Elementary Sorts

Chapter 2 – Sorting

Algorithms 4th Ed.

Wayne & Sedgewick

Agenda

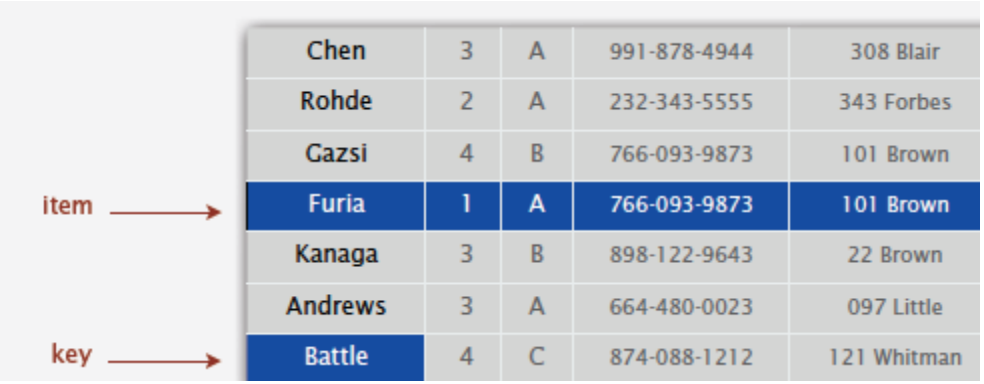
- Comparable
- Selection Sort
- Insertion Sort
- Shell Sort
- Shuffles

Goal of sorting

- The objective of a sorting algorithm is to rearrange the items such that their keys are ordered according to some well-defined ordering rule (usually numerical or alphabetical order)
- Each item contains a key
- Keys as SORTABLE.

Example

- Unsorted



Chen	3	A	991-878-4944	308 Blair
Rohde	2	A	232-343-5555	343 Forbes
Gazsi	4	B	766-093-9873	101 Brown
Furia	1	A	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman

- Sorted

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

Examples

```
public static void main(String[] args) {
int n= 10;
Integer a[] = new Integer[n];

Random generator = new Random();
System.out.println("UnSorted Array:");
for (int i=0;i<n;i++){
a[i] = generator.nextInt(50);
    System.out.print(a[i] + ", " );
}

Insertion.sort(a);
System.out.println("\nSorted Array:");
for (int i=0;i<n;i++){
    System.out.print(a[i] + ", ");
}
```

```
public static void main(String[] args) {
int n= 10;
String a[] = {"apple", "pear",
“avocado","banana","pinapple",
"peach","plum","orange","blueberry","kiwi"};

System.out.println("UnSorted Array:");
for (int i=0;i<n;i++){
    System.out.print(a[i] + ", " );
}

Insertion.sort(a);
System.out.println("\nSorted Array:");
for (int i=0;i<n;i++){
    System.out.print(a[i] + ", ");
}
```

UnSorted Array:
19, 3, 24, 31, 48, 22, 44, 37, 25, 45,
Sorted Array:
3, 19, 22, 24, 25, 31, 37, 44, 45, 48,

UnSorted Array:
apple, pear, avocado, banana, pinapple, peach, plum, orange, blueberry, kiwi,
Sorted Array:
apple, avocado, banana, blueberry, kiwi, orange, peach, pear, pinapple, plum,

http://algs4.cs.princeton.edu/21elementary/Insertion.java

Examples

```
public static void main(String[] args) {
    int n= 10;
    File directory = new File(".");
    File[] files = directory.listFiles();
    Insertion.sort(files);
    System.out.println("Sorted Array:");
    for (int i=0;i<n;i++){
        System.out.print(files[i].getName() + ", " );
    }
}
```

Sorted Array:
.classpath, .project, .settings, 16Kints.txt, 1Kints.txt, 1Mints.txt, 2Kints.txt, 32Kints.txt, 4Kints.txt, 8Kints.txt

Comparable Interface

- Q.How does the same sort() method in previous examples work with Files, Strings, Doubles???
- A.They all implement the Comparable interface. (Remember interfaces from 1st Week)
- Sometimes known as “Callback”

Comparable Example

client

```
public static void main(String[] args) {
      int n= 10;
      File directory = new File(".");
      File[] files = directory.listFiles();
      Insertion.sort(files);
      System.out.println("Sorted Array:");
      for (int i=0;i<n;i++){
          System.out.print(files[i].getName() + ", " );
      }
}
```

object implementation

```
public class File
implements Comparable<File>
{
   ...
   public int compareTo(File b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
   public int compareTo(Item that);
}
```

sort implementation

```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

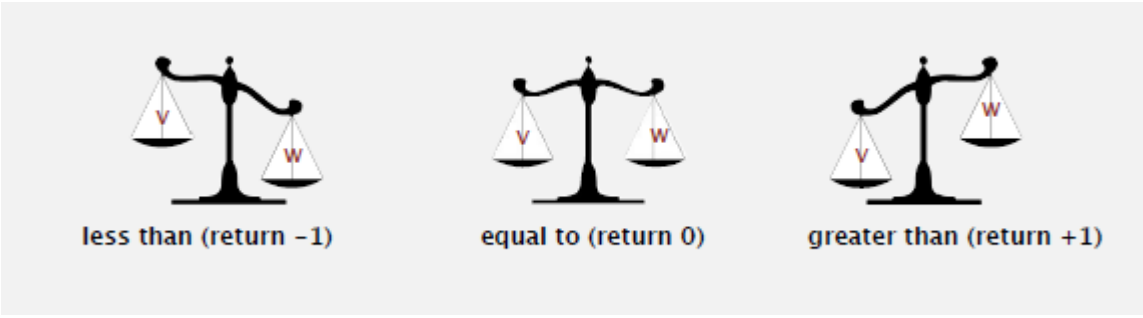
key point: no dependence on File data type

Total Order

- compareTo() must implement a *total order. It must be:*
  - *Reflexive (for all v, v = v)*
  - *Antisymmetric (for all v and w, if* v < w then w > v and if v = w then w = v)
  - *Transitive (for all v, w, and x, if* v <= w and w <= x then v <=x)
- Example of something that doesn't have Total Order
  - Rock, Paper, Scissors game...

Implementing Comparable Interface

- Must implement compareTo() such that v.compareTo(w)
  - Returns a negative integer if v is less than w
  - Returns a positive integer if v is greater than w
  - Returns 0 if v is equal to w
  - Is a Total Order



Date Class Example

```
public class Date implements Comparable<Date>
{
   private final int month, day, year;

   public Date(int m, int d, int y)
   {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date that)
   {
      if (this.year  < that.year ) return -1;
      if (this.year  > that.year ) return +1;
      if (this.month < that.month) return -1;
      if (this.month > that.month) return +1;
      if (this.day   < that.day  ) return -1;
      if (this.day   > that.day  ) return +1;
      return 0;
   }
}
```

only compare dates to other dates

http://algs4.cs.princeton.edu/21elementary/Date.java

Selection Sort

- General Idea:
  - Iterate through and array of n items a[n-1], starting at i=0
  - In iteration i, find the index of the smallest remaining enrty
  - Swap a[i] and a[min]
- http://en.wikipedia.org/wiki/Selection_sort
- http://algs4.cs.princeton.edu/lectures/21DemoSelectionSort.mov

Selection Sort – Java Implementation

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }

private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }

private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

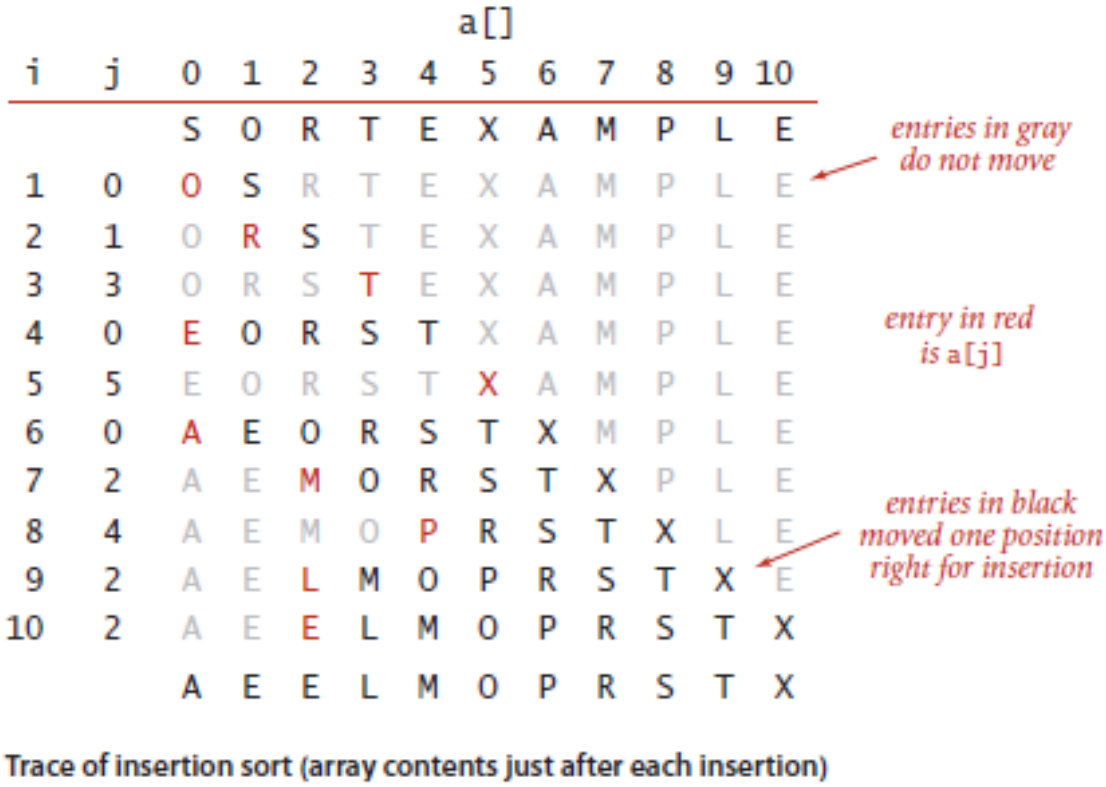
Selection Sort Efficiency

- **Input size metric:** length of Array, N.
- **Is running cost the same for different inputs of same size?** YES
- **Basic Operation:** Comparison (the "if" statement)
- (N-1) + (N-2) + (N-3) + ...+ 2 + 1 compares and at most N exchanges.
- $\sim n^2$ running time (classed as quadratic)

Insertion Sort

- General Idea:
  - Iterate through and array of n items a[n-1], starting at i=0
  - In iteration i, swap a[i] with each larger entry to its left
  - http://en.wikipedia.org/wiki/Insertion_sort
  - http://algs4.cs.princeton.edu/lectures/21DemoInsertionSort.mov

Insertion Sort



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

Insertion Sort Implementation

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
               exch(a, j, j-1);
            else break;
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

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

Insertion Sort Efficiency

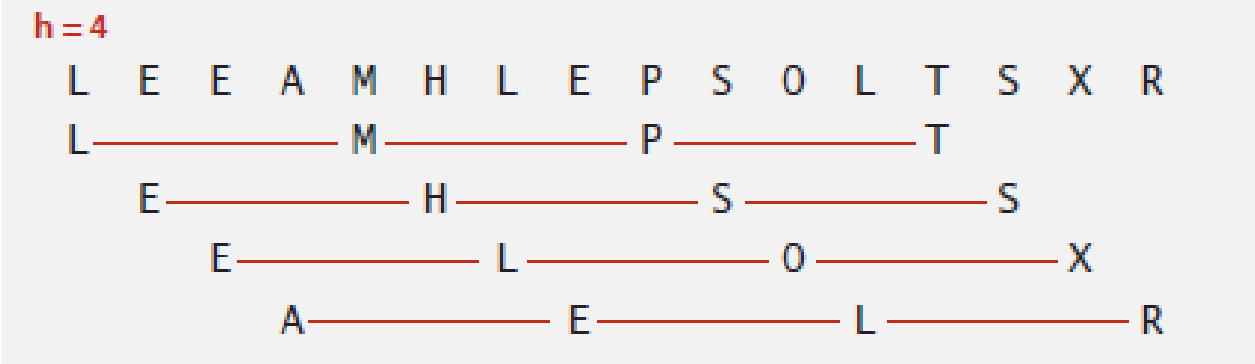
- Input Size Metric : Length of array, N.
- Is running cost the same for different inputs of same size? NO
- Best Case: Array already sorted so N-1 compares and 0 exchanges (linear)
- Worst Case: Array in descending order(wrong way sorted) ½ N^2 compares and ½ N^2 exchanges (quadratic)
- Average Case: For a random ordered class
  approx ¼ N^2 compares and ¼ N^2 exchanges on average.

Insertion Sort and Partially Sorted Arrays

- Efficiency class of selection and insertion are the same for random arrays
- However, insertion sort works better for "partially sorted arrays" where:
  - each entry is not far from final position
  - Small array appended to larger sorted array
  - Array with only a few elements out of place

Shell Sort

- General Idea:
  - Same as insertion sort but move entries more than one position at a time.
- Known as h-sorting. Swap a[i] with larger entry h positions to the left



Example

- Insertion sort with step size h (red entry indicates a[i], black entries have moved h to left, grey entries stay the same)

Shell Sort – 7,3,1

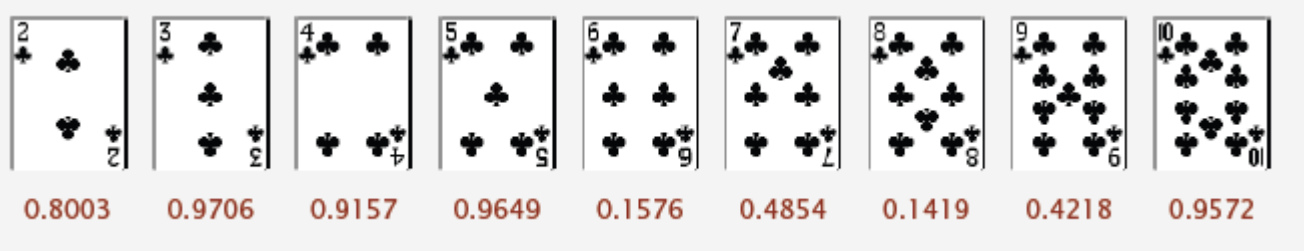
(red entry indicates a[i], black entries have moved h to left, grey entries stay the same)

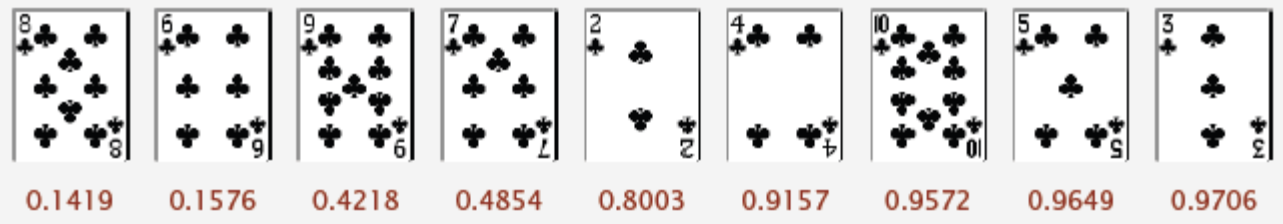
Shellsort vs. Insertion/Selection

- Simple idea leading to substantial performance gains
- Fast unless array size is huge (used for small subarrays).
- Tiny, fixed footprint for code (used in some embedded systems).
- Best sequence of increments still a mystery...

Shuffling

- Like shuffling a deck of cards – goal is to rearrange items so that they are uniformly random
- How to use sorting to do this:
  - Generate random number for each item
  - Sort based on random number





Not so random...

- Microsoft to provide randomised choice of browser for Windows 7

http://www.robweir.com/blog/2010/02/microsoft-random-browser-ballot.html

More Shuffling

- Knuth (author: The Art of Programming) provided another solution:
- For an array a[n] of items.
  - In iteration i, pick integer r between 0-i
  - Swap a[i] and a[r]
- Produces uniform random array
- Efficiency class: O(n) – linear.
- http://algs4.cs.princeton.edu/lectures/21DemoKnuthShuffle.mov

Code

```
public class StdRandom
{
   ...
   public static void shuffle(Object[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int r = StdRandom.uniform(i + 1);
         exch(a, i, r);
      }
   }
}
```

between 0 and i

Got to be careful with shuffles

- Online poker (www.planetpoker.com) got it wrong...
- http://www.cigital.com/papers/download/developer_gambling.php

Shuffling algorithm in FAQ at www.planetpoker.com

```
for i := 1 to 52 do begin
   r := random(51) + 1;
   swap := card[r];
   card[r] := card[i];
   card[i] := swap;
end;
```

← between 1 and 51

Bug 1. Random number r never 52 ⇒ 52nd card can't end up in 52nd place.

Bug 2. Shuffle not uniform (should be between 1 and i).

Bug 3. random() uses 32-bit seed ⇒ 2^{32} possible shuffles.

Bug 4. Seed = milliseconds since midnight ⇒ 86.4 million shuffles.

Leave randomness it to the Hardware…