CIS 2168 Data Structures **Fall 2014**

Lecture 14: Nov. 11 & 13

Lecturer: Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

14.1 Sorting

14.1.1 Insertion Sort

Insertion sort sorts the input by inserting each item into a sorted list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Figure 14.1: Insertion Sort

Listing 1: Insertion Sort

```
1 import java.util.Comparator;
2 /** *
3  *  @author Robert Sedgewick
4  *  @author Kevin Wayne
```

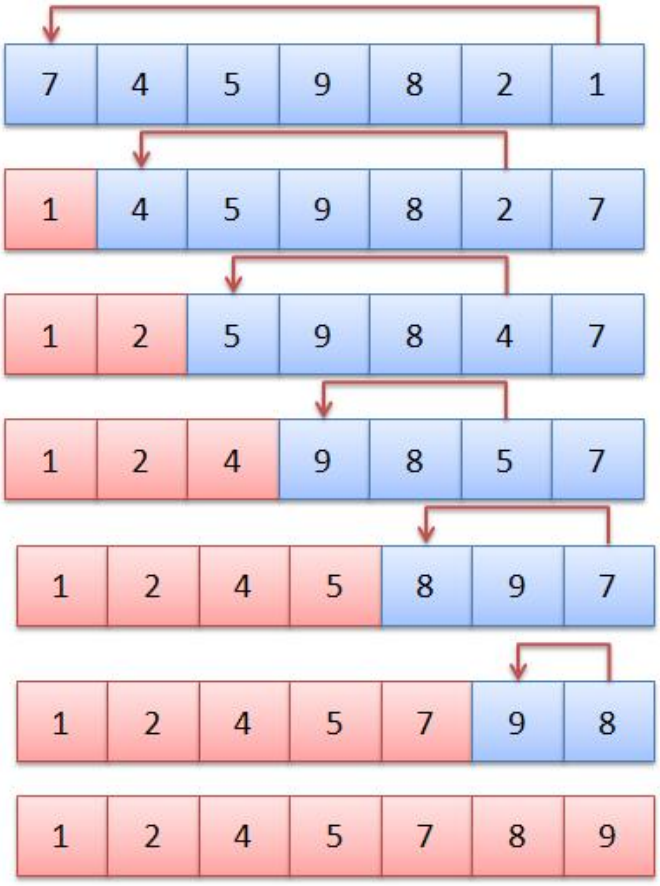
```
5  */
6  public class Insertion {
7      // This class should not be instantiated.
8      private Insertion() { }
9
10     /**
11      * Rearranges the array in ascending order, using the natural order.
12      * @param a the array to be sorted
13      */
14     public static void sort(Comparable[] a) {
15         int N = a.length;
16         for (int i = 0; i < N; i++) {
17             for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
18                 exch(a, j, j-1);
19             }
20         }
21     }
22
23     /**
24      * Rearranges the array in ascending order, using a comparator.
25      * @param a the array
26      * @param c the comparator specifying the order
27      */
28     public static void sort(Object[] a, Comparator c) {
29         int N = a.length;
30         for (int i = 0; i < N; i++) {
31             for (int j = i; j > 0 && less(c, a[j], a[j-1]); j--) {
32                 exch(a, j, j-1);
33             }
34         }
35     }
36
37    /***************************************************************************
38     *  Helper sorting functions
39     ***************************************************************************/
40     // is v < w ?
41     private static  boolean less(Comparable v, Comparable w) {
42         return (v.compareTo(w) < 0);
43     }
44
45     // is v < w ?
46     private static  boolean less(Comparator c, Object v, Object w) {
47         return (c.compare(v, w) < 0);
48     }
49
50     // exchange a[i] and a[j]
51     private static  void exch(Object[] a, int i, int j) {
52         Object swap = a[i];
53         a[i] = a[j];
54         a[j] = swap;
55     }
```

```
56
57      // exchange a[i] and a[j]  (for indirect sort)
58      private static  void exch(int[] a, int i, int j) {
59          int swap = a[i];
60          a[i] = a[j];
61          a[j] = swap;
62      }
63
64      /**
65       * Reads in a sequence of strings from standard input; insertion sorts them;
66       * and prints them to standard output in ascending order.
67       */
68      public static void main(String[] args) {
69          String[] a = new String[]{"Bob", "Alice", "Frank", "Harry", "Cathy",
70                                    "David"};
71          Insertion.sort(a);
72          for(String s:a){
73                  System.out.print(s+",");
74          }
75                  System.out.println();
76      }
77  }
```

14.1.2 Selection Sort

Selection sort divides the list into two parts: sorted list (shown as red in Figure 14.2) and unsorted list (shown as black in Figure 14.2). Initially the sorted list is empty. In each iteration, the algorithm finds the smallest (or largest, depending on sorting order) element in the unsorted list, exchanges it with the leftmost unsorted element (putting it in sorted order), and moves the unsorted list boundary one element to the right. It will repeat until the unsorted list becomes empty.

Figure 14.2: Selection Sort



The sort method of selection is sort is shown in Listing 2. "less" and "exch" methods are same as in Insertion Sort code.

Listing 2: Selection Sort

```
1  public static void sort(Comparable[] a) {
2          int N = a.length;
3          for (int i = 0; i < N; i++) {
4              int min = i;
5              for (int j = i+1; j < N; j++) {
6                  if (less(a[j], a[min])) min = j;
7              }
8              exch(a, i, min);
9          }
10     }
```

14.1.3 Merge Sort

Merge sort is a divide and conquer algorithm and it works as follows:

- Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted) as shown in Figure 14.3.
- Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list, as show in Figure 14.4.

Figure 14.3: Partition

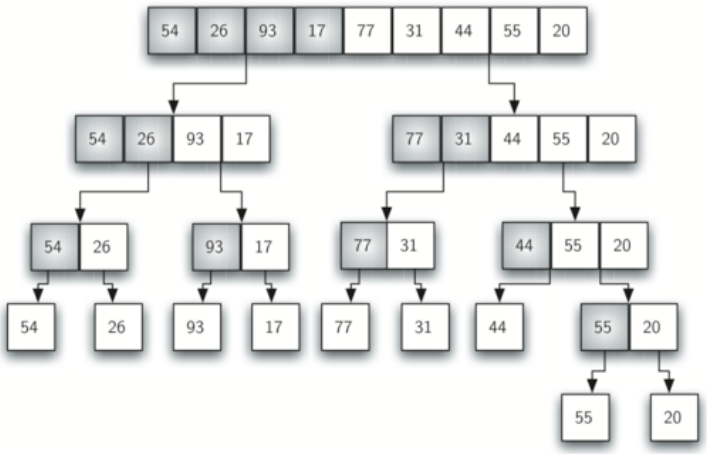
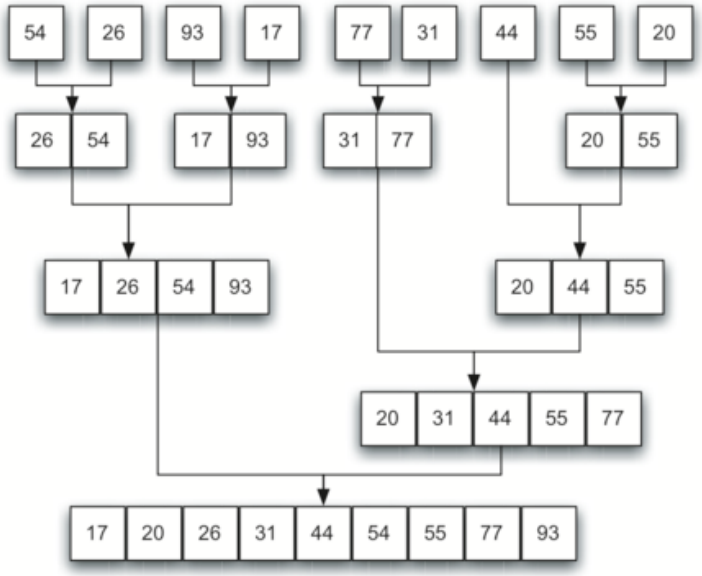


Figure 14.4: Merging



Listing 3: Merge Sort

```
1  public class Merge {
2
3      // This class should not be instantiated.
4      private Merge() { }
5
6      // stably merge a[lo .. mid] with a[mid+1 ..hi] using aux[lo .. hi]
7      private static void merge(Comparable[] a, Comparable[] aux, int lo,
8                  int mid, int hi) {
9          // precondition: a[lo .. mid] and a[mid+1 .. hi] are sorted subarrays
10
11         // copy to aux[]
12         for (int k = lo; k <= hi; k++) {
13             aux[k] = a[k];
14         }
15
16         // merge back to a[]
17         int i = lo, j = mid+1;
18         for (int k = lo; k <= hi; k++) {
19             if       (i > mid)                a[k] = aux[j++];
20             else if (j > hi)                  a[k] = aux[i++];
21             else if (less(aux[j], aux[i])) a[k] = aux[j++];
22             else                              a[k] = aux[i++];
23         }
24     }
25
26     // mergesort a[lo..hi] using auxiliary array aux[lo..hi]
27     private static void sort(Comparable[] a,Comparable[] aux, int lo,int hi) {
28         if (hi <= lo) return;
29         int mid = lo + (hi - lo) / 2;
30         sort(a, aux, lo, mid);
31         sort(a, aux, mid + 1, hi);
32         merge(a, aux, lo, mid, hi);
33     }
34
35     /**
36      * Rearranges the array in ascending order, using the natural order.
37      * @param a the array to be sorted
38      */
39     public static void sort(Comparable[] a) {
40         Comparable[] aux = new Comparable[a.length];
41         sort(a, aux, 0, a.length-1);
42     }
43
44    /***************************************************************************
45     *  Helper sorting functions
46     ***************************************************************************/
47
48     // is v < w ?
49     private static boolean less(Comparable v, Comparable w) {
50         return (v.compareTo(w) < 0);
51     }
```

```
52
53      // exchange a[i] and a[j]
54      private static void exch(Object[] a, int i, int j) {
55          Object swap = a[i];
56          a[i] = a[j];
57          a[j] = swap;
58      }
59
60      /**
61       * Reads in a sequence of strings from standard input; mergesorts them;
62       * and prints them to standard output in ascending order.
63       */
64      public static void main(String[] args) {
65          String[] a = new String[]{"Alice", "David","Bob","Cathy", "Harry"};
66          Merge.sort(a);
67          for(String s:a){System.out.print(s+",");}
68          System.out.println();
69      }
70  }
```

14.1.4 Quick Sort

14.1.5 Heap Sort

14.1.6 Sorting Algorithm Comparison

Name	Best	Average	Worst	Memory	Stable
Bubble Sort	n	n^2	n^2	1	yes
Selection Sort	n^2	n^2	n2	1	no
Insertion Sort	n	n^2	n^2	1	yes
Merge Sort	$nlogn$	$nlogn$	nlogn	n	yes
Quick Sort	$nlogn$	$nlogn$	n^2	$logn$	no
Heap Sort	$nlogn$	$nlogn$	$nlogn$	1	no

References

[Algorithms, 4/E] Princeton University, Robert Sedgewick and Kevin Wayne