COS 226 Lecture 6: Radix sorting

Bits and digits

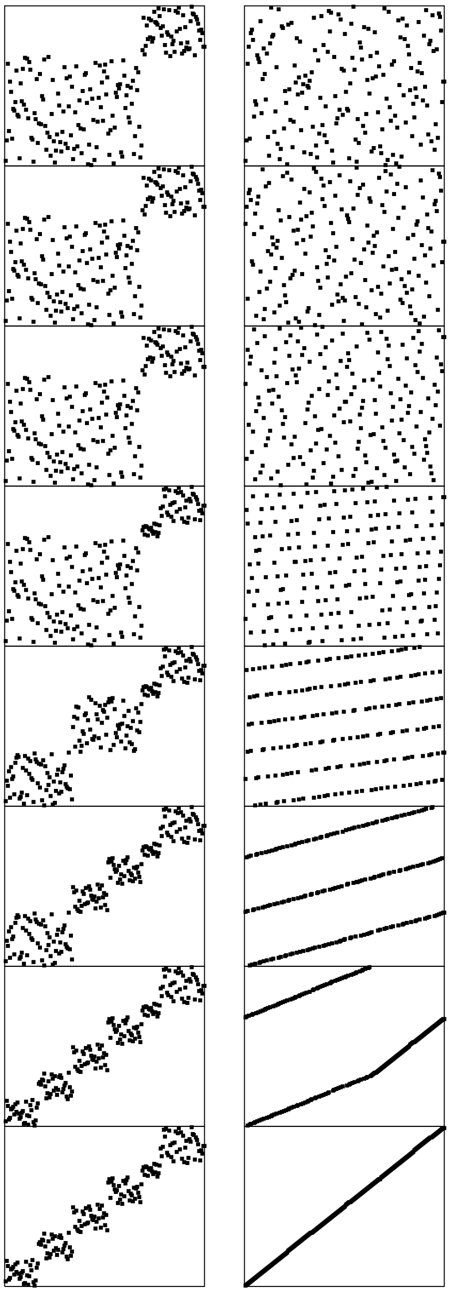
Binary Quicksort

MSD radix sort

Three-way radix Quicksort

LSD radix sort

Sorting in linear time



Bits and digits

Extracting bits 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
```

Easy to extract 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: x = 0X61626364
     digit(x, 2) = (x >> 8) & 255 = c
```

```
Ex: Single-bit access: bitsbyte = 1
     digit(x, 11) = (x >> 20) & 1 = 0
```

Binary Quicksort

Partition file into two pieces
- all keys with first bit 0
- all keys with first bit 1

Sort two pieces recursively

```
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);
}
```

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, BUT keys one bit shorter

Binary Quicksort (continued)

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

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

Random bits?

should sort out after lgN bits examined

Nonrandom bits?

take bigger chunks

MSD radix sort

Partition file into M buckets
- all keys with first byte 0
- all keys with first byte 1
- all keys with first byte 2

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

Sort M pieces recursively

Take M=2^bitsbyte

Tradeoff
- large M
  - space for buckets
  - (too many empty buckets)
- small M
  - too many passes
  - (too many keys per bucket)

Upper bound on running time:

(bitsword/bitsbyte)*N

(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

Method for sorting file on keys with R values
Basis for radix sorts

Basic method

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

```
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];
}
```

Keys must be integers (in a small range)!
Need one counter for each different key value

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 code

Modify key-indexed counting to access
bytes as keys

Add a loop to do recursive calls

```
#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);
  }
```

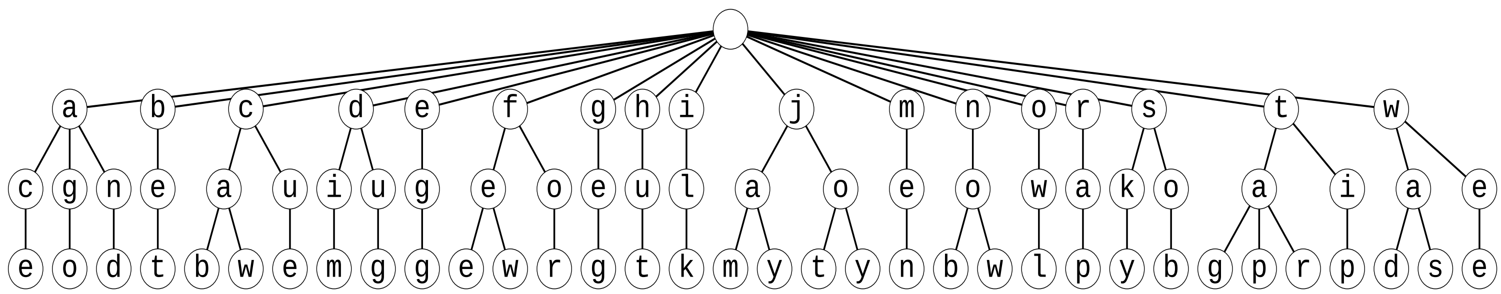
Variable-length keys terminated with 0 (C strings)

* remove first if statement
* remove first recursive call

Recursive structure of MSD radix sort

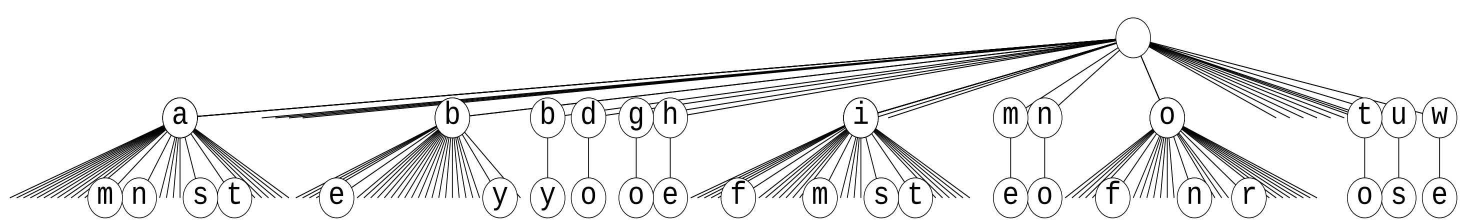
Tree structure to describe recursive call

Paths in tree give keys



Problem: algorithm touches empty nodes

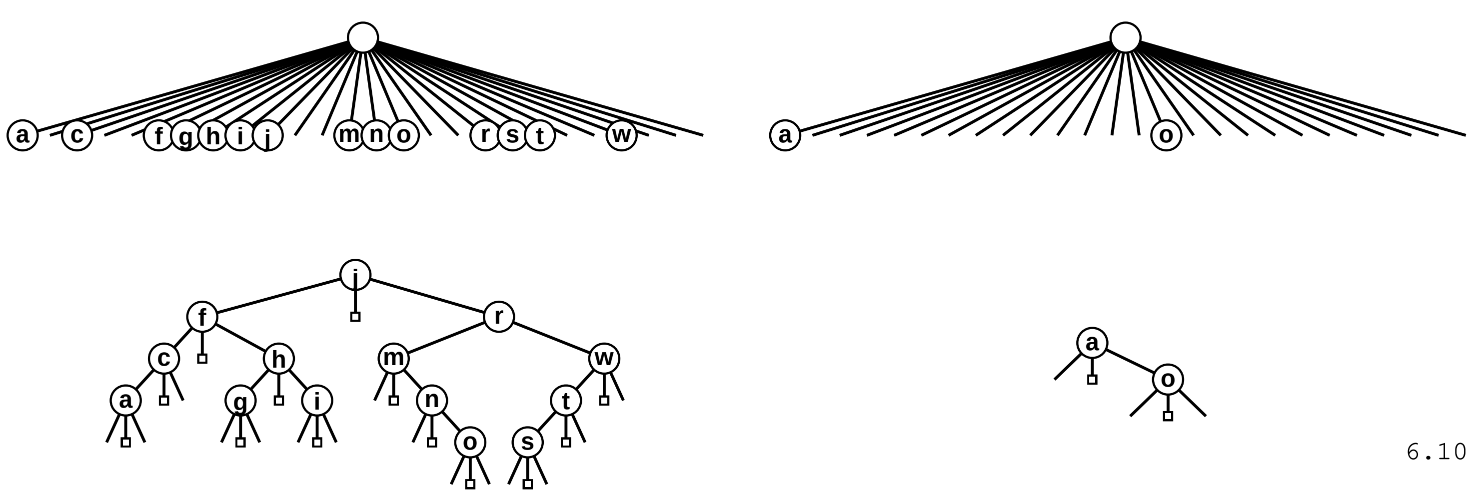
Tree can be as much as M times bigger



Could use linked lists for nonempty buckets

Better solution:

3-way Quicksort partition on bytes



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
```

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)

```
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];
  }
}
```

Running time: N*(bitsword/bitsbyte)

Disadvantage:

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

LSD radix sort example

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

Proof that it works:

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

Another proof that it works:

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

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