



## FACULTY OF ENGINEERING & TECHNOLOGY

### Lecture -03

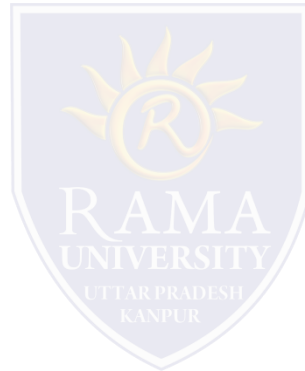
Mr. Nilesh

Assistant Professor

Computer Science & Engineering

# □ Outline

## ❖ SHELL SORT



## ➤ Introduction

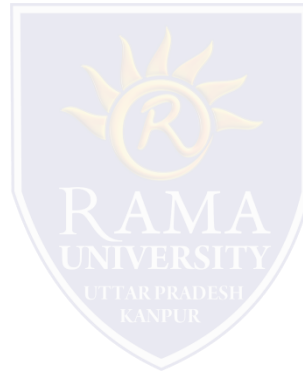
- ❖ Shell sort is one of the oldest sorting algorithm devised by Donald Shell in 1959.
- ❖ Shell sort is also called the diminishing increment sort.
- ❖ The shell sort was introduced to improve the efficiency of simple insertion sort
- ❖ In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved.
- ❖ The shell sort improved the insertion sort by comparing elements far apart, then elements less far apart and finally comparing adjacent elements.
- ❖ The shell sort algorithm avoids large shifts as in the case of insertion sort, if the smaller values is to the far right and has to be moved to the far left.
- ❖ The shell sort improved insertion sort by breaking the original list into a number of smaller sublist, each of which is sorted using an insertion sort.
- ❖ The unique way these sublists are chosen is through the key to the shell sort.
- ❖ Instead of breaking the list into sublists of contiguous items, the shell sort uses an incremental  $i$ , sometimes called the gap or interval to create a sublist by choosing all  $i$  items that  $i$  items apart.
- ❖

# □ How Shell Sort Works steps

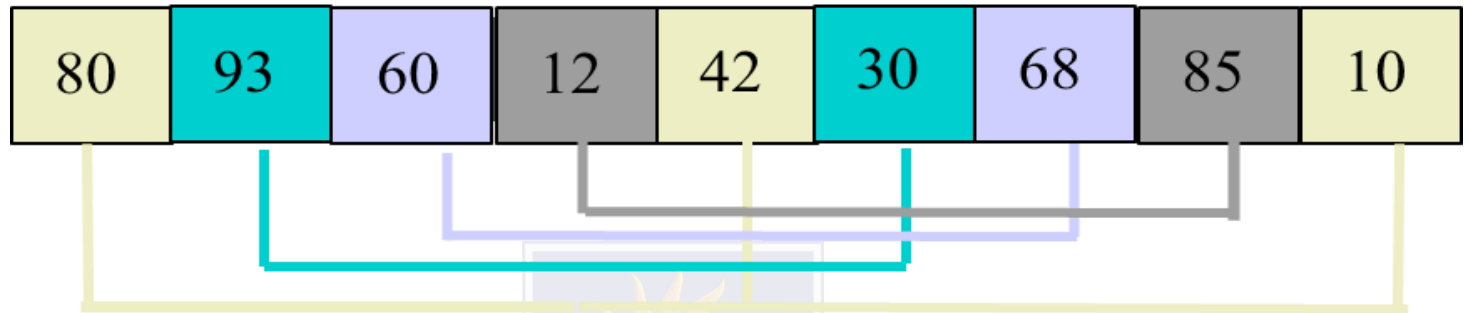
Consider the following unsorted array A with elements

80, 93, 60, 12, 42, 30, 68, 85, 10

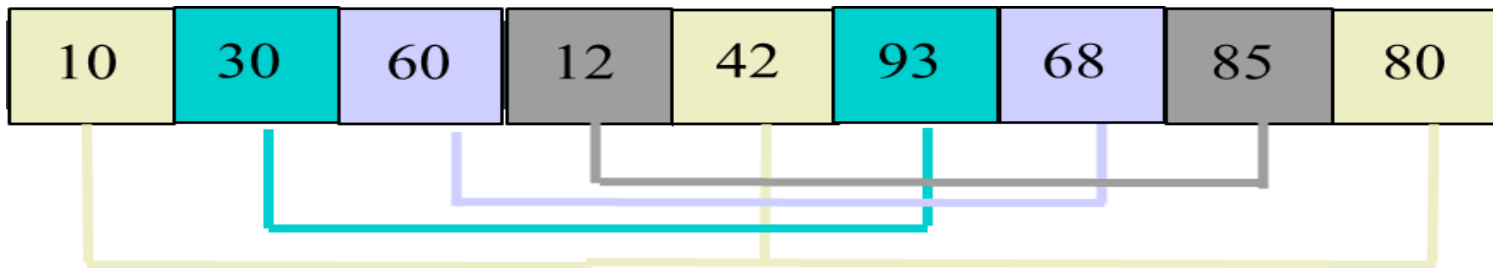
Sort using the Shell sort.



Initial Segmenting gap =  $\text{INT}\left(\frac{9}{2}\right) = 4$

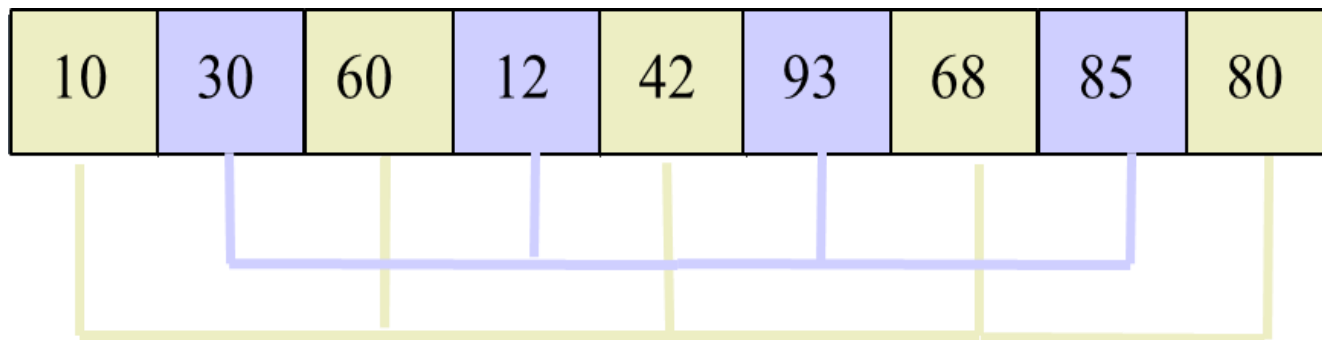
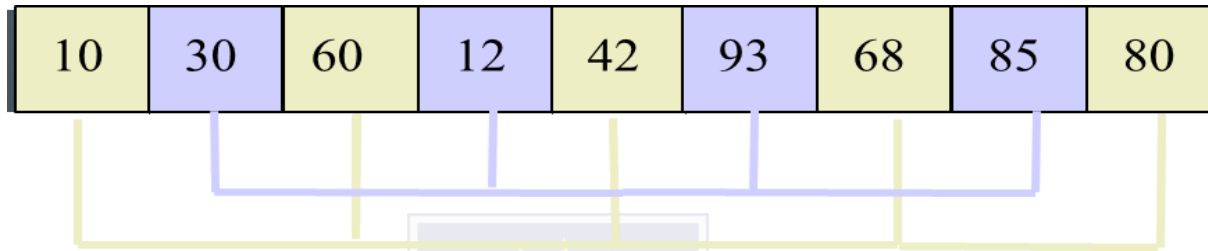


No of comparison: 6  
No of swaps: 4



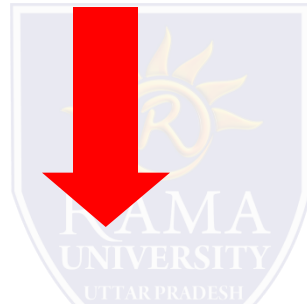
# Re segmenting Gap

$$\text{INT}\left(\frac{4}{2}\right) = 2$$



$$\text{Re-segmenting Gap} = \text{INT} \left( \frac{2}{2} \right) = 1$$

10	12	42	30	60	85	68	93	80
----	----	----	----	----	----	----	----	----



10	12	30	42	60	68	80	85	93
----	----	----	----	----	----	----	----	----

Class work

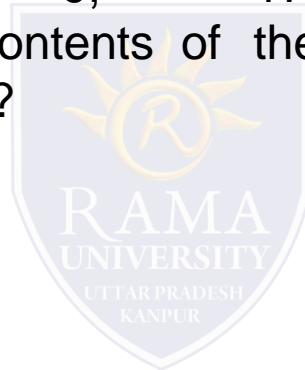
Q1.

Given the following list of numbers:

[5, 16, 20, 12, 3, 8, 9, 17, 19, 7]

Which answer  
complete for

illustrates the contents of the list after all swapping is  
a gap size of 3?





- (A) [5, 3, 8, 7, 16, 19, 9, 17, 20, 12]

(Correct! Each group of numbers represented by index positions 3 apart are sorted correctly)

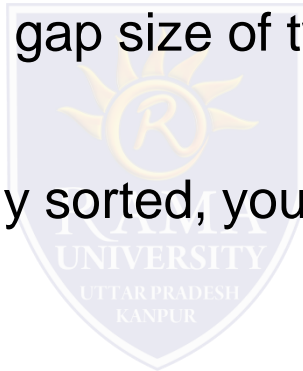
- (B) [3, 7, 5, 8, 9, 12, 19, 16, 20, 17]

(Incorrect. This solution is for a gap size of two.) (C) [3, 5, 7, 8, 9, 12, 16, 17, 19, 20]

- (Incorrect. This is list completely sorted, you have gone too far.)

- (D) [5, 16, 20, 3, 8, 12, 9, 17, 20, 7]

(Incorrect. The gap size of three indicates that the group represented by every third number e.g. 0, 3, 6, 9 and 1, 4, 7 and 2, 5, 8 are sorted not groups of 3.)



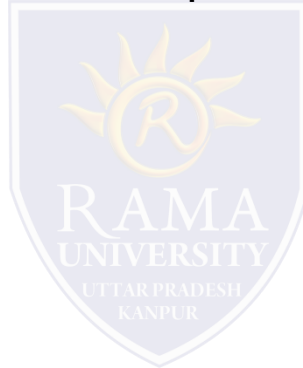
Best case

The best case is when the array is already sorted in the right order.

The number of comparisons is almost zero and swapping.

Average case Less comparison and swamping

Worst case More comparison and swapping



# Advantages

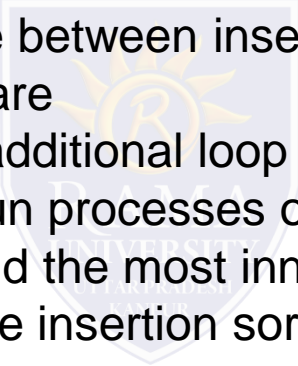
1. Shell sort is easy to implement but not easier than insertion sort.
2. Shell sort can be sorted for a gap greater than one and thus less exchange than insertion sort hence, fast.
3. Efficient for medium size lists.
4. It is easy to understand.
5. It is easy to implement.

DisAv.

It is a complex algorithm and its not nearly as efficient as the merge, heap and quick sorts.

2.

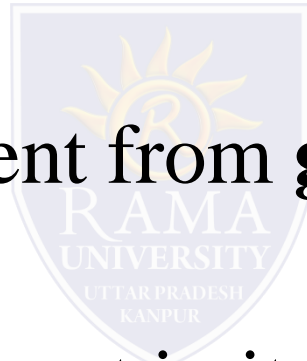




The difference between insertion sort from the shell sort are  
There is one additional loop (the outer loop)- each run processes one increment.  
The middle and the most inner loops are same as in the insertion sort with  $gap = 1$ .

# Shell Sort Algorithm

1. Set **gap** to  $n/2$
2. while **gap**  $> 0$
3.     for each element from **gap** to end, by  
       **gap**
4.     Insert element in its **gap**-separated sub-  
array
5.     if **gap** is 2, set it to 1
6.     otherwise set it to **gap** / 2.2



## Shell Sort Algorithm: Inner Loop

- set **nextPos** to position of element to insert
- 2. set **nextVal** to value of that element
- 3. while **nextPos** > **gap** and  
3.4 element at **nextPos-gap** is > **nextVal**  
Shift element at **nextPos-gap** to  
**nextPos**
- 3.5 Decrement **nextPos** by **gap**
- 3.6 Insert **nextVal** at **nextPos**