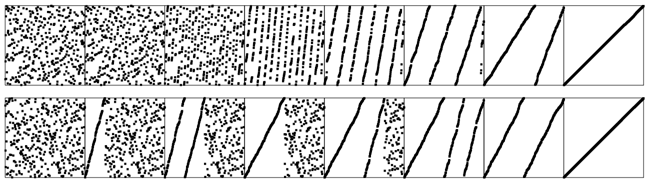
COS 226 Lecture 4: Mergesort

Prototypical divide-and-conquer algorithm



Why study mergesort?

Guaranteed to run in O(N log N) steps
Method of choice for linked lists

Drawback:
- Linear extra space
- (can only sort half the memory)

An "optimal" sorting method

Leads us to consider
- recurrence relationships
- computational complexity
- deep hacking
- fractals

Merging two sorted files

```
#define T Item
merge(T c[], T a[], int N, T b[], int M )
{ int i, j, k;
   for (i = 0, j = 0, k = 0; k < N+M; k++)
     {
       if (i == N) { c[k] = b[j++]; continue; }
       if (j == M) { c[k] = a[i++]; continue; }
       if (less(a[i], b[j]))
          c[k] = a[i++]; else c[k] = b[j++];
     }
}
```


Merging example

							A	R	S	T	G	I	N
A	R	S	T	G	I	N							
	R	S	T	G	I	N	A						
	R	S	T		I	N	A	G					
	R	S	T			N	A	G	I				
	R	S	T				A	G	I	N			
		S	T				A	G	I	N	R		
			T				A	G	I	N	R	S	
							A	G	I	N	R	S	T

Trivial computation?

Try doing it without using linear extra space

Abstract inplace merge

Easier for calling routine to assume merge is inplace
- assume files to be merged are both in arg array
- copy files into temp array
- merge back into arg array

Trick: reverse second file when copying
- avoids special tests for ends of arrays

```
               A R S T G I N
A R S T N I G
  R S T N I G  A
  R S T N I    A G
  R S T N      A G I
  R S T        A G I N
    S T        A G I N R
      T        A G I N R S
               A G I N R S T
```


Abstract inplace merge implementation

```
Item aux[maxN];
merge(Item a[], int l, int m, int r)
{ int i, j, k;
   for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
   for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
   for (k = l; k <= r; k++)
      if (less(aux[i], aux[j]))
          a[k] = aux[i++]; else a[k] = aux[j--];
}
```

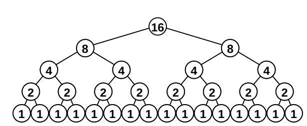

Mergesort example

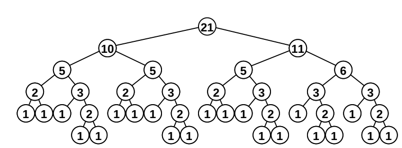
```
A S O R T I N G E X A M P L E
A S
    O R
A O R S
        I T
            G N
        G I N T
A G I N O R S T
                E X
                    A M
                A E M X
                        L P
                        E L P
                A E E L M P X
A A E E G I L M N O P R S T X
```


Mergesort implementation

```
void mergesort(Item a[], int l, int r)
{
   int m = (r+l)/2;
   if (r <= l) return;
   mergesort(a, l, m);
   mergesort(a, m+1, r);
   merge(a, l, m, r);
}
```

Tree structures describe merge file sizes





Recurrences

Direct relationship to recursive programs

- (most programs are "recursive")

Easy telescoping recurrences

- T(N) = T(N-1) + 1   T(N) = N
- T(2^n) = T(2^(n-1)) + 1   T(N) = lg N if N=2^n

Short list of important recurrences

- T(N) = T(N/2) + 1   T(N) = lg N
- T(N) = T(N/2) + N   T(N) = N
- T(N) = 2T(N/2) + 1   T(N) = N
- T(N) = 2T(N/2) + N   T(N) = N lg N

Details in Chapter 2

Mergesort analysis

THM: Mergesort uses N lg N comparisons

Proof:

- From code,
  T(N) = 2T(N/2) + N
- For N = 2^n (n = lg N),
  T(2^n) = 2T(2^(n-1)) + 2^n
- Divide both sides by 2^n
  T(2^n)/2^n = T(2^(n-1))/2^(n-1) + 1
- Telescope:
  T(2^n)/2^n = n
- Therefore,
  T(N) = N lg N

Exact for powers of two, approximate otherwise

Guaranteed worst-case bound

Mergesort and numbers

THM: Number of compares used by Mergesort for

- is the same as

number of bits in the binary representations
of all the numbers less than N (plus N-1).

Proof: They satisfy the same recurrence

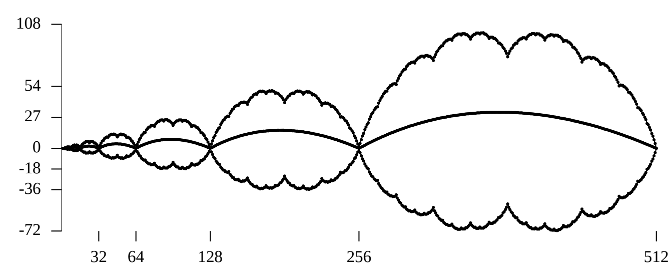
- C(2N) = C(N) + C(N) + 2N
- C(2N+1) = C(N) + C(N+1) + 2N+1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Mergesort and fractals

Divide-and-conquer algs exhibit erratic periodic behavior

number of bits in numbers less than N
= number of 0 bits + number of 1 bits
= (N lg N)/2 + periodic term
+ (N lg N)/2 + periodic term
= N lg N + periodic term



Divide-and-conquer

Basic algorithm design paradigm

``Master Theorem`` for analyzing algorithms

- T(N) = aT(N/b) + N^c(lg N)^d

Interested in learning more?

- Stay tuned for a few more in CS 226
- Take CS 341, CS 423
- Read ``Introduction to the Analysis of Algs`` by Sedgewick and Flajolet

Computational Complexity

Framework to study efficiency of algorithms

Machine model: count fundamental operations

Average case:

- predict performance (need input model)

Worst case:

- guarantee performance (any input)

Upper bound: algorithm to solve the problem
Lower bound: proof that no algorithm can do

Complexity studies provide

- starting point for practical implementations
- indication of approaches to be avoided

Complexity of sorting

N lg N comparisons necessary and sufficient

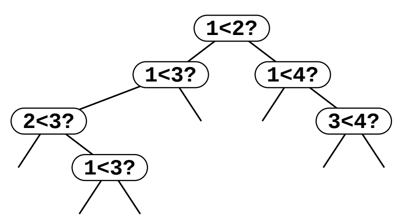
Upper bound: N lg N (Mergesort)
Lower bound: N lg N - N/(ln 2) + lg N

THM: All comparison-based sorting methods
must use at least N lg N comparisons

Proof:

COMPARISON TREE (all sequences of comparisons)

Comparison tree for sorting



Path from root to leaf describes operation of sorting algorithm on given input

Claim 1: at least N! leaves
Claim 2: height at least lg N!
Claim 3: (Stirling's formula for lg N!)

- height at least N lg N - N/(ln 2) + lg N

Caveat: what if we don't use comparisons??
Stay tuned for radix sort

Mergesort without move

Alternative to abstract inplace merge

```
void mergesort(T a[], T b[], int l, int r)
{ int m = (l+r)/2;
  if (r-l <= 10)
    { insertion(a, l, r); return; }
  mergesort(b, a, l, m);
  mergesort(b, a, m+1, r);
  merge(a+l, b+l, m-l+1, b+m+1, r-m);
}
void sort(Item a[], int l, int r)
{ int i;
  for (i = l; i <= r; i++) aux[i] = a[i];
  mergesort(a, aux, l, r);
}
```


Deep hacking on Mergesort inner loop

CODE OPTIMIZATION: Improve performance by tuning code

- concentrate on inner loop

For mergesort,

- Avoid move with recursive argument switch
- Avoid sentinels with "up-down" trick

Combine the two? Doable, but mindbending

Can make mergesort almost as fast as quicksort

- mergesort inner loop: compare, store, two incs
- quicksort inner loop: compare, inc

Bottom-up mergesort

Pass through the file

- merge adjacent subfiles
- size doubles each time through

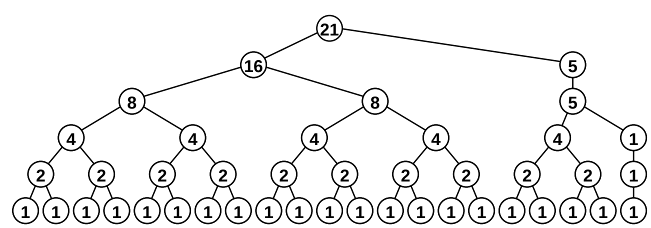
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S													
		O	R											
				I	T									
						G	N							
								E	X					
										A	M			
												L	P	
A	O	R	S											
				G	I	N	T							
								A	E	M	X			
												E	L	P
A	G	I	N	O	R	S	T							
								A	E	E	L	M	P	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Bottom-up mergesort implementation

```
void mergesort(Item a[], int l, int r)
{ int i, m;
  for (m = 1; m < r-l; m = m+m)
    for (i = l; i <= r-m; i += m+m)
      merge(a, i, i+m-1, min(i+m+m-1, r));
}
```

Different set of merges than for top-down

- unless N is a power of two



Merging linked lists

Problem: sort data on a linked list
(rearrange list so items are in order)

```
typedef struct node *link;
struct node { Item item; link next; };
```

First step: merge implementation

```
link merge(link a, link b)
{ struct node head; link c = &head;
  while ((a != NULL) && (b != NULL))
    if (less(a->item, b->item))
      { c->next = a; c = a; a = a->next; }
    else
      { c->next = b; c = b; b = b->next; }
  c->next = (a == NULL) ? b : a;
  return head.next;
}
```


Top-down list mergesort

Split, sort, and merge

```
link mergesort(link c)
{ link a, b;
  if (c->next == NULL) return c;
  a = c; b = c->next;
  while ((b != NULL) && (b->next != NULL))
    { c = c->next; b = b->next->next; }
  b = c->next; c->next = NULL;
  return merge(mergesort(a), mergesort(b));
}
```


Bottom-up list mergesort

Cycle through a circular list

```
link mergesort(link t)
{ link u;
  for (initQ(); t != NULL; t = u)
    { u = t->next; t->next = NULL; putQ(t); }
  t = getQ();
  while (!emptyQ())
    { putQ(t); t = merge(getQ(), getQ()); }
  return t;
}
```