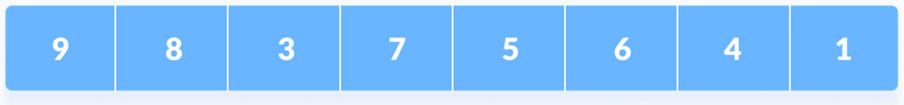
Shell Sort Algorithm

Shell sort is an algorithm that first sorts the elements far apart from each other and successively reduces the interval between the elements to be sorted. It is a generalized version of insertion sort.

In shell sort, elements at a specific interval are sorted. The interval between the elements is gradually decreased based on the sequence used. The performance of the shell sort depends on the type of sequence used for a given input array.

How Shell Sort Works?

1. Suppose, we need to sort the following array.



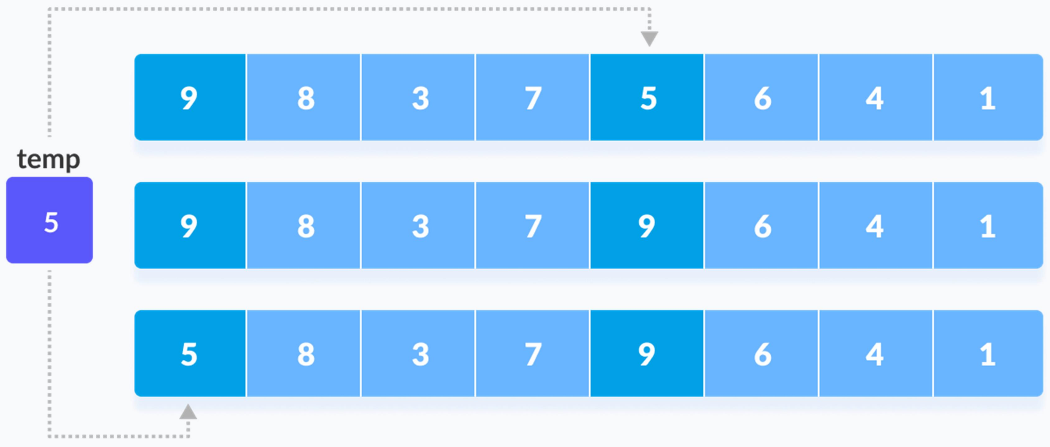
Initial array

2. We are using the shell's original sequence `(N/2, N/4, ...1)` as intervals in our algorithm.

   In the first loop, if the array size is `N = 8` then, the elements lying at the interval of `N/2 = 4` are compared and swapped if they are not in order.

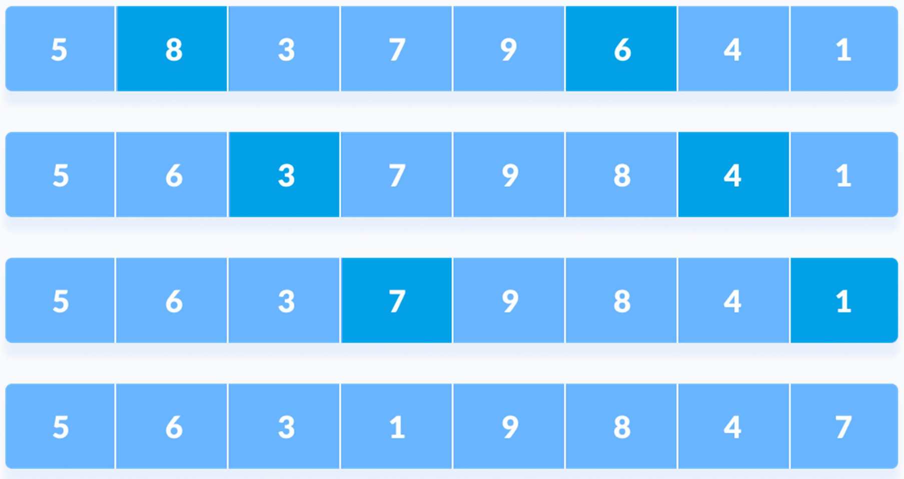
a. The 0th element is compared with the `4th` element.

b. If the 0th element is greater than the `4th` one then, the `4th` element is first stored in `temp` variable and the `0th` element (ie. greater element) is stored in the `4th` position and the element stored in `temp` is stored in the `0th` position.



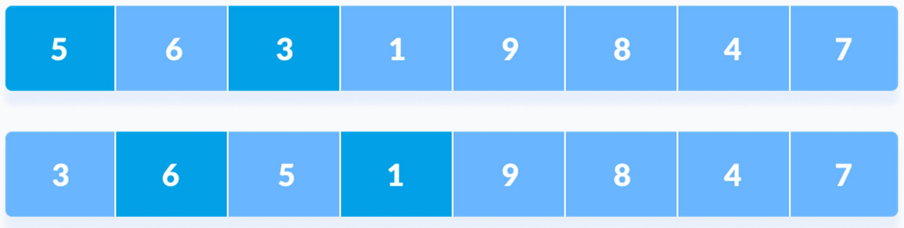
Rearrange the elements at n/2 interval

This process goes on for all the remaining elements.



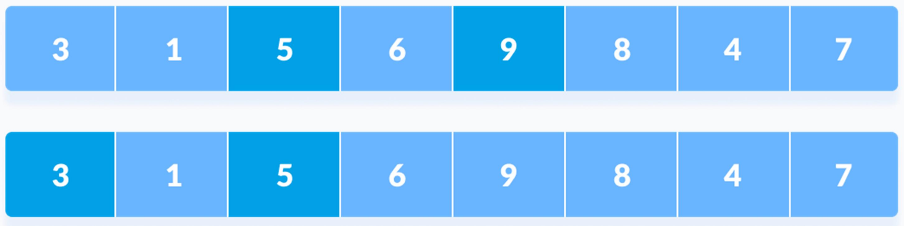
Rearrange all the elements at n/2 interval

3. In the second loop, an interval of `N/4 = 8/4 = 2` is taken and again the elements lying at these intervals are sorted.



Rearrange the elements at n/4 interval

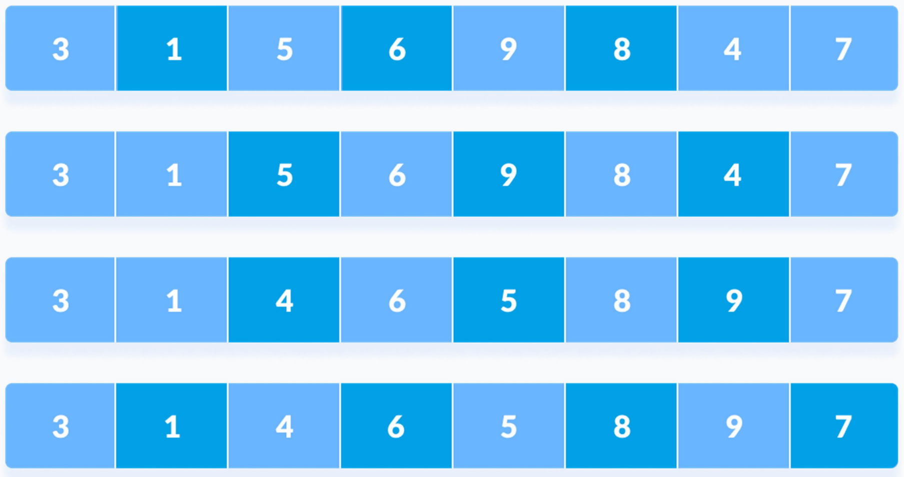
You might get confused at this point.



All the elements in the array lying at the current interval are compared.

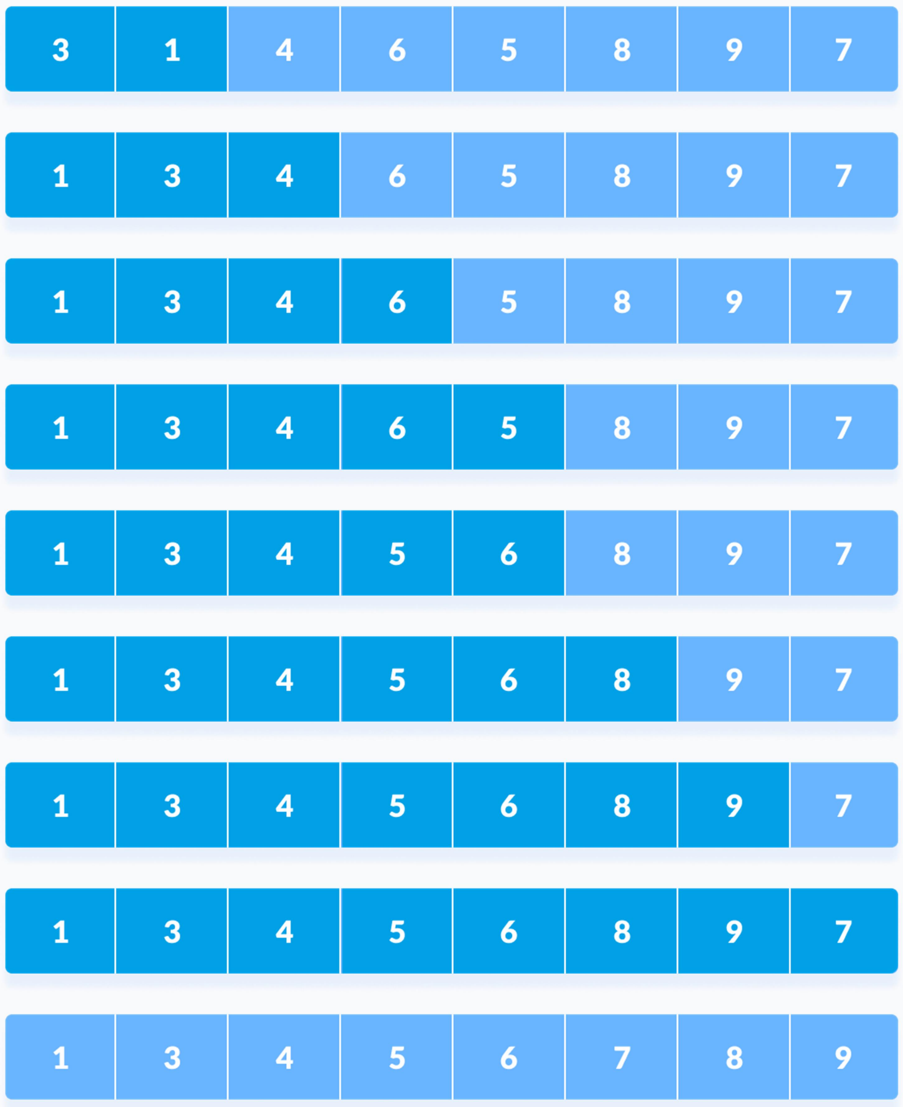
The elements at `4th` and `2nd` position are compared. The elements at `2nd` and `0th` position are also compared. All the elements in the array lying at the current interval are compared.

4. The same process goes on for remaining elements.



Rearrange all the elements at n/4 interval

5. Finally, when the interval is `N/8 = 8/8 =1` then the array elements lying at the interval of 1 are sorted. The array is now completely sorted.



Rearrange the elements at n/8 interval

Shell Sort Algorithm

```
for(gap=n/2; gap>=1; gap/2)
{
  for(j=gap; j<n; j++)
  {
    for(i=j-gap;i>=0;i-gap)
    {
      if(a[i+gap] > a[i])
      {
        break;
       }
       else
       {
       swap(a[i+gap], a[i]);
        }
    }
  }
}
```

```
Example: 23 29 15 19 31 7 9 5 2

n=9

gap=n/2=9/2=4

j=gap=4

i=j-gap=4-4=0

a[i+gap] > a[i]

a[0+4]   >  a[o]

31  >  23  true

Break;


j=5

i=5-4=1

a[i+gap] > a[i]

a[1+4] > a[1]

a[5] > a[1]

7  >  29 false

else

swap(7,29)
```

```
23 7 15 19 31 29 9 5 2


i=i-gap=-3
j=6
i=j-gap=6-4=2
a[i+gap]>a[i]
a[2+4] > a[2]
9 > 15 false
else
swap(9,15)
23 7 9 19 31 29 15 5 2


I=i-gap=2-4=-2
J=7
I=7-4=3
a[i+gap] > a[i]
a[3+4] > a[3]
5 > 19 false
```

```
Else

Swap(5,19)

23 7 9 5 31 29 15 19 2


j=8

i=j-gap=8-4=4

a[i+gap]>a[i]

a[4+4]>a[4]

2>31 false

Else

Swap(2,31)

23 7 9 5 2 29 15 19 31

i=i-gap=4-4=0

a[i+gap]>a[i]

a[0+4]>a[0]

2>23 false

Swap(2,23)

2 7 9 5 23 29 15 19 31
```

Complexity

Shell sort is an unstable sorting algorithm because this algorithm does not examine the elements lying in between the intervals.

Time Complexity

- **Worst Case Complexity**: less than or equal to `O(n²)`
  Worst case complexity for shell sort is always less than or equal to `O(n²)`.

  According to Poonen Theorem, worst case complexity for shell sort is `Θ(Nlog N)²/(log log N)²)` or `Θ(Nlog N)²/log log N)` or `Θ(N(log N)²)` or something in between.
- **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 the size of the array.
- **Average Case Complexity**: `O(n*log n)`
  It is around `O(n1.25)`.

The complexity depends on the interval chosen. The above complexities differ for different increment sequences chosen. Best increment sequence is unknown.

Space Complexity:

The space complexity for shell sort is `O(1)`.