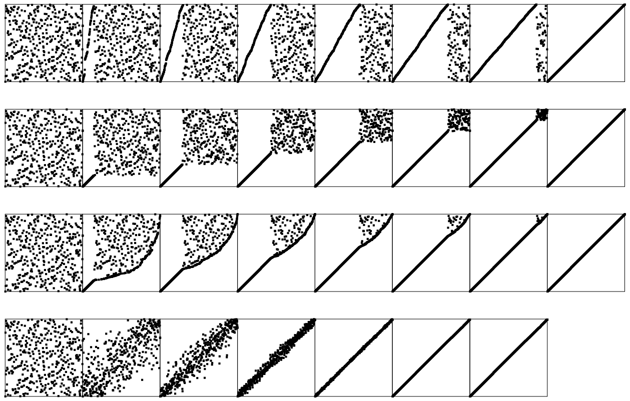
COS 226 Lecture 2: Elementary sorting algs



Why study elementary algorithms?

- Easy to code
- Fastest for small files
- Context for developing ground rules
- Fastest in some special situations
- May not be so elementary

Ground rules

FILES of RECORDS containing KEYS

File fits in memory

Use abstract comparison, exchange

```
typedef int Item
#define less(A, B)  (A < B)
#define exch(A, B)
     { Item t = A; A = B; B = t; }
```

Macros or subroutines?

- Macros: low cost, simple
- Subroutines: more general

Selection 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	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

Selection sort implementation

```
void selection(Item a[], int l, int r)
{ int i, j;
   for (i = l; i < r; i++)
      { int min = i;
         for (j = i+1; j <= r; j++)
              if (less(a[j], a[min])) min = j;
         exch(a[i], a[min]);
      }
}
```


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

Insertion sort implementation

```
void insertion(Item a[], int l, int r)
{ int i, j;
   for (i = l+1; i <= r; i++)
      { Item v = a[i];
         j = i;
         while (j > l && less(v, a[j-1]))
            { a[j] = a[j-1]; j--; }
         a[j] = v;
      }
}
```


Bubble sort example

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

Bubble sort implementation

```
void bubble(Item a[], int l, int r)
{ int i, j;
   for (i = l; i < r; i++)
      for (j = r; j > i; j--)
            compexch(a[j], a[j-1]);
}
```

Improvements:

- add a test to exit if no exchanges
- go back and forth

Properties of elementary sorts

All: quadratic running time

Selection sort

- comparisons: N-1 + N-2 + ... + 2 + 1 = N^2/2
- exchanges: N

Insertion sort (average case)

- comparisons: (N-1 + N-2 + ... + 1)/2 = N^2/4
- exchanges: N^2/4

Bubble sort

- comparisons: N-1 + N-2 + ... + 2 + 1 = N^2/2
- exchanges: about N^2/2

Special situations

Large records, small keys

- selection sort linear in amount of data
- N records M words (1-word keys)
  - comparison cost N^2/2
  - exchange cost NM
- if N is about equal to M
  - costs and amount of data are both about M^2
  - LINEAR sort

Files nearly in order

- bubble and insertion sort can be linear
- (even quicksort can be quadratic)

Pointer sort

Sort large records by swapping *references* to the records, not the records themselves

1	9	Fox	1	---	[associated info]	---
2	6	Quilici	1	---	...	---
3	8	Chen	2	---	...	---
4	3	Furia	3	---	...	---
5	1	Kanaga	3	---	...	---
6	4	Andrews	3	---	...	---
7	10	Rohde	3	---	...	---
8	5	Battle	4	---	...	---
9	2	Aaron	4	---	...	---
10	7	Gazsi	4	---	...	---

Trivial to implement: change abstract comparison

Pointer sort implementations

Array indices

```
typedef int Item
#define less(A, B) (data[A].key < data[B].key)
#define exch(A, B)
  { Item t = A; A = B; B = t; }
```

True pointers

```
typedef dataType* Item
#define less(A, B) (*A.key < *B.key)
#define exch(A, B)
  { Item t = A; A = B; B = t; }
```


Stable sorting for two-key records

Sort on the first key, then on the second

Aaron	4	Fox	1
Andrews	3	Quilici	1
Battle	4	Chen	2
Chen	2	Furia	3
Fox	1	Kanaga	3
Furia	3	Andrews	3
Gazsi	4	Rohde	3
Kanaga	3	Battle	4
Quilici	1	Aaron	4
Rohde	3	Gazsi	4

Invalid assumption: second sort preserves first sort

Stable sort

File stays sorted on first key where equal on second

Aaron	4	Fox	1
Andrews	3	Quilici	1
Battle	4	Chen	2
Chen	2	Andrews	3
Fox	1	Furia	3
Furia	3	Kanaga	3
Gazsi	4	Rohde	3
Kanaga	3	Aaron	4
Quilici	1	Battle	4
Rohde	3	Gazsi	4

Which of the elementary methods are stable?

4-sorting

Divide into 4 subfiles

- every 4th element starting at the 1st
- every 4th element starting at the 2nd
- every 4th element starting at the 3rd
- every 4th element starting at the 4th

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

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

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

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

Interleaved 4-sorting

Use insertion sort with an ``increment'' of 4

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

4-sorting implementation

```
h = 4;
for (i = l+h; i <= r; i++)
  { Item v = a[i];
    j = i;
    while (j >= l+h && less(v, a[j-h]))
      { a[j] = a[j-h]; j -= h; }
    a[j] = v;
  }
```


Shellsort

Use a decreasing sequence of increments

Each pass makes the next easier
provided increments are properly chosen

poor choice: happens to everyone
good choice: lots have been studied
best choice: research challenge (still)

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

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

Shellsort implementation

```
void shellsort(Item a[], int l, int r)
{ int i, j;
  int incs[16] = { 1391376, 463792, 198768,
     86961, 33936, 13776, 4592, 1968, 861,
     336, 112, 48, 21, 7, 3, 1 };
  for ( k = 0; k < 16; k++)
    { int h = incs[k];
      for (i = l+h; i <= r; i++)
        { Item v = a[i];
          j = i;
          while (j >= h && less(v, a[j-h]))
            { a[j] = a[j-h]; j -= h; }
          a[j] = v;
        }
    }
}
```


Shellsort summary

Need a sort routine, fast? Use Shellsort!

- not much code
- best method for small and medium files
- still OK even for giant files

How do we know what increments to use?

- plenty of proven winners to use
- easiest: 1, 4, 13, 40, 121, 364, 1093, ...

Relatively prime increment sequences

When we h-sort a file that is k-sorted,

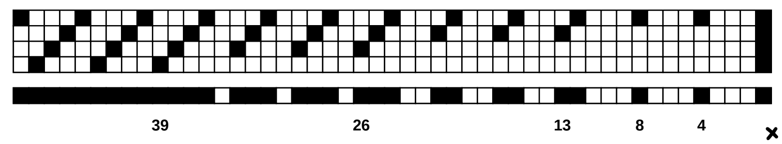
- it stays k-sorted

(Know an easy proof? SEND MAIL)

Only 18N comparisons are needed to 1-sort a file

- that is 4-sorted and 13-sorted

Elements to the left of x that could be greater:



Shellsort theory

In general, if h and k are relatively prime:

(h-1)(k-1)N comparisons (at most) to 1-sort a file that is h-sorted and k-sorted

(h-1)(k-1)N/g comparisons (at most) to g-sort a file that is h-sorted and k-sorted

Big increments (small files) h(N/h)^2 = N^2/h

Small increments, use theorem: h^2N/h = Nh

Tradeoff best bounds: N^(3/2) total

Similar methods (harder proofs) give 4/3, 5/4, 6/5 ...

More increment sequences

On the other hand, common divisors are good:

N comparisons to 1-sort a file that is 2-sorted and 3-sorted
N comparisons to 2-sort a file that is 4-sorted and 6-sorted
N comparisons to 3-sort a file that is 6-sorted and 9-sorted

```
.                     1
.                   2   3
.                 4   6   9
.               8  12  18  27
.            16  24  36  54  81
.          32  48  72 108 162 243
.        64  96 144 216 324 486 729
```

Total time: N (log N)(log N)

Too many increments for real sizes

- start with bigger numbers than 2 and 3
- throw in some primes

Have a better idea for an increment sequence?

- SEND MAIL if it beats 1 3 7 21 48 112 336 ...