Design and Analysis of Algorithm

Linear Time Sorting
(Shell Sort)

Lecture -20

Overview

- Running time of Shell sort in worst case is $O(n^2)$ or float between $O(n \log n)$ *and* $O(n^2)$.
- Running time of Shell sort in best case is $\mathrm{O}(n \lg n)$. And $O(n\ )$ if the total number of comparisons for each interval (or increment) is equal to the size of the array.
- Is not a stable sorting.

Shell Sort

- Designed by Donald Shell and named the sorting algorithm after himself in 1959.
- Shell sort works by comparing elements that are distant rather than adjacent elements in an array or list where adjacent elements are compared.
- Shell sort is also known as diminishing increment sort.

Shell Sort

- Shell sort improves on the efficiency of **insertion sort** by quickly shifting values to their destination.
- This algorithm tries to decreases the distance between comparisons (i.e. gap) as the sorting algorithm runs and reach to its last phase where, the adjacent elements are compared only.

Shell Sort

- The distance of comparisons (i.e. gap) is maintained by the following methods:
  - divide by 2(Two) [Designed by Donaled Shell)
  - Knuth Method$(i.e. gap \leftarrow gap * 3 + 1)$ $(initially\ the\ gap\ start\ with\ 1)$

Shell Sort

- Let's execute an example with the help of Knuth's gap method on the following array.

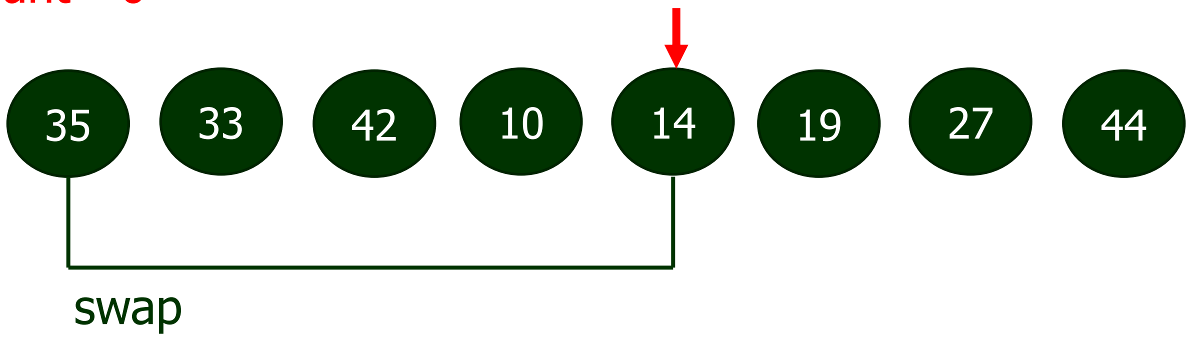
35	33	42	10	14	19	27	44

- At the beginning the gap is initialized as 1
- Hence the new gap value for iteration 1 is calculated as follows:

$$
\begin{aligned}
gap &= gap * 3 + 1 \\
&= 1 * 3 + 1 = 4
\end{aligned}
$$

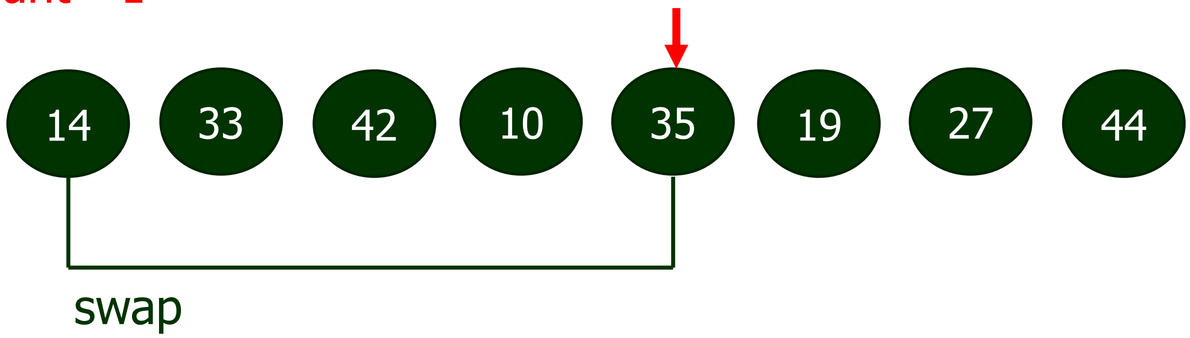
Shell Sort

Swap count =0



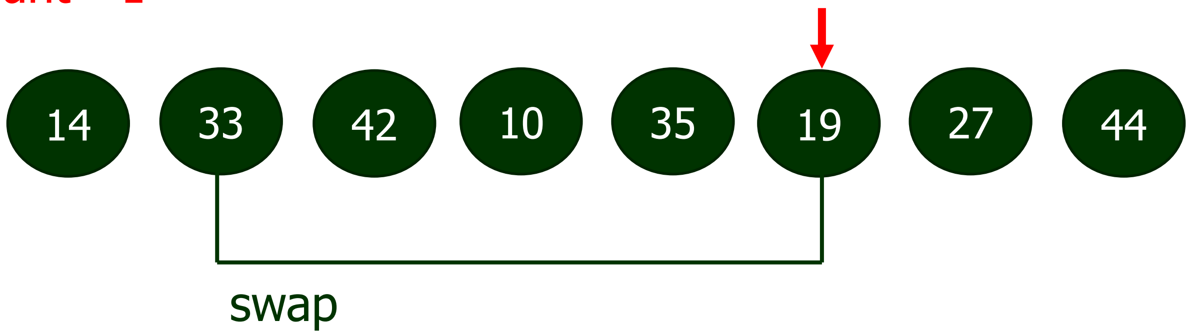
Shell Sort

Swap count =1



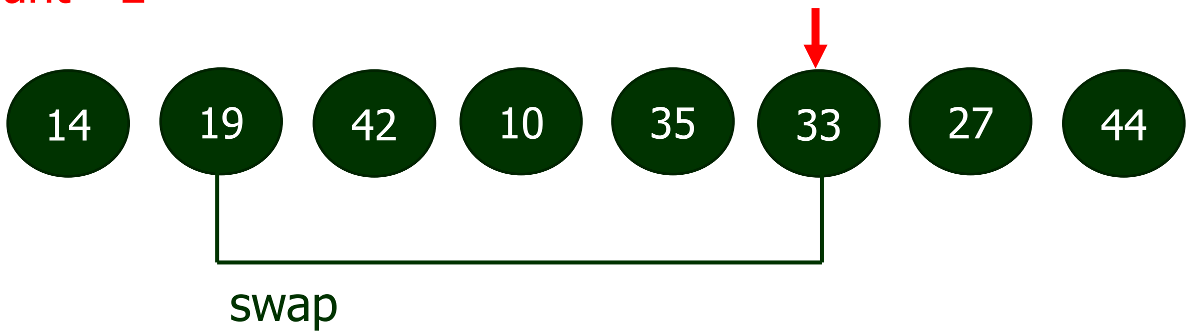
Shell Sort

Swap count =1



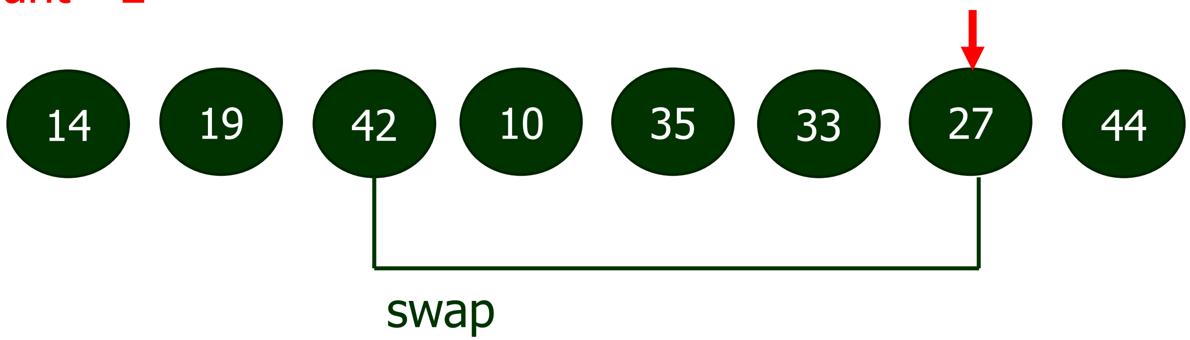
Shell Sort

Swap count =2



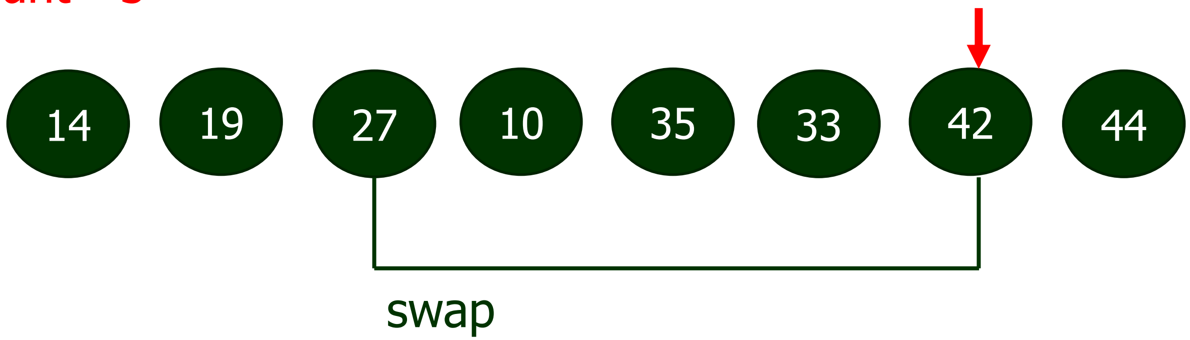
Shell Sort

Swap count =2



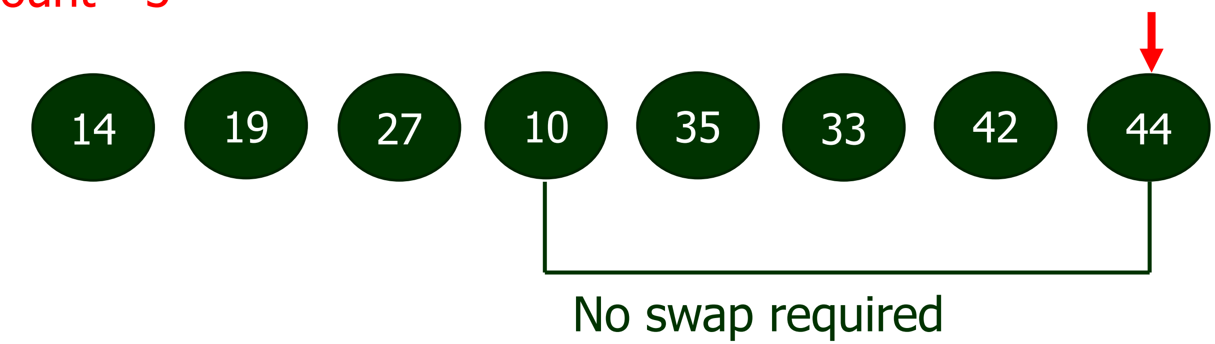
Shell Sort

Swap count =3



Shell Sort

Swap count =3



Shell Sort

- After the first iteration the array looks like as follows.

14	19	27	10	35	33	42	44

- Again we finding the gap value for nest iteration.

$$gap = gap * 3 + 1$$

- We can write the above equation as follows:

$$gap = \frac{gap \ - 1}{3}$$

- So, the new gap is $gap = \frac{gap\ -1}{3} = \frac{4\ -1}{3} = 1$

Shell Sort

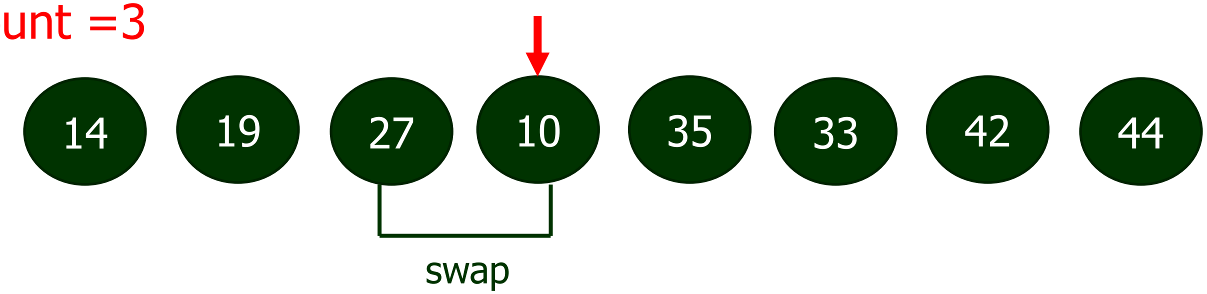
Swap count =3

Shell Sort

Swap count =3

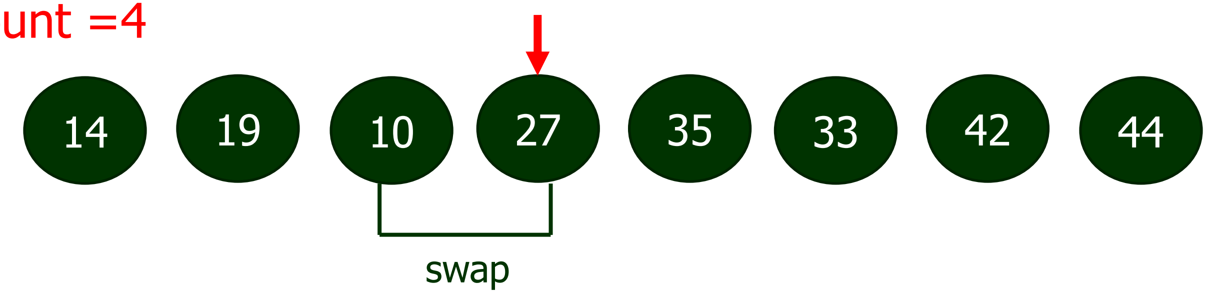
Shell Sort

Swap count =3



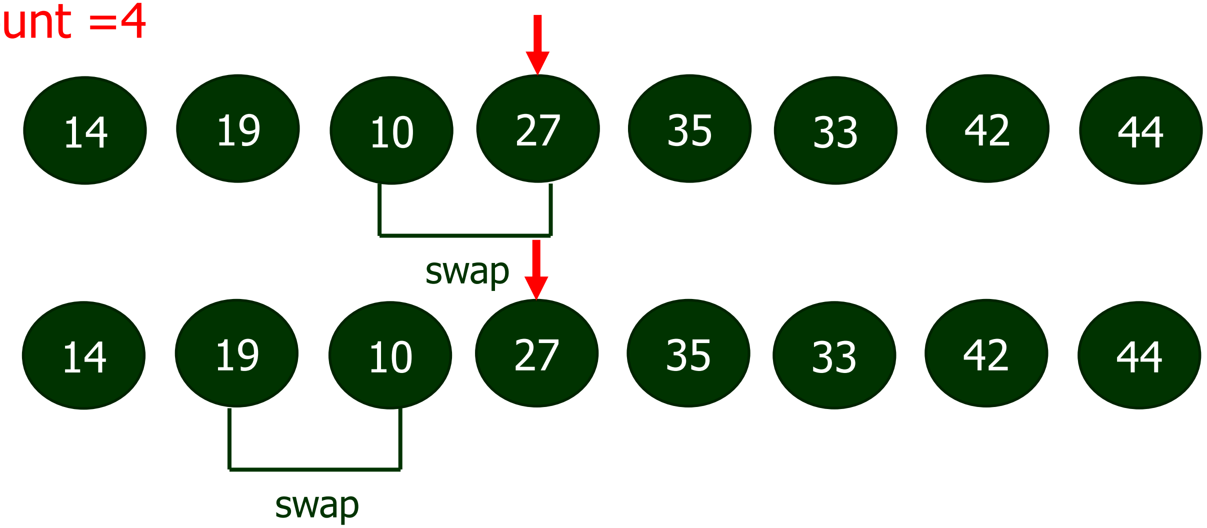
Shell Sort

Swap count =4



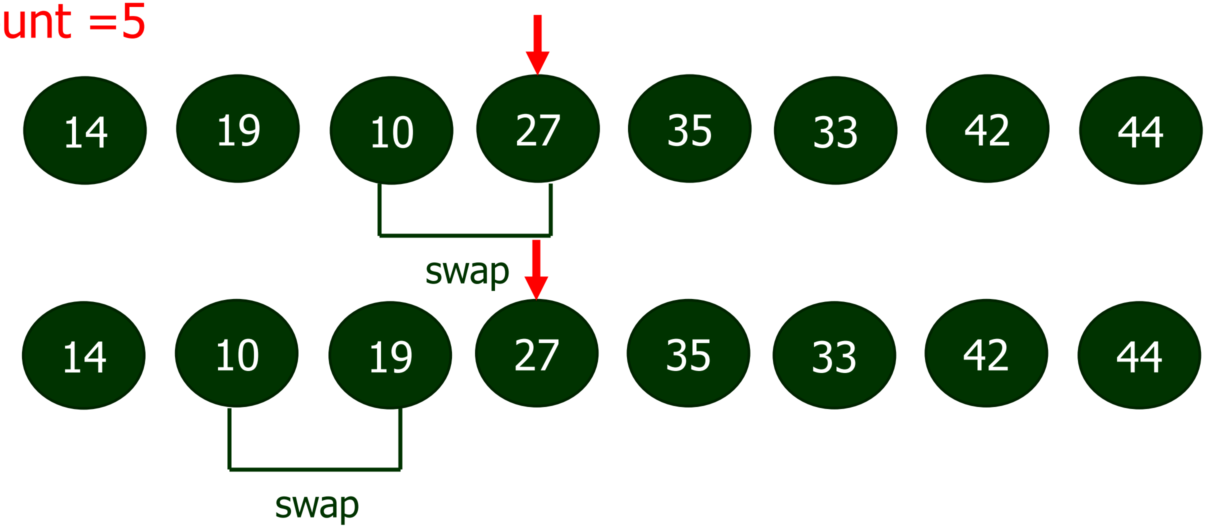
Shell Sort

Swap count =4



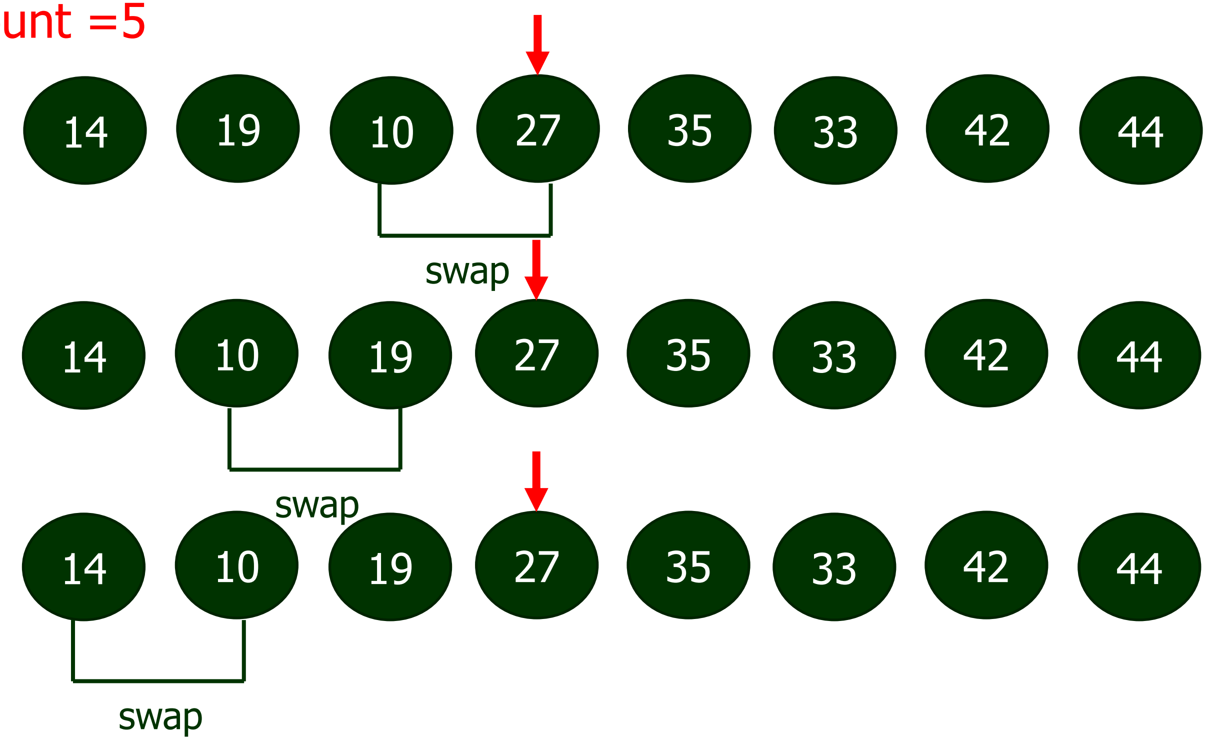
Shell Sort

Swap count =5



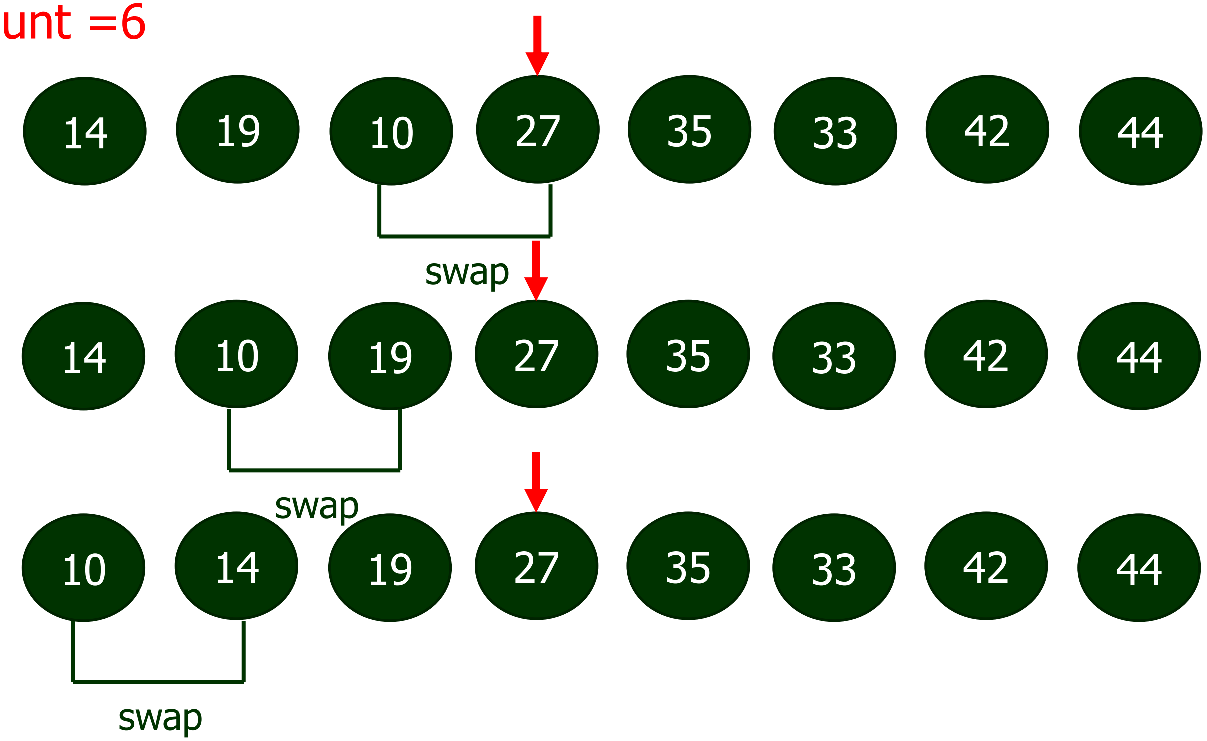
Shell Sort

Swap count =5



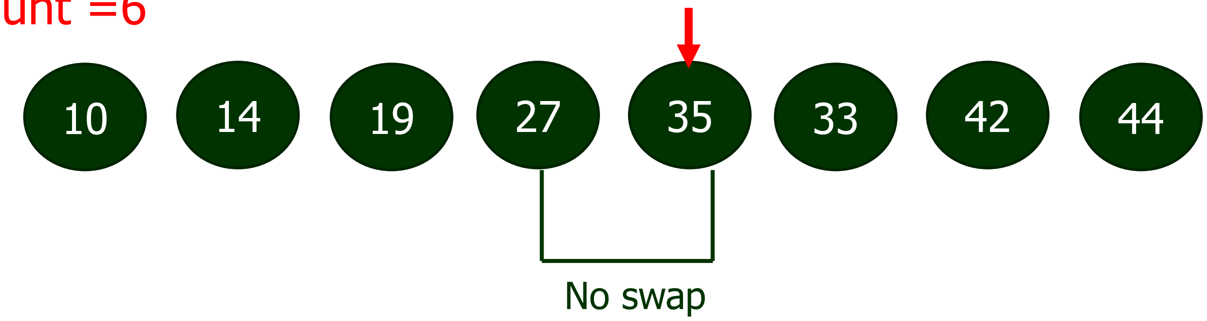
Shell Sort

Swap count =6



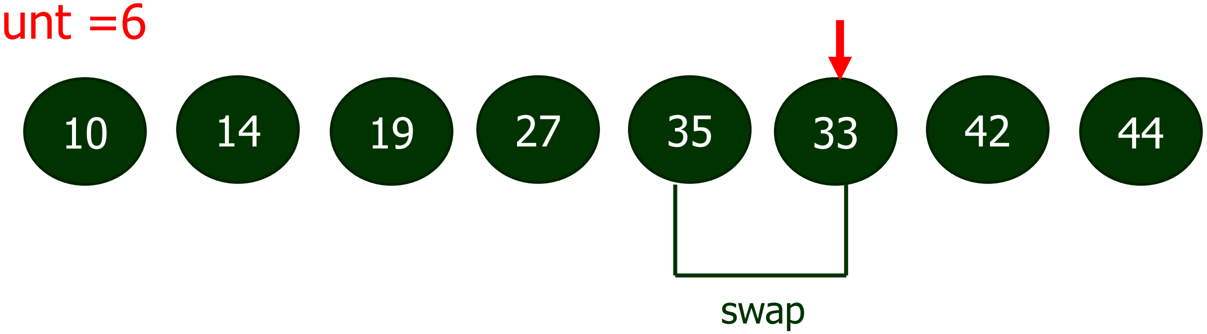
Shell Sort

Swap count =6



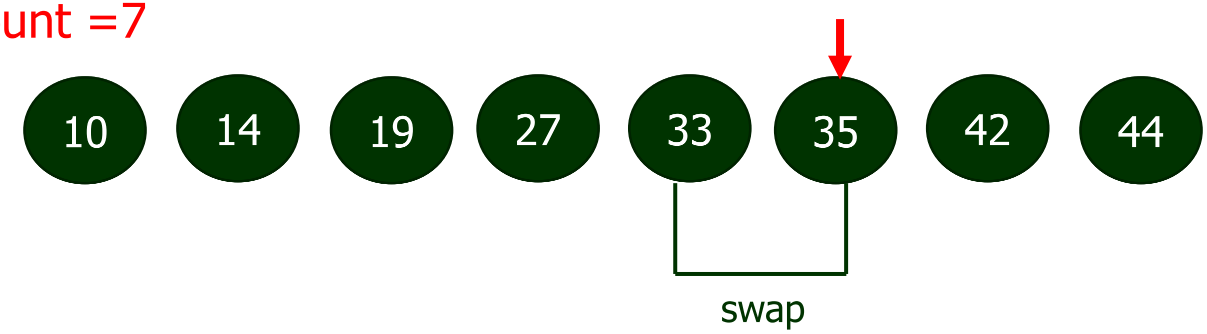
Shell Sort

Swap count =6



Shell Sort

Swap count =7



Shell Sort

Swap count =7

Shell Sort

Swap count =7

Shell Sort

Swap count =7

- Hence ,total number of swap required in
  - 1st iteration= 3
  - 2nd iteration= 4
- So total 7 numbers of swap required to sort the array by shell sort.

Shell Sort

<u>Algorithm Shell sort (Knuth Method)</u>

```
gap=1
while(gap < A.length/3)
    gap=gap*3+1
while( gap>0)
    for(outer=gap; outer<A.length; outer++)
        Ins_value=A[outer]
        inner=outer
        while(inner>gap-1 && A[inner-gap]≥ Ins_value)
            A[inner]=A[inner-gap]
            inner=inner-gap
        A[inner]=Ins_value
    gap=(gap-1)/3
```

Shell Sort

- Let us dry run the shell sort algorithm with the same example as already discussed.

35 33 42 10 14 19 27 44

At the beginning

A .length=8 and gap=1

After first three line execution the gap value changed to 4

Now, gap>0 (i.e. 4>0)

Now in for loop outer=4;outer<8;outer++

Ins_value=A[outer]=A[4]=14

inner=outer i.e. inner=4

Now the line no 8 is true$\Longrightarrow$ *change occured* and the updated array is looked as follow

14 33 42 10 35 19 27 44

Swap

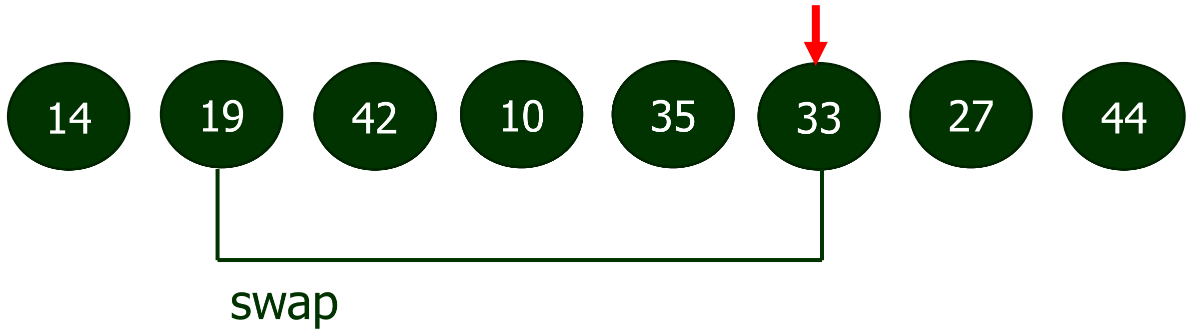
Shell Sort

Now in for loop outer=5 ;outer<8; outer++

Ins_value=A[outer]=A[5]=19

inner=outer i.e. inner=5

Now the line no 8 is true$\Longrightarrow$ *change occured* and the updated array is looked as follow



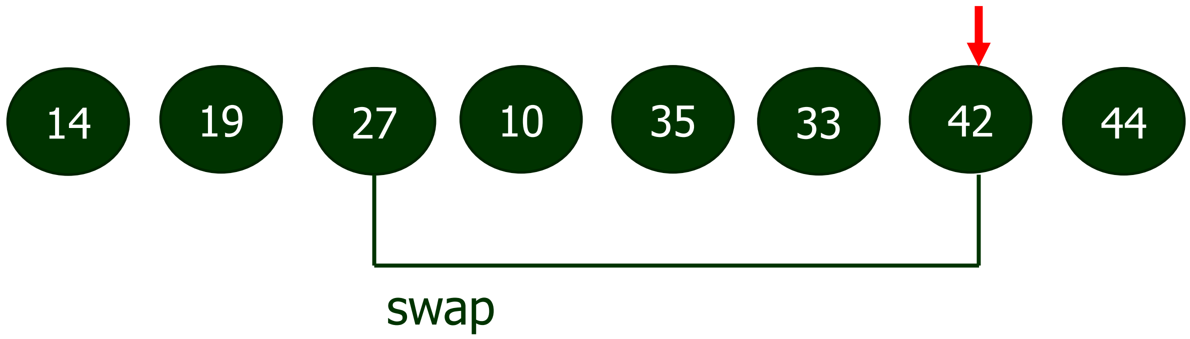
Shell Sort

Now in for loop outer=6 ;outer<8; outer++

Ins_value=A[outer]=A[6]=27

inner=outer i.e. inner=6

Now the line no 8 is true$\Longrightarrow$ *change occured* and the updated array is looked as follow



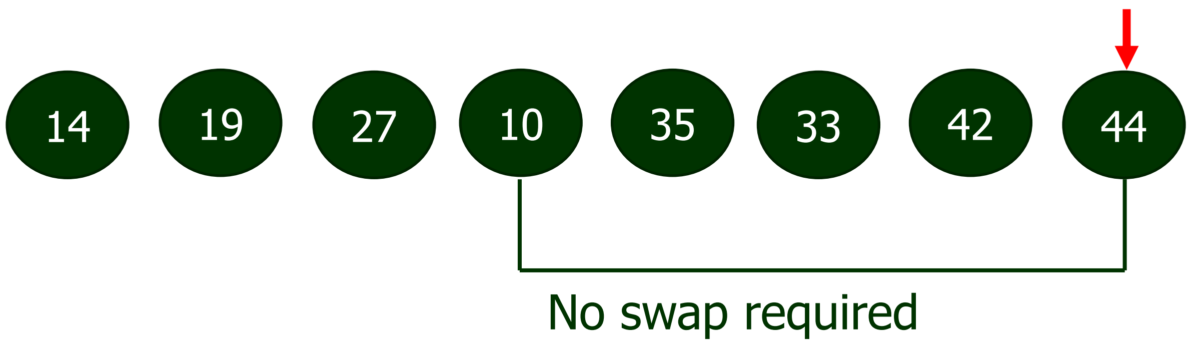
Shell Sort

Now in for loop outer=7 ;outer<8; outer++

Ins_value=A[outer]=A[7]=44

inner=outer i.e. inner=7

Now the line no 8 is False ⟹ no change in array and the updated array is looked as follow



Shell Sort

- Now again gap value will be calculated .
- The new gap value is 1. And again the same procedure will be continued .

Shell Sort

- Now again gap value will be calculated .
- The new gap value is 1. And again the same procedure will be continued .

Home Assignment

Shell Sort

- Now again gap value will be calculated .
- The new gap value is 1. And again the same procedure will be continued . And finally the sorted array looks as given below with 7(seven) number of swap

10	14	19	27	33	35	42	44

Shell Sort

Analysis:

- Shell sort is efficient for medium size lists.
- For bigger list, this algorithm is not the best choice.
- But it is the fastest of all $O(n^2)$ sorting algorithm.
- The best case in shell sort is when the array is already sorted in the right order i.e. $O(n)$
- The worst case time complexity is based on the gap sequence. That's why various scientist give their gap intervals. They are:
  1. Donald Shell give the gap interval ${}^{n}/_{2}$. $\Longrightarrow O(n \log n)$
  2. Knuth give the gap interval $gap \leftarrow gap * 3 + 1 \quad \Longrightarrow O(n^{3/2})$
  3. Hibbard give the gap interval $2^{k-1} \Longrightarrow O(n \log n)$

Shell Sort

Analysis:

In General

- Shell sort is an unstable sorting algorithm because this algorithm does not examine the elements lying in between the intervals.
- **Worst Case Complexity:** less than or equal to $O(n^2)$ or float between $O(n \log n)$ and $O(n^2)$.
- **Best Case Complexity:** $O(n \log n)$
  When the array is already sorted, the total number of comparisons for each interval (or increment) is equal to $O(n)$ i.e. the size of the array.
- Average Case Complexity: $O(n \log n)$
  It is around $O(n^{1.25})$.

Shell Sort

Analysis:

In General

- Shell sort is an unstable sorting algorithm because this algorithm does not examine the elements lying in between the intervals.
- **Worst Case Complexity:** less than or equal to $O(n^2)$ or float between $O(n \log n)$ *and* $O(n^2)$.
- **Best Case Complexity:** $O(n \log n)$
  When the array is already sorted, the total number of comparisons for each interval (or increment) is equal to $O(n\ )$ i.e. the size of the array.
- Average Case Complexity: $O(n \log n)$
  It is around $O(n^{1.25})$.

(Remark: Accurate model not yet been discovered)

Thank u