Elementary Sorts

Insertion sort
Selection sort
Bubble sort


Basic Terms

Ex: student record in a University.

Sort: rearrange records such that keys are in ascending order.

Why Study Sorting Algorithms?

Q. Isn’t the system sort good enough.

A. Maybe.

- Is your file randomly ordered?
- Need guaranteed performance?

- Stable?
  - Multiple key types?
  - Multiple keys?
  - Deterministic?
  - Keys all distinct?
  - Linked list or arrays?
  - Large or small records?

A. An elementary sorting algorithm may be the method of choice.
A. Use well understood topic to study basic issues.

Sorting Applications

Applications.
- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.
- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

Obvious applications
- problems become easy once items are in sorted order
- non-obvious applications

Many more combinations of attributes than algorithms
Stability

A stable sort preserves the relative order of records with equal keys.

Ex: sort file on first key

Then sort file on second key

Records with key value 3 no longer in order on first key

Selection Sort

Selection sort.
- \(\uparrow\) scans from left to right.
- Elements to the left of \(\uparrow\) are fixed and in ascending order.
- No element to left of \(\uparrow\) is larger than any element to its right.

Selection Sort Example

Selection Sort Inner Loop: Maintaining the Invariant

Selection sort inner loop.
- Select minimum.
- Exchange into position.
Selection Sort in Java

```java
public class SelectionSorter {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double[] a = new double[N];
        for (int i = 0; i < N; i++)
            a[i] = Math.random();

        for (int i = 0; i < N; i++) {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (a[j] < a[min]) min = j;
            double swap = a[i];
            a[i] = a[min];
            a[min] = swap;
        }

        for (int i = 0; i < N; i++)
            System.out.println(a[i]);
    }
}
```

Abstract Comparisons

**Goal:** specify sort key such that code is reusable.

Make record implement the `Comparable` interface.

- Write `compareTo` method so that `a.compareTo(b)`
  - returns a negative integer if `a` is "less" than `b`
  - returns zero if `a` is "equal" to `b`
  - returns a positive integer if `a` is "greater" than `b`

It is the programmer’s responsibility to ensure consistency, e.g.,
transitivity: `a < b, b < c ⇒ a < c`.

**Ex:** implementation of `compareTo` to sort by `Student` GPA.

```java
public int compareTo(Object obj) {
    Student a = this;
    Student b = (Student) obj;
    return a.gpa - b.gpa;
}
```

Data Type for Student Database Records

```java
public class Student implements Comparable {
    private String first, last, email;
    private int section;
    Student(String first, String last, String email, int section) {
        this.first = first;
        this.last = last;
        this.email = email;
        this.section = section;
    }

    public int compareTo(Object obj) {
        Student a = this;
        Student b = (Student) obj;
        return a.section - b.section;
    }

    public String toString() {
        return section + " " + first + " " + last + " " + email;
    }
}
```

Sorting Student Database Records

A sample client program to process student records.

```java
public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    Student[] students = new Student[N];

    for (int i = 0; i < N; i++)
        students[i] = new Student(StdIn.readString(),
                                  StdIn.readString(),
                                  StdIn.readString(),
                                  StdIn.readInt());

    ArraySort.sort(students, 0, N-1);
    for (int i = 0; i < N; i++)
        System.out.println(students[i]);
}
```
Abstract Pointer Sort

Write abstract sorting routine `ArraySort`.

- Maintain array of Java reference to records.
- Arrange references using abstract comparisons.
- Avoid excessive data movement with large records
- Rule of thumb: cost of compare similar to cost of exchange

Abstract Selection Sort in Java

```java
public class ArraySort {

    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;
    }

    public static void sort(Comparable a[], int L, int R) {
        for (int i = L; i < R; i++) {
            int min = i;
            for (int j = i+1; j <= R; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }
}
```

Insertion Sort

- Insertion sort.
  - Scans from left to right.
  - Element to right of ▲ are not touched.
  - Invariant: elements to the left of ▲ are in ascending order.
Insertion Sort Example

| A | S | O | R | T | I | N | G | E | X | A | M | P | L | E |
| A | S | O | R | T | I | N | G | E | X | A | M | P | L | E |
| A | O | S | R | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |

= 1 comparison and 1 assignment to memory

Insertion Sort Inner Loop: Maintaining the Invariant

Insertion sort inner loop.

- Save current element.
  
  ```java
  Comparable v = a[i];
  ```

- Shift right all larger elements on left.
  
  ```java
  int j = i; // don't run off end of array
  while (j > L && less(v, a[j-1])) {
    a[j] = a[j-1]; j--;
  }
  ```

- Store v in vacant spot.
  
  ```java
  a[j] = v;
  ```

(Optimized) Insertion Sort in Java

```java
public static void sort(Comparable a[], int L, int R) {
    for (int i = R; i > L; i--)
        if (less(a[i-1], a[i]))
            exch(a, i-1, i);
    for (int i = L + 2; i <= R; i++) {
        Comparable v = a[i];
        int j = i;
        while (less(v, a[j-1])) {
            a[j] = a[j-1]; j--;
        }
        a[j] = v;
    }
}
```

Bubble Sort

- ↑ scans from left to right.
- Compare and exchange element at ↑ with element on its right.

Implications.

- First pass puts max element into position.
- Like selection sort, but with more data movement.
Bubble Sort Example

A S O R T I N G E X A M P L E
A O R I N G E S A M P L E T X
A O I N G E R A M P L E S T X
A I N G E O A M P L E R S T X
A I G E N A M O L E R P S T X
A G E I A M N L E O P R S T X
A E G A I M L E N O P R S T X
A E A G I E L M N O P R S T X
A A E G I E L M N O P R S T X
A A E G I E L M N O P R S T X
A A E G I E L M N O P R S T X
A A E G I E L M N O P R S T X
A A E G I E L M N O P R S T X

≈ 1 comparison and 2 assignments to memory

Performance for Randomly Ordered Files

Insertion.
- Each element moves halfway back.
- \(\frac{(1 + 2 + \ldots + N)}{2} \approx \frac{N^2}{4}\) compares.
- \(\approx \frac{N^2}{4}\) exchanges.

Selection.
- Always search through right part.
- \(\frac{(1 + 2 + \ldots + N)}{2} \approx \frac{N^2}{2}\) compares.
- \(\approx N\) exchanges.

Bubble.
- Mostly compare-exchanges.
- \(\frac{(1 + 2 + \ldots + N)}{2} \approx \frac{N^2}{2}\) compares.
- \(\approx \frac{N^2}{2}\) exchanges.

Bottom line: insertion, selection similar; never use bubble.

Sorting Challenge 1

Problem: sort a file of huge records with tiny keys.
Ex: reorganizing your MP3 files.
Which sorting method to use?

a) system sort, guaranteed to run in time \(N \log N\)
b) insertion sort
c) selection sort
d) bubble sort

Sorting Challenge 2

Problem: sort a huge randomly-ordered file of small records.
Ex: process transaction records for a phone company.
Which sorting method to use?

a) system sort
b) insertion sort
c) selection sort
d) bubble sort
Sorting Challenge 3

**Problem:** sort a huge number of tiny files (each file is independent)

**Ex:** daily customer transaction records.

Which sorting method to use?

a) system sort
b) insertion sort
c) selection sort
d) bubble sort

Sorting Challenge 4

**Problem:** sort a huge file that is already almost in order.

**Ex:** re-sort a huge database after a few changes.

Which sorting method to use?

a) system sort
b) insertion sort
c) selection sort
d) bubble sort

Visual Sorting Puzzle

A. Insertion sort.
B. Selection sort.
C. Bubble sort.