Northeastern University

Lecture 27: An Introduction to Sorting - 3

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5

Shell Sort

Donald L. Shell
(March 1, 1924 – November 2, 2015)

Observation from Insertion Sort

- An array entry moves to an adjacent location.
- When an entry is far from its correct sorted position, it must make many such moves.
- When an array is completely scrambled, an insertion sort takes a good deal (a lot) of time.
- When an array is almost sorted, an insertion sort is more efficient.

Shell Sort

- Algorithms seen so far are simple but inefficient for large arrays at $O(n^2)$.
- Note, the more sorted an array is, the less work `insertInOrder` must do.
- Improved insertion sort developed by Donald Shell.

Shell Sort (cont.)

- Move entries beyond their adjacent locations.
- Use Insertion Sort repeatedly
- Steps:
  - Create subarrays of entries at equally spaced indices
  - Run Insertion Sort to each subarray
  - Merge subarrays
  - Reduce space to half and repeat above steps until space becomes 1

Shell Sort

0	1	2	3	4	5	6	7	8	9	10	11	12
10	16	11	4	15	3	9	6	1	17	8	12	7

Before Ordering

10 ---- 9 ---- 7
16 ---- 6
11 ---- 1
4 ---- 17
15 ---- 8
3 ---- 12

After Ordering

7 ---- 9 ---- 10
6 ---- 16
1 ---- 11
4 ---- 17
8 ---- 15
3 ---- 12

7	6	1	4	8	3	9	16	11	17	15	12	10
0	1	2	3	4	5	6	7	8	9	10	11	12

Figure 15-11, 15-12: An array and the subarrays formed by grouping entries whose indices are 6 apart.

Shell Sort

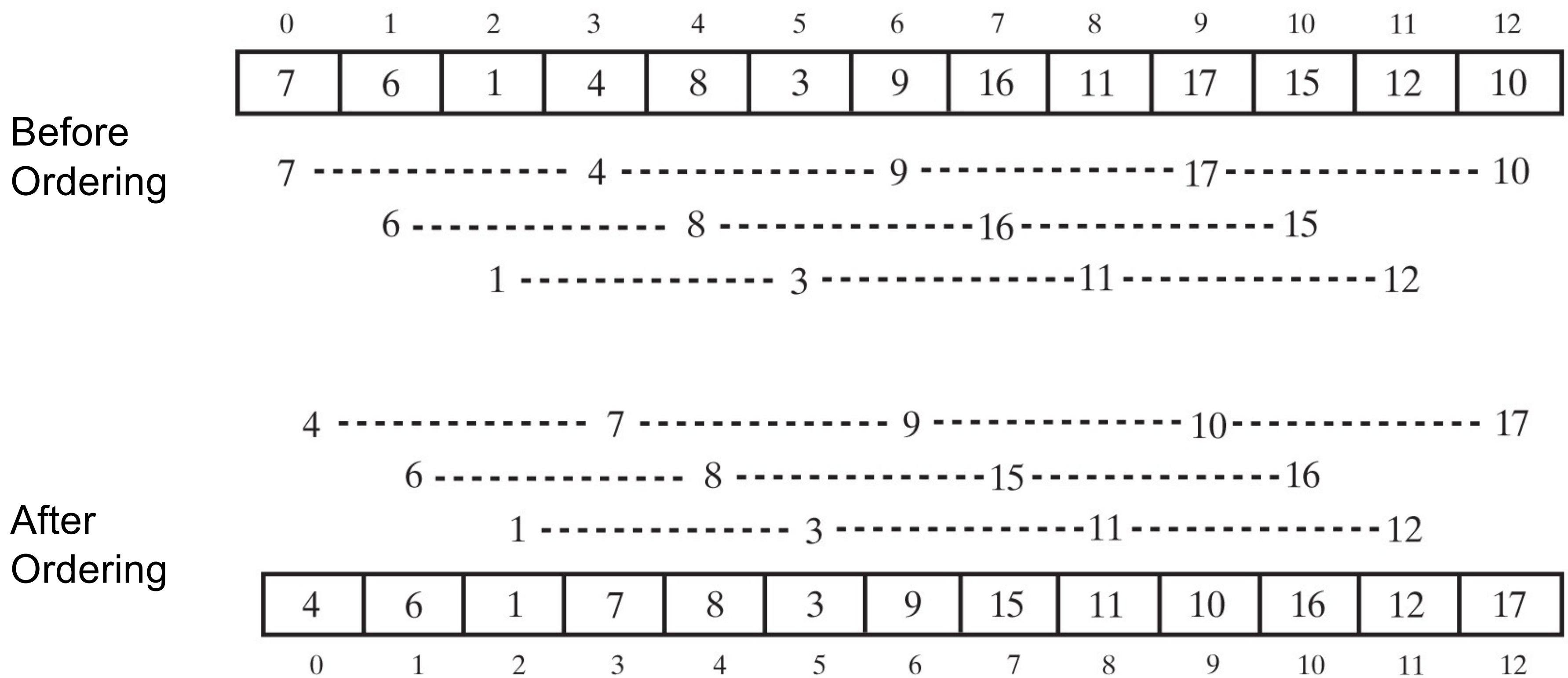

Figure 15-13, 15-14: Grouped entries in the array in Figure 15-12 whose indices are 3 apart

Shell Sort

```
Algorithm shellSort(a, first, last)
// Sorts the entries of an array a[first..last] into ascending order.
// first >= 0 and < a.length; last >= first and < a.length.

   n = number of array entries
   space = n / 2
   while (space > 0)
   {
      for (begin = first through first + space - 1)
      {
         incrementalInsertionSort(a, begin, last, space)
      }
      space = space / 2
   }
```

Algorithm to perform a Shell sort will invoke `incrementalInsertionSort` and supply any sequence of spacing factors. Efficiency (worst) can be $O(n^{1.5})$

Shell Sort

```
Algorithm incrementalInsertionSort(a, first, last, space)
// Sorts equally spaced entries of an array a[first..last] into ascending order.
// first >= 0 and < a.length; last >= first and < a.length;
// space is the difference between the indices of the entries to sort.

   for (unsorted = first + space through last at increments of space)
   {
      nextToInsert = a[unsorted]
      index = unsorted - space
      while ( (index >= first) and (nextToInsert.compareTo(a[index]) < 0) )
      {
         a[index + space] = a[index]
         index = index - space
      }
      a[index + space] = nextToInsert
   }
```

Algorithm that sorts array entries whose indices are separated by an increment of `space`.

Exercise (L27_E1)

- Please implement a method `shellSort()` that sorts an array of integers by using Shell Sort algorithm.

- The array has 30 random generated integers
- The `shellSort()` takes `int[]` as a parameter, i.e., `shellSort(int[] a)` and return `void`.

Answer

```java
public class MyShellSort {

  public static void main(String[] args){
    int size = 30;
    int[] data = new int[size];
    Random generator = new Random();

    for(int i = 0; i < size; i++){
      data[i] = generator.nextInt(size);
    }

    shellSort(data);
    System.out.println(Arrays.toString(data));
  }

  public static void shellSort(int[] a) {
    int increment = a.length / 2;
    while (increment > 0) {
      for (int i = increment; i < a.length; i++) {
        int j = i;
        int temp = a[i];
        while (j >= increment && a[j - increment] > temp) {
          a[j] = a[j - increment];
          j = j - increment;
        }
        a[j] = temp;
      }
      increment = increment / 2;
    }
  }

}
```

Comparing the Algorithms

	Best Case	Average Case	Worst Case
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Shell Sort	$O(n)$	$O(n^{1.5})$	$O(n^{1.5})$

The time efficiencies of three sorting algorithms, expressed in Big Oh notation

[1] The Complexity of Shellsort: https://www.baeldung.com/cs/shellsort-complexity