Elementary Sorts

Neerja Mhaskar

Dept. of Computing and Software, McMaster University, Canada

Acknowledgments: Material mainly based on the textbook Algorithms by Robert Sedgewick and Kevin Wayne (Chapters 2.1) and Prof. Janicki's course slides

Selection Sort

- Scans items from left to right.
- At iteration i, find the index position j of the smallest item in the remaining entries.
- Swap the item at i and j.

ALGORITHM 2.1 Selection sort

```
public class Selection
{
   public static void sort(Comparable[] a)
   {  // Sort a[] into increasing order.
      int N = a.length;              // array length
      for (int i = 0; i < N; i++)
      {  // Exchange a[i] with smallest entry in a[i+1...N).
         int min = i;                // index of minimal entr.
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min])) min = j;
         exch(a, i, min);
      }
   }
   // See page 245 for less(), exch(), isSorted(), and main().
}
```

See Demo - `https://algs4.cs.princeton.edu/lectures/`

Selection Sort - Example

a[]

i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Trace of selection sort (array contents just after each exchange)

Selection Sort - Analysis

Selection sort uses $(N-1)+(N-2)+\ldots+0$ compares and N exchanges. Therefore, the running time of the algorithm $T(N)=N^2+n=\Theta(N^2)$.

- Running time insensitive to input.
- Quadratic time, even if input is sorted.
- Data movement is minimal.
- Linear number of exchanges.

Insertion Sort

- Scan items from left to right.
- At iteration i, swap $a[i]$ with each larger entry to its left.

ALGORITHM 2.2 Insertion sort

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {  // Sort a[] into increasing order.
      int N = a.length;
      for (int i = 1; i < N; i++)
      {  // Insert a[i] among a[i-1], a[i-2], a[i-3]... ..
         for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
      }
   }
   // See page 245 for less(), exch(), isSorted(), and main().
}
```

See Demo - `https://algs4.cs.princeton.edu/lectures/`

Insertion Sort - Example

i	j	a[] 0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in gray do not move

entry in red is a[j]

entries in black moved one position right for insertion

Trace of insertion sort (array contents just after each insertion)

Insertion Sort - Analysis

- Best case: If the array is in order (or nearly in order), insertion sort makes $N - 1$ compares and 0 exchanges. Therefore, $T(N) = \Omega(N)$

  Ex: 1 2 3 4 5 6 7 8 9

- Worst case: If the array is in reverse order (or nearly in reverse order) insertion sort makes $\frac{N^2-N}{2}$ compares and $\frac{N^2-N}{2}$ exchanges, i.e., $T(N) = O(N^2)$.

  Ex. 9 8 7 6 5 4 3 2 1

Shellsort

- Shellsort is an improvement over Insertion sort.
- Insertion Sort Problem: Slow for large unordered arrays
  - as exchanges involve adjacent entries only,
  - so items move through the array only one place at a time
- Shellsort allowing exchanges of array entries that are far apart, to produce partially sorted arrays that can be efficiently sorted, eventually by insertion sort.

h-Sort - See Demo

- In iteration i, swap $a[i]$ with each larger entry h positions to its left.
- Therefore, h-sort is Insertion sort, with stride length h.

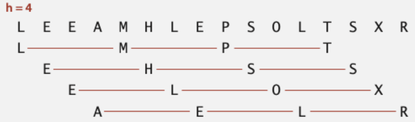
3-sorting an array

M	O	L	E	E	X	A	S	P	R	T
E	O	L	M	E	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T

h-Sort

Idea: Move entries more than one position at a time by h-sorting the array.

an h–sorted array is h interleaved sorted subsequences



Shellsort: h-sort array for decreasing sequence of values of h.

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
1-sort	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

Shellsort example: h values 7, 3, 1

input

S O R T E X A M P L E

7-sort

S	O	R	T	E	X	A	M	P	L	E
M	O	R	T	E	X	A	S	P	L	E
M	O	R	T	E	X	A	S	P	L	E
M	O	L	T	E	X	A	S	P	R	E
M	O	L	E	E	X	A	S	P	R	T

3-sort

M	O	L	E	E	X	A	S	P	R	T
E	O	L	M	E	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T

1-sort

A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	E	L	O	P	M	S	X	R	T
A	E	E	L	O	P	M	S	X	R	T
A	E	E	L	O	P	M	S	X	R	T
A	E	E	L	M	O	P	S	X	R	T
A	E	E	L	M	O	P	S	X	R	T
A	E	E	L	M	O	P	S	X	R	T
A	E	E	L	M	O	P	R	S	X	T
A	E	E	L	M	O	P	R	S	T	X

result

A E E L M O P R S T X

Shellsort: Java implementation (Algorithm 2.3)

```
public class Shell
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;

      int h = 1;
      while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, ...

      while (h >= 1)
      {  // h-sort the array.
         for (int i = h; i < N; i++)
         {
            for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
               exch(a, j, j-h);
         }

         h = h/3;
      }
   }

   private static boolean less(Comparable v, Comparable w)
   { /* as before */ }
   private static void exch(Comparable[] a, int i, int j)
   { /* as before */ }
}
```

3x+1 increment sequence

insertion sort

move to next increment

Algorithm 2.3 increment sequence

- Algorithm 2.3 uses the sequence of decreasing values $\frac{1}{2}(3^k - 1)$, starting at the largest increment less than $N/3$ and decreasing to 1 - the book refers to such a sequence as an **increment sequence**.
- The sequences generated by $\frac{1}{2}(3^k - 1)$, where $k > 1$, and the one used in the *while* loop of the code $h = 3h + 1$ are the same; that is, the sequence $1, 4, 13, 40, 121, \ldots$.
- This sequence is easy to compute and use, and performs nearly as well as more sophisticated increment sequences.

Shellsort: which increment sequence to use?

- Difficult one to answer.
- Many different increment sequences have been studied in the literature, but no provably best sequence has been found.
- The sequence used by Algorithm 2.3 is easy to compute and use, and performs nearly as well as more sophisticated increment sequences.

Shellsort: Properties

- Useful in practice.
  - Fast unless array size is huge.
  - Used in some embedded systems.
  - Hardware sort prototype.
- Time complexity (for 3x+1):
  - Best: $O(N \log N)$
  - Worst: $O(N^{1.5})$
  - Average: open research problem even for uniform input distribution