Shell Sort:

Shell sort is mainly a variation of 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 idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

Algorithm:

Step 1 – Start

Step 2 – Initialize the value of gap size. Example: h

Step 3 – Divide the list into smaller sub-part. Each must have equal intervals to h

Step 4 – Sort these sub-lists using insertion sort

Step 5 – Repeat this step 2 until the list is sorted.
Step 6 – Print a sorted list.
Step 7 – Stop.

Shell Sort Algorithm

```
shellSort(array, size)
  for interval i <- size/2n down to 1
    for each interval "i" in array
        sort all the elements at interval "i"
end shellSort
```

```
// Shell Sort in C programming

#include <stdio.h>

// Shell sort
void shellSort(int array[], int n) {
  // Rearrange elements at each n/2, n/4, n/8, ... intervals
  for (int interval = n / 2; interval > 0; interval /= 2) {
    for (int i = interval; i < n; i += 1) {
      int temp = array[i];
      int j;
      for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {
        array[j] = array[j - interval];
      }
```

```
      array[j] = temp;
    }
  }
}

// Print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

// Driver code
int main() {
  int data[] = {9, 8, 3, 7, 5, 6, 4, 1};
  int size = sizeof(data) / sizeof(data[0]);
  shellSort(data, size);
  printf("Sorted array: \n");
  printArray(data, size);
}
```

Shell Sort Complexity

	Time Complexity
Best	O(nlog n)
Worst	$O(n^2)$
Average	O(nlog n)
Space Complexity	O(1)
Stability	No

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 $O(n^2)$

  Worst case complexity for shell sort is always less than or equal to $O(n^2)$.

  According to Poonen Theorem, worst case complexity for shell sort is $\theta(N\log N)^2/(\log\log N)^2)$ or $\theta(N\log N)^2/\log\log N)$ or $\theta(N(\log N)^2)$ 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 $O(n^{1.25})$.

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 $O(1)$.

Shell Sort Applications

Shell sort is used when:

- calling a stack is overhead. `uClibc` library uses this sort.
- recursion exceeds a limit. `bzip2` compressor uses it.
- Insertion sort does not perform well when the close elements are far apart. Shell sort helps in reducing the distance between the close elements. Thus, there will be less number of swappings to be performed.