	88.26	4	88.26

Applications:

The insertion sort & shell sort are one of the simplest sorting algorithm.
They can be used to sort data in complicated scenarios where O(nlogn) sorting algorithms cannot be used.

Conclusion:

At the end of assignment I'm capable of implementing insertion sort & shell sort algorithm.
~~Also~~ Also able to analyze space & time complexity of algorithms.