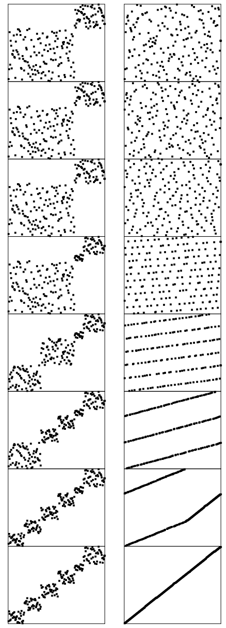
COS 226 Lecture 6: Radix sorting

- Bits and digits
- Binary Quicksort
- MSD radix sort
- Three-way radix Quicksort
- LSD radix sort
- Sorting in linear time



Bits and digits

Extracting bits is easy in C

Radix: base of number system

Power of 2 radix: groups of bits

- binary (radix-2): 1 bit at a time
- hexadecimal(radix-16): 4 bits at a time
- ascii(radix-256): 8 bits at a time

```
bin    01100001011000100110001101100100
hex       6   1   6   2   6   3   6   4
ascii         a       b       c       d
```


Extracting digits with macros

```
#define bitsword 32
#define bitsbyte 8
#define bytesword 4
#define R (1 << bitsbyte)
#define digit(A, B)
  ((A >> (bitsword-(B+1)*bitsbyte)) & (R-1))
```

Ex: Single-byte access: bitsbyte = 8

```
x = 0X61626364
digit(x, 2) = (x >> 8) & 255 = c
0110 0001 0110 0010 0110 0011 0110 0100  x
0000 0000 0110 0001 0110 0010 0110 0011  x >> 8
0000 0000 0000 0000 0000 0000 1111 1111  255 (R-1)
0000 0000 0000 0000 0000 0000 0110 0011  c
```

Ex: Single-bit access: bitsbyte = 1

```
digit(x, 11) = (x >> 20) & 1 = 0
0110 0001 0110 0010 0110 0011 0110 0100  x
0000 0000 0000 0000 0000 0110 0001 0110  x >> 20
0000 0000 0000 0000 0000 0000 0000 0001  1 (R-1)
```


Binary Quicksort

Partition file into two pieces

- all keys with first bit 0
- all keys with first bit 1

Sort two pieces recursively

Equivalent to partitioning on the VALUE 2^(bitsword-w+1)

- instead of some key in the file.

Bad partition if all keys have same leading bit

- one subfile of size N
- one empty subfile
- BUT keys one bit shorter

Worst case: one pass per key bit

Binary Quicksort code

```
quicksortB(int a[], int l, int r, int w)
{ int i = l, j = r;
   if (r <= l || w > bitsword) return;
   while (j != i)
   {
      while (digit(a[i], w) == 0 && (i < j)) i++;
      while (digit(a[j], w) == 1 && (j > i)) j--;
      exch(a[i], a[j]);
   }
   if (digit(a[r], w) == 0) j++;
   quicksortB(a, l, j-1, w+1);
   quicksortB(a, j, r, w+1);
}
```


Binary Quicksort example

A	00001	A	00001	A	00001	A	00001	A	00001	A	00001
S	10011	E	00101	E	00101	A	00001	A	00001	A	00001
O	01111	O	01111	A	00001	E	00101	E	00101	E	00101
R	10010	L	01100	E	00101	E	00101	E	00101	E	00101
T	10100	M	01101	G	00111	G	00111	G	00111	G	00111
I	01001	I	01001	I	01001	I	01001	I	01001	I	01001
N	01110	N	01110	N	01110	N	01110	L	01100	L	01100
G	00111	G	00111	M	01101	M	01101	M	01101	M	01101
E	00101	E	00101	L	01100	L	01100	N	01110	N	01110
X	11000	A	00001	O	01111	O	01111	O	01111	O	01111
A	00001	X	11000	S	10011	S	10011	P	10000	P	10000
M	01101	T	10100	T	10100	R	10010	R	10010	R	10010
P	10000	P	10000	P	10000	P	10000	S	10011	S	10011
L	01100	R	10010	R	10010	T	10100	T	10100	T	10100
E	00101	S	10011	X	11000	X	11000	X	11000	X	11000

Binary Quicksort issues

Problems:

- leading 0 bits
- cost of inner loop

  (could be advantage if carefully done)

Worst case: all keys equal

- 32N passes on a 32-bit machine
- 64N passes on a 64-bit machine

Good way to avoid quadratic worst case of quicksort

Random bits?

- should sort out after lgN bits examined

Nonrandom bits?

- take bigger chunks

MSD radix sort

Partition file into R buckets

- all keys with first byte 0
- all keys with first byte 1
- all keys with first byte 2

  ...
- all keys with first byte R-1

Sort R pieces recursively

Take R=2^bitsbyte

Tradeoff

- large R: space for buckets (too many empty buckets)
- small R: too many passes (too many keys per bucket)

Upper bound on running time: (bytesword)*(N + R)
(Worst case: all keys equal)

MSD radix sort example

now	a|ce	ac|e	ace
for	a|go	ag|o	ago
tip	a|nd	an|d	and
ilk	b|et	be|t	bet
dim	c|ab	ca|b	cab
tag	c|aw	ca|w	caw
jot	c|ue	cu|e	cue
sob	d|im	di|m	dim
nob	d|ug	du|g	dug
sky	e|gg	eg|g	egg
hut	f|or	fe|w	fee
ace	f|ee	fe|e	few
bet	f|ew	fo|r	for
men	g|ig	gi|g	gig
egg	h|ut	hu|t	hut
few	i|lk	il|k	ilk
jay	j|am	ja|y	jam
owl	j|ay	ja|m	jay
joy	j|ot	jo|t	jot
rap	j|oy	jo|y	joy
gig	m|en	me|n	men
wee	n|ow	no|w	nob
was	n|ob	no|b	now
cab	o|wl	ow|l	owl
wad	r|ap	ra|p	rap
caw	s|ob	sk|y	sky
cue	s|ky	so|b	sob
fee	t|ip	ta|g	tag
tap	t|ag	ta|p	tap
ago	t|ap	ta|r	tar
tar	t|ar	ti|p	tip
jam	w|ee	wa|d	wad
dug	w|as	wa|s	was
and	w|ad	we|e	wee

Key-indexed counting

Basis for radix sorts: sort file of keys with R values

- count number of keys with each value
- take sums to turn counts into indices
- move keys to auxiliary array using indices

Need one counter for each different key value

```
void keycount(int a[], int l, int r)
{ int i, j, cnt[R+1];
   int b[maxN];
   for (j = 0; j <  R; j++) cnt[j] = 0;
   for (i = l; i <= r; i++) cnt[a[i]+1]++;
   for (j = 1; j <  R; j++) cnt[j] += cnt[j-1];
   for (i = l; i <= r; i++)
      b[cnt[a[i]]++] = a[i];
   for (i = l; i <= r; i++) a[i] = b[i];
}
```


Key-indexed counting example

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	3	3	0	1	1	0	3	0	2	0	1	1	2	0

0	1	2	3
0	6	4	2
0	6	10	12

0	0														
3	0												3		
3	0												3	3	
0	0	0											3	3	
1	0	0					1						3	3	
1	0	0					1	1					3	3	
0	0	0	0				1	1					3	3	
3	0	0	0				1	1					3	3	3
0	0	0	0	0			1	1					3	3	3
2	0	0	0	0			1	1			2		3	3	3
0	0	0	0	0	0		1	1			2		3	3	3
1	0	0	0	0	0		1	1	1		2		3	3	3
1	0	0	0	0	0		1	1	1	1	2		3	3	3
2	0	0	0	0	0		1	1	1	1	2	2	3	3	3
0	0	0	0	0	0	0	1	1	1	1	2	2	3	3	3
	0	0	0	0	0	0	1	1	1	1	2	2	3	3	3

MSD radix sort

Three changes to key-indexed counting code

1. Modify key access to extract bytes
   - start with Most Significant Digit
   - divides files into R subfiles
2. Sort the R subfiles recursively
   - but use insertion sort for small files
3. To handle variable-length keys terminated with 0
   - remove test for end of key
   - remove recursive call corresponding to 0

Most important keys to good performance:

- fast byte extraction
- cutoff to insertion sort

MSD radix sort code

```
#define bin(A) l+count[A]
void radixMSD(Item a[], int l, int r, int w)
  { int i, j, count[R+1];
    if (w > bytesword) return;
    if (r-l <= M) { insertion(a,l,r); return; }
    for (j = 0; j < R; j++) count[j] = 0;
    for (i = l; i <= r; i++)
      count[digit(a[i], w) + 1]++;
    for (j = 1; j < R; j++)
      count[j] += count[j-1];
    for (i = l; i <= r; i++)
      b[l+count[digit(a[i], w)]++] = a[i];
    for (i = l; i <= r; i++) a[i] = b[i];
    radixMSD(a, l, bin(0)-1, w+1);
    for (j = 0; j < R-1; j++)
      radixMSD(a, bin(j), bin(j+1)-1, w+1);
  }
```


MSD radix sort potential fatal flaw

each pass ALWAYS takes time proportional to N+R

- initialize the buckets
- scan the keys

Ex: (ASCII bytes) R = 256

- 100 times slower than insertion sort for N = 2

Ex: (UNICODE) R = 65536

- 30,000 times slower than insertion sort for N = 2

TOO SLOW FOR SMALL FILES

RECURSIVE PROGRAM WILL CALL ITSELF
FOR A HUGE NUMBER OF SMALL FILES

Solution: cutoff to insertion sort

LSD radix sort

Ancient (older than computers) method

- used for card-sorting

Consider digits from right to left

- use key-indexed counting (has to be stable)

Running time: N*(bitsword/bitsbyte)

Disadvantage:

- doesn't work for variable-length keys
- totally out of order until MSD encountered

LSD radix sort code

```
void radixLSD(Item a[], int l, int r)
{
  int i, j, w, count[R+1];
  for (w = bytesword-1; w >= 0; w--)
  {
    for (j = 0; j < R; j++) count[j] = 0;
    for (i = l; i <= r; i++)
      count[digit(a[i], w) + 1]++;
    for (j = 1; j < R; j++)
      count[j] += count[j-1];
    for (i = l; i <= r; i++)
      b[count[digit(a[i], w)]++] = a[i];
    for (i = l; i <= r; i++) a[i] = b[i];
  }
}
```

LSD radix sort example

now	sob	cab	ace
for	nob	wad	ago
tip	cab	tag	and
ilk	wad	jam	bet
dim	and	rap	cab
tag	ace	tap	caw
jot	wee	tar	cue
sob	cue	was	dim
nob	fee	caw	dug
sky	tag	raw	egg
hut	egg	jay	fee
ace	gig	ace	few
bet	dug	wee	for
men	ilk	fee	gig
egg	owl	men	hut
few	dim	bet	ilk
jay	jam	few	jam
owl	men	egg	jay
joy	ago	ago	jot
rap	tip	gig	joy
gig	rap	dim	men
wee	tap	tip	nob
was	for	sky	now
cab	tar	ilk	owl
wad	was	and	rap
tap	jot	sob	raw
caw	hut	nob	sky
cue	bet	for	sob
fee	you	jot	tag
raw	now	you	tap
ago	few	now	tar
tar	caw	joy	tip
jam	raw	cue	wad
dug	sky	dug	was
you	jay	hut	wee
and	joy	owl	you

Binary LSD radix sort example

Cannot use Quicksort-style partitioning

- 0-1 sort has to be stable
- stable inplace 0-1 sort? (possible, but not easy)

A	00001	R	10010	T	10100	X	11000	P	10000	A	00001
S	10011	T	10100	X	11000	P	10000	A	00001	A	00001
O	01111	N	01110	P	10000	A	00001	A	00001	E	00101
R	10010	X	11000	L	01100	I	01001	R	10010	E	00101
T	10100	P	10000	A	00001	A	00001	S	10011	G	00111
I	01001	L	01100	I	01001	R	10010	T	10100	I	01001
N	01110	A	00001	E	00101	S	10011	E	00101	L	01100
G	00111	S	10011	A	00001	T	10100	E	00101	M	01101
E	00101	O	01111	M	01101	L	01100	G	00111	N	01110
X	11000	I	01001	E	00101	E	00101	X	11000	O	01111
A	00001	G	00111	R	10010	M	01101	I	01001	P	10000
M	01101	E	00101	N	01110	E	00101	L	01100	R	10010
P	10000	A	00001	S	10011	N	01110	M	01101	S	10011
L	01100	M	01101	O	01111	O	01111	N	01110	T	10100
E	00101	E	00101	G	00111	G	00111	O	01111	X	11000

Two proofs for LSD radix sort

Left-right

- if two keys differ on first bit
  - 0-1 sort puts them in proper relative order
- if two keys agree on first bit
  - stability keeps them in proper relative order

Right-left

- if the bits not yet examined differ
  - doesn't matter what we do now
- if the bits not yet examined agree
  - later pass won't affect their order

Linear sorting method

LSD radix sort!

To sort N 64-bit keys take bitsbyte=16

- 4N steps, linear extra memory (plus 2^16)

Does not violate NlgN lower bound because

- comparisons are not used

LSD radix sort liabilities

- inner loop has a lot of instructions
- accesses memory ``randomly''
- wastes time on low-order bits

Therefore, use just ``enough'' bits

LSD-MSD hybrid

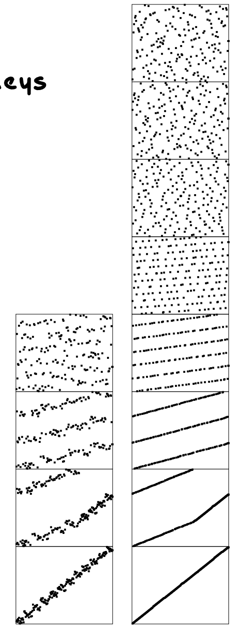
MSD radix sort also linear

Use LSD-MSD hybrid for random keys

- (assume fixed-size keys)
- use (lgN)/2 < bitsbyte < lgN

Three passes

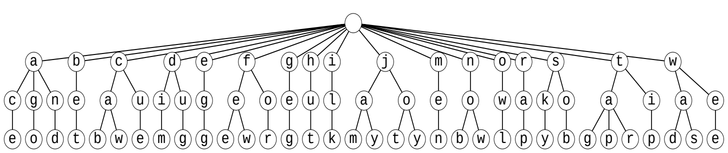
- LSD radix sort on 2nd byte
- LSD radix sort on 1st byte
- insertion sort to clean up



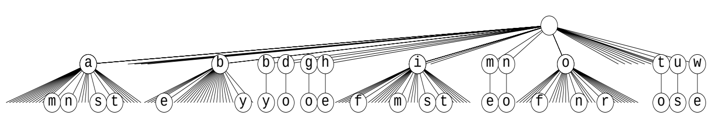
Recursive structure of MSD radix sort

Tree structure to describe recursive call

Paths in tree give keys



Problem: algorithm touches empty nodes



Tree can be as much as M times bigger than they seem

Sorting strings

PROBLEM:

- long key strings costly to compare
  when they differ only at the end
- [this is the common case!]

```
absolutism
absolut
absolutely
absolute
```

SOLUTION: 3-way radix Quicksort

- Use three-way partitioning on key characters
- Recurse and pass current char index

3-way radix Quicksort partitioning

actinian	coenobite	actinian
jeffrey	conelrad	bracteal
coenobite	actinian	coenobite
conelrad	bracteal	conelrad
secureness	secureness	cumin
cumin	dilatedly	chariness
chariness	inkblot	centesimal
bracteal	jeffrey	cankerous
displease	displease	circumflex
millwright	millwright	millwright
repertoire	repertoire	repertoire
dourness	dourness	dourness
centesimal	southeast	southeast
fondler	fondler	fondler
interval	interval	interval
reversionary	reversionary	reversionary
dilatedly	cumin	secureness
inkblot	chariness	dilatedly
southeast	centesimal	inkblot
cankerous	cankerous	jeffrey
circumflex	circumflex	displease

3-way radix Quicksort code

```
#define ch(A) digit(A, D)
void quicksortX(Item a[], int l, int r, int D)
{
   int i, j, k, p, q; int v;
   if (r-l <= M) { insertion(a, l, r); return; }
   v = ch(a[r]); i = l-1; j = r; p = l-1; q = r;
   while (i < j)
   {
      while (ch(a[++i]) < v) ;
      while (v < ch(a[--j])) if (j == l) break;
      if (i > j) break;
      exch(a[i], a[j]);
      if (ch(a[i])==v) { p++; exch(a[p], a[i]); }
      if (v==ch(a[j])) { q--; exch(a[j], a[q]); }
   }
   if (p == q)
      { if (v != '\0') quicksortX(a, l, r, D+1);
        return; }
   if (ch(a[i]) < v) i++;
   for (k = l; k <= p; k++, j--) exch(a[k], a[j]);
   for (k = r; k >= q; k--, i++) exch(a[k], a[i]);
   quicksortX(a, l, j, D);
   if ((i == r) && (ch(a[i]) == v)) i++;
   if (v != '\0') quicksortX(a, j+1, i-1, D+1);
   quicksortX(a, i, r, D);
}
```


3-way radix Quicksort example

```
now  gig   ace   ago   a|go
for  for   bet   bet   a|ce
tip  dug   dug   and   a|nd
ilk  ilk   cab   ace   b|et
dim  dim   dim   c|ab
tag  ago   ago   c|aw
jot  and   and   c|ue
sob  fee   egg   egg
nob  cue   cue   dug
sky  caw   caw   dim
hut  hut   f|ee
ace  ace   f|or
bet  bet   f|ew
men  cab   ilk
egg  egg   gig
few  few   hut
jay  j|ay  ja|m
owl  j|ot  ja|y
joy  j|oy  jo|y
rap  j|am  jo|t
gig  owl   owl   m|en
wee  wee   now   owl
was  was   nob   nob
cab  men   men   now
wad  wad   r|ap
caw  sky   sky   sky   sky
cue  nob   was   tip   sob
fee  sob   sob   sob   t|ip  ta|r
tap  tap   tap   tap   t|ap  ta|p
ago  tag   tag   tag   t|ag  ta|g
tar  tar   tar   tar   t|ar  ti|p
dug  tip   tip   w|as
and  now   wee   w|ee
jam  rap   wad   w|ad
```


Another sublinear sort

Three-way radix quicksort is SUBLINEAR

N records with w-byte keys
- Bytes of data: Nw
- Bytes examined by sort: 2 N ln N

Ex: 100000 keys, 100 bytes per key
- 10 million bytes of data
- algorithm examines 2.3 million bytes

  1/5 of the data

Corresponds to collapsing null links in MSD trees