Insertion Sort and Shell Sort

CS 260P: Fundamentals of Algorithms With Applications

Michael T. Goodrich

UNIVERSITY of CALIFORNIA IRVINE

Some slides are from J. Miller, CSE 373, U. Washington

Insertion sort

- **insertion sort**: orders a list of values by repetitively inserting a particular value into a sorted subset of the list

- more specifically:
  - consider the first item to be a sorted sublist of length 1
  - insert the second item into the sorted sublist, shifting the first item if needed
  - insert the third item into the sorted sublist, shifting the other items as needed
  - repeat until all values have been inserted into their proper positions

Insertion sort

- Simple sorting algorithm.
  - n-1 passes over the array
  - At the end of pass *i*, the elements that occupied A[0]…A[*i*] originally are still in those spots and in sorted order.

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

after pass 2

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

after pass 3

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

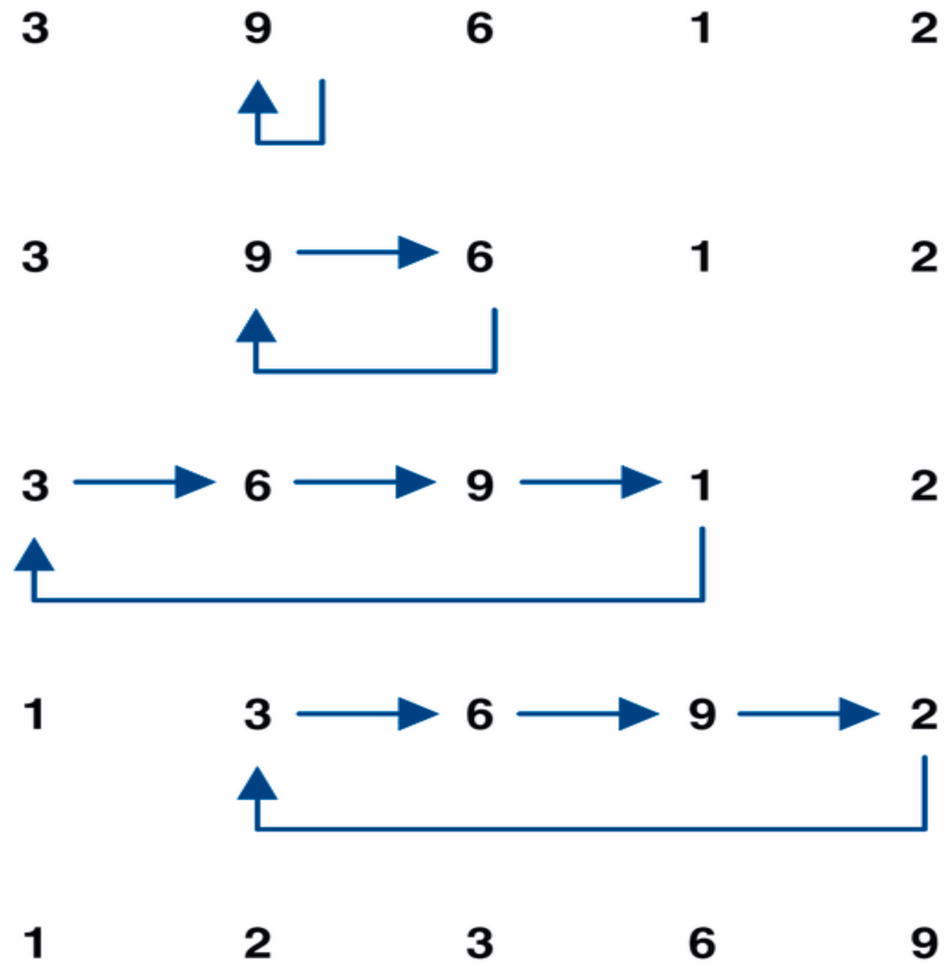
Insertion sort example

3 is sorted.
Shift nothing. Insert 9.

3 and 9 are sorted.
Shift 9 to the right. Insert 6.

3, 6, and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 1.

1, 3, 6, and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 2.



Insertion sort code

```
public static void insertionSort(int[] a) {
    for (int i = 1; i < a.length; i++) {
        int temp = a[i];

        // slide elements down to make room for a[i]
        int j = i;
        while (j > 0 && a[j - 1] > temp) {
            a[j] = a[j - 1];
            j--;
        }

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

Analysis of Insertion Sort

- In the worst case, we spend O(n) in each iteration (to slide element to its place). So worst-case running time is $O(n^2)$.
- Each time we slide an element, we swap two elements that were out of order.
- If K is the number of out-of-order pairs, then running time actually is O(n+K).

Shell sort description

- **shell sort**: orders a list of values by comparing elements that are separated by a gap of >1 indexes
  - a generalization of insertion sort
  - invented by computer scientist Donald Shell in 1959

- based on some observations about insertion sort:
  - insertion sort runs fast if the input is almost sorted
  - insertion sort's weakness is that it swaps each element just one step at a time, taking many swaps to get the element into its correct position

Shell sort example

- Idea: Sort all elements that are 5 indexes apart, then sort all elements that are 3 indexes apart, ...

Original	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
After 5-sort	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 swaps
After 3-sort	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 swaps
After 1-sort	16	19	24	32	35	40	43	54	66	68	72	82	93	95	15 swaps

Shell sort code

```
public static void shellSort(int[] a) {
    for (int gap = a.length / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < a.length; i++) {
            // slide element i back by gap indexes
            // until it's "in order"
            int temp = a[i];
            int j = i;
            while (j >= gap && temp < a[j - gap]) {
                a[j] = a[j - gap];
                j -= gap;
            }
            a[j] = temp;
        }
    }
}
```

Analysis of Shell sort

- The worst-case running time depends on the gap sequence.
  - $N/2^k$: $O(n^2)$ time
  - 2^k-1: $O(n^{3/2})$ time
  - 2^j3^k: $O(n \log^2 n)$ time

- Other gap sequences might be even better…

Experimental Analysis

- Has never been done for all possible gap sequences.
- Even known gap sequences might have different real-world performance.

- That is where Project 1 comes in…