Elementary Sorts

Insertion sort

Selection sort

Bubble sort

Reference: Chapter 6, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Basic Terms

Ex: student record in a University.

file →	Fox	1	A	243-456-9091	101 Brown
	Quilici	1	C	343-987-5642	32 McCosh
	Chen	2	A	884-232-5341	11 Dickinson
	Furia	3	A	766-093-9873	22 Brown
	Kanaga	3	B	898-122-9643	343 Forbes
record →	Andrews	3	A	874-088-1212	121 Whitman
	Rohde	3	A	232-343-5555	115 Holder
	Battle	4	C	991-878-4944	308 Blair
key →	Aaron	4	A	664-480-0023	097 Little
	Gazsi	4	B	665-303-0266	113 Walker

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

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Sorting Applications

Applications.

- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.

← obvious applications

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

← problems become easy once items are in sorted order

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

← non-obvious applications

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?

		attributes								
		1	2	3	4	.	.	.	.	M
algorithm	A	●			●					
	B			●		●			●	
	C		●		●					
	D						●			
	E			●						
	F		●			●		●		
	G	●								●
	.			●		●		●		
	.		●	●				●		
	.						●			●
	K	●				●				

many more combinations of attributes than algorithms

A. An elementary sorting algorithm may be the method of choice.

A. Use well understood topic to study basic issues.

Stability

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

Ex: sort file on first key

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Then sort file on second key

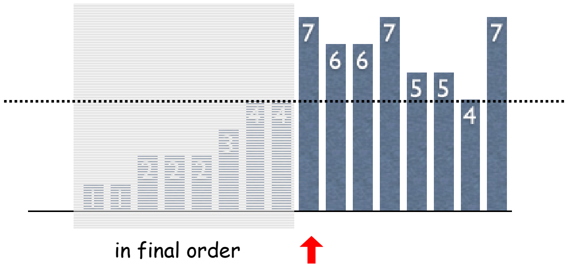
@#%&@!!
records with
key value 3
no longer in order
on first key

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Furia	3	A	766-093-9873	22 Brown
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Gazsi	4	B	665-303-0266	113 Walker
Aaron	4	A	664-480-0023	097 Little

Selection Sort

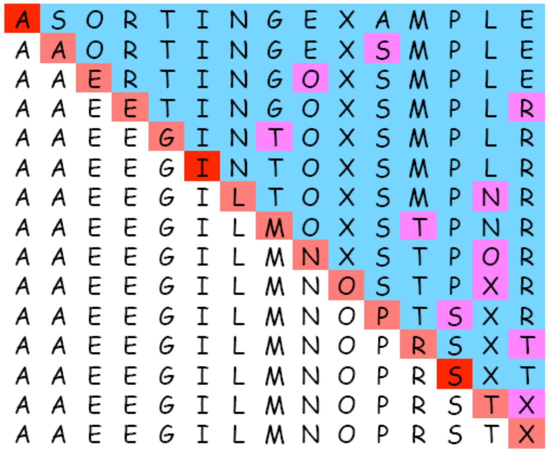
Selection sort.

- ↑ scans from left to right.
- Elements to the left of ↑ are fixed and in ascending order.
- No element to left of ↑ is larger than any element to its right.



in final order

Selection Sort Example



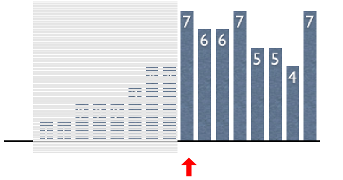
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	O	R	T	I	N	G	E	X	S	M	P	L	E
A	A	E	R	T	I	N	G	O	X	S	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E	E	G	I	N	T	O	X	S	M	P	L	R
A	A	E	E	G	I	N	T	O	X	S	M	P	L	R
A	A	E	E	G	I	L	T	O	X	S	M	P	N	R
A	A	E	E	G	I	L	M	O	X	S	T	P	N	R
A	A	E	E	G	I	L	M	N	X	S	T	P	O	R
A	A	E	E	G	I	L	M	N	O	S	T	P	X	R
A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

🟥 = 1 assignment to memory

🟦 = 1 comparison

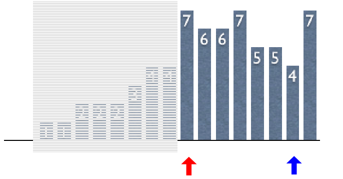
Selection Sort Inner Loop: Maintaining the Invariant

Selection sort inner loop.



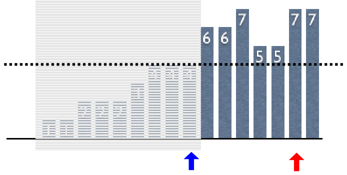
- Select minimum.

```
int min = i;
for (int j = i+1; j <= R; j++)
   if (a[j] < a[min]) min = j;
```



- Exchange into position.

```
double swap = a[j];
a[j] = a[j-1];
a[j-1] = swap;
```



Selection Sort in 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]);
   }
}
```

create an array of N real numbers between 0 and 1

selection sort it

print results

SelectionSorter.java

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: $a < b, b < c \Rightarrow a < c$.

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

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

Data Type for Student Database Records

```
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;
   }
}
```

must implement compareTo

a.compareTo(b) compares a and b

Sorting Student Database Records

A sample client program to process student records.

```
public static void main(String[] args) {

   int N = Integer.parseInt(args[0]);
   Student[] students = new Student[N];

   for (int i = 0; i < N; i++) {
      String first = StdIn.readString();
      String last  = StdIn.readString();
      String email = StdIn.readString();
      int section  = StdIn.readInt();
      students[i]  = new Student(first, last, email, section);
   }

   ArraySort.sort(students, 0, N-1);

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

we will implement this next

Sample Output

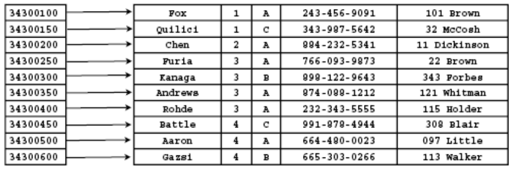
```
% more students.txt
Abraham Flamholz flamholz 3
Aditi Shrivastava ashrivas 4
Alexander Thorn athorn 2
Alicia Myers amyers 1
Amy Trangsrud atrangsr 1
Anand Dharan adharan 1
Anshuman Sahoo asahoo 3
Arthur Shum ashum 1
Arti Sheth asheth 5
Ashley Evans amevans 1
Avi Ziskind aziskind 5
Benjamin Amster bamster 2
Bryant Chen bryantc 1
Cameron Brien cbrien 3
Caroline Teichner cteichne 3
Charles Alden calden 1
Cole Deforest cde 1
Daniel Potter dpotter 3
Darin Sleiter dsleiter 3
David Astle dastle 1
. . .
Leizhi Sun leizhis 3
Lester MacKey lmackey 3
```

```
% java SedgewickSort 79 < students.txt
1 Alicia Myers amyers
1 Amy Trangsrud atrangsr
1 Anand Dharan adharan
1 Arthur Shum ashum
1 Ashley Evans amevans
1 Bryant Chen bryantc
1 Charles Alden calden
1 Cole Deforest cde
1 David Astle dastle
1 Elinor Keith ekeith
1 John Kim johnkim
1 Josh Probst jprobst
1 Julia Ressler jressler
1 Kira Hohensee hohensee
1 Maria Miguez mmiguez
1 Michael Reilly mcreilly
1 Nancy Khov nkhov
1 Tarik Jones tarikj
2 Alexander Thorn athorn
2 Benjamin Amster bamster
. . .
5 Tom Brennan tpbrenna
5 Yiting Jin ycjin
```

Abstract Pointer Sort

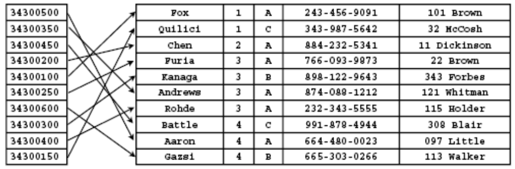
Write abstract sorting routine `ArraySort`.

- Maintain array of Java reference to records.



34300100	Fox	1	A	243-456-9091	101 Brown
34300150	Quilici	1	C	343-987-5642	32 McCosh
34300200	Chen	2	A	884-232-5341	11 Dickinson
34300250	Furia	3	A	766-093-9873	22 Brown
34300300	Kanaga	3	B	898-122-9643	343 Forbes
34300350	Andrews	3	A	874-088-1212	121 Whitman
34300400	Rohde	3	A	232-343-5555	115 Holder
34300450	Battle	4	C	991-878-4944	308 Blair
34300500	Aaron	4	A	664-480-0023	097 Little
34300600	Gazsi	4	B	665-303-0266	113 Walker

- Rearrange references using abstract comparisons.



34300500	Fox	1	A	243-456-9091	101 Brown
34300350	Quilici	1	C	343-987-5642	32 McCosh
34300450	Chen	2	A	884-232-5341	11 Dickinson
34300200	Furia	3	A	766-093-9873	22 Brown
34300100	Kanaga	3	B	898-122-9643	343 Forbes
34300250	Andrews	3	A	874-088-1212	121 Whitman
34300600	Rohde	3	A	232-343-5555	115 Holder
34300300	Battle	4	C	991-878-4944	308 Blair
34300400	Aaron	4	A	664-480-0023	097 Little
34300150	Gazsi	4	B	665-303-0266	113 Walker

  - avoids excessive data movement with large records
  - rule of thumb: cost of compare similar to cost of exchange

Abstract Selection Sort in Java

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

is v less than w?

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

swap references a[i] and a[j]

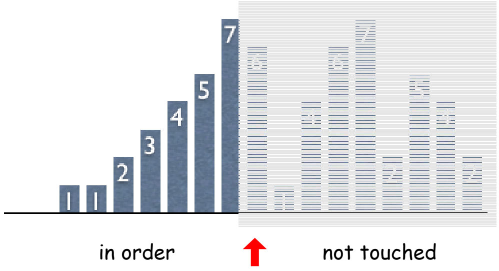
```
   public static void sort(Comparable a[], int L, int R) {
      for (int i = L; i < R; i++) {
         int min = i;
         for (int j = i+1; j <= R; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }
}
```

selection sort a[L] to a[R]

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

▨ = 1 comparison and 1 assignment to memory

Insertion Sort Inner Loop: Maintaining the Invariant

Insertion sort inner loop.

- Save current element.

```
Comparable v = a[i];
```

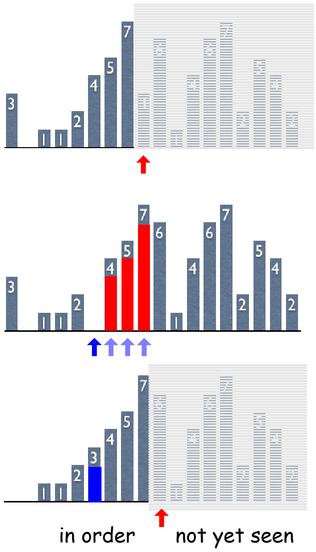
- Shift right all larger elements on left.

```
int j = i;
while (j > L && less(v, a[j-1])) {
   a[j] = a[j-1];
   j--;
}
```

don't run off end of array

- Store v in vacant spot.

```
a[j] = v;
```



(Optimized) Insertion Sort in 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;
   }

}
```

put smallest element into position to act as "sentinel"

with sentinel, no need to worry about end of array

Bubble Sort

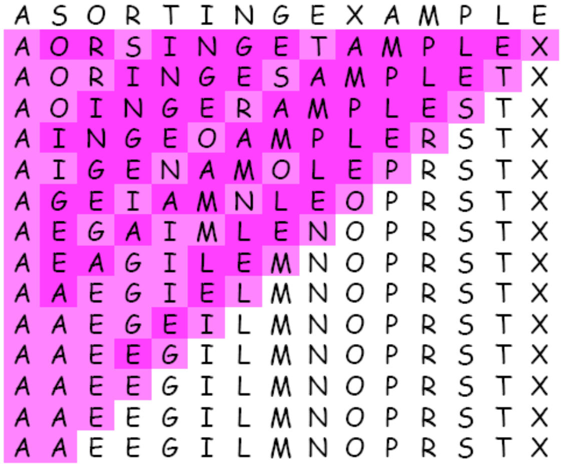
Bubble sort.

- ↑ 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



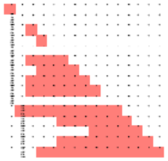
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	O	R	S	I	N	G	E	T	A	M	P	L	E	X
A	O	R	I	N	G	E	S	A	M	P	L	E	T	X
A	O	I	N	G	E	R	A	M	P	L	E	S	T	X
A	I	N	G	E	O	A	M	P	L	E	R	S	T	X
A	I	G	E	N	A	M	O	L	E	P	R	S	T	X
A	G	E	I	A	M	N	L	E	O	P	R	S	T	X
A	E	G	A	I	M	L	E	N	O	P	R	S	T	X
A	E	A	G	I	L	E	M	N	O	P	R	S	T	X
A	A	E	G	I	E	L	M	N	O	P	R	S	T	X
A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

🟪 = 1 comparison and 2 assignments to memory

Performance for Randomly Ordered Files

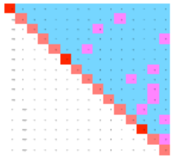
Insertion.

- Each element moves halfway back.
- (1 + 2 + ... + N) / 2 ~ N^2 / 4 compares.
  ~ N^2 / 4 exchanges.



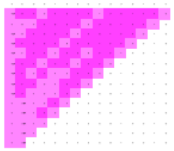
Selection.

- Always search through right part.
- (1 + 2 + ... + N) ~ N^2 / 2 compares.
  ~ N exchanges.



Bubble.

- Mostly compare-exchanges.
- (1 + 2 + ... + N) ~ N^2 / 2 compares.
  ~ N^2 / 2 exchanges.



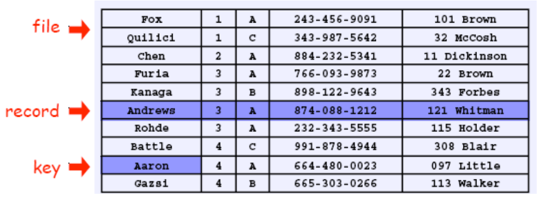
Bottom line: insertion, selection similar; never use bubble.

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


file
record
key

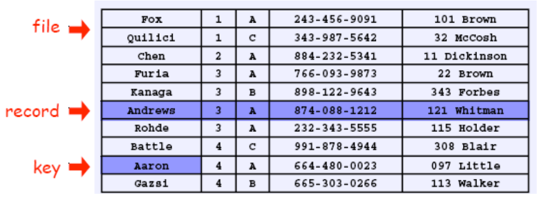
Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

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


file
record
key

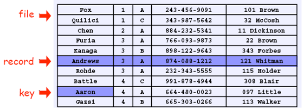
Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 3

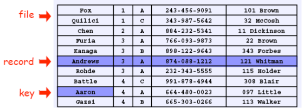
Problem: sort a huge number of tiny files (each file is independent)
Ex: daily customer transaction records.

Which sorting method to use?

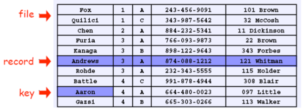
a) system sort
b) insertion sort
c) selection sort
d) bubble sort


file
record
key

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker


file
record
key

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker


file
record
key

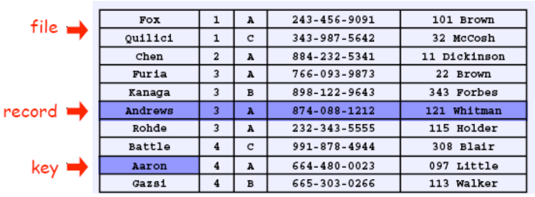
Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

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?

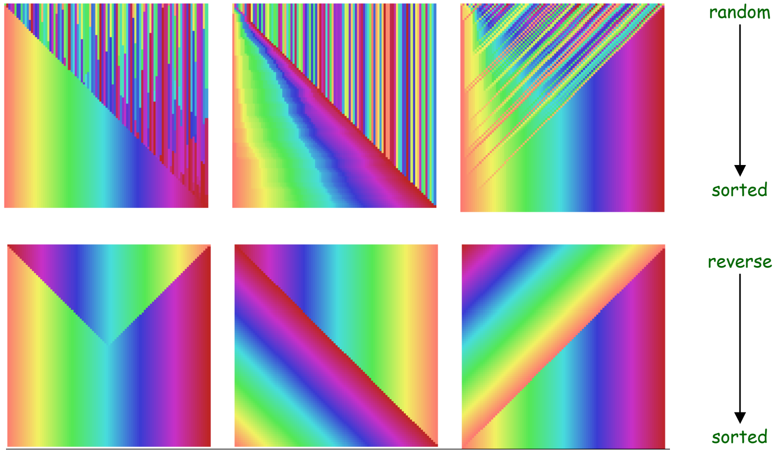
a) system sort
b) insertion sort
c) selection sort
d) bubble sort


file
record
key

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Visual Sorting Puzzle

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


random
sorted
reverse
sorted