CS314: Recursive Algorithms

Note Title 9/4/2013

Announcements

- Homework 1 up (probably) tomorrow
  - due next Friday
  - written this time
    (HW2 will be oral grading)
- Turn in HW 0 now
- Picnic next week!
  (4pm next Wed)

Another (old) example: Merge Sort

According to Knuth, suggested by von Newmann around 1945.

Idea: ① Subdivide array into 2 parts.
② Recursively sort the 2 parts.
③ Merge them back together.

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Divide:	S	O	R	T	I	N	G	E	X	A	M	P	L
Recurse:	I	N	O	S	R	T	A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	S	R	T	X

Key: If thinking recursively only step 3 is non-trivial!

```
MergeSort(A[1..n]):
  if (n > 1)
      m ← ⌊n/2⌋
      MergeSort(A[1..m])
      MergeSort(A[m+1..n])
      Merge(A[1..n], m)
```

(Again, avoid unrolling.)

What's my base case here?

size 1 (or less)

How to merge?

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Divide:	S	O	R	T	I	N	G	E	X	A	M	P	L
Recurse:	I	N	O	S	R	T	A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	S	R	T	X

$K=1$

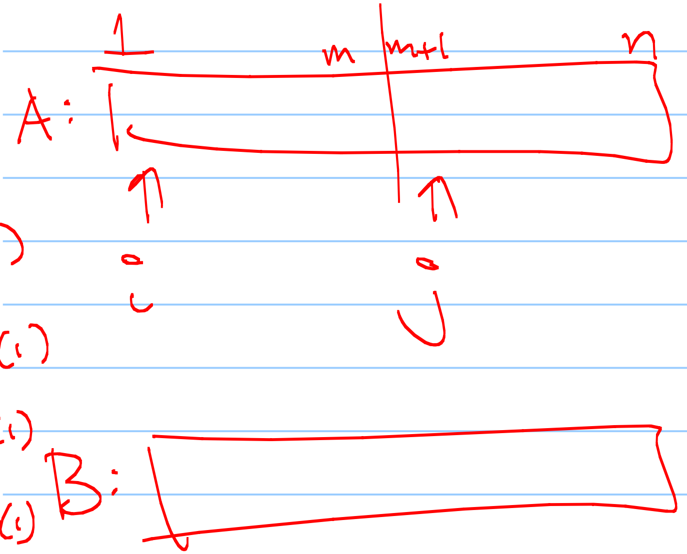
Write a subroutine:

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
    if j > n
        B[k] ← A[i]; i ← i+1        O(1)
    else if i > m
        B[k] ← A[j]; j ← j+1        O(1)
    else if A[i] < A[j]
        B[k] ← A[i]; i ← i+1        O(1)
    else
        B[k] ← A[j]; j ← j+1        O(1)

  for k ← 1 to n
    A[k] ← B[k]
```

n times → $O(n)$

$O(n)$



$\rightarrow O(n) + O(n) + 3 = O(n)$

Proof of correctness: Actually, 2 of them.

Lemma: MERGE results in sorted order.

given 2 sorted subarrays.

pf: induction on sizes of A[i..m] and A[j..n]

Base case A[i...m] empty
or A[j...n] empty
correct thing is to just keep taking elements from non-empty list, which is what our first 2 ifs do.

IH: Alg is correct for smaller inputs.

Cont: IS: Consider $A[i \ldots m]$ & $A[j \ldots n]$ not base case, so neither is empty.

Since these are sorted, first element of one of them must be minimum (or else not sorted).

Alg: Finds min & moves it to B. & shrinks one of the subarrays by 1.

By IH, correct on rest.

$\boxed{M}$

Pf that mergesort works.

Base case: size 0 or 1, do nothing

IH: works for lists of size $k < n$

IS: Consider $A[1 \ldots n]$

By IH, $A[1 \ldots m]$ & $A[m+1 \ldots n]$ are sorted.

Now need to show A ends in sorted order, which it does since Merge works (by prev. lemma).

Runtime: Let $M(n)$ = runtime on n elements of mergesort

$$M(0) = M(1) = O(1)$$

(1 comparison)

$$M(n) = \underline{3} + M\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + M\left(\left\lceil \frac{n}{2} \right\rceil\right) + \underbrace{(\text{runtime of Merge})}_{O(n)}$$

$$= 2M\left(\frac{n}{2}\right) + O(n)$$

$$\Rightarrow M(n) = O(n \log n)$$

Multiplication: fundamental

```
         31415962
       × 27182818
        251327696
       31415962
     251327696
    62831924
  251327696
 31415962
219911734
62831924
853974377340916
```

or

x	y	$prod$
		0
123	+456	= 456
61	+912	= 1368
30	~~1824~~	
15	+3648	= 5016
7	+7296	= 12312
3	+14592	= 26904
1	+29184	= 56088

$O(n^2)$

How fast? (n-~~bit~~ digit number)

$O(n^2)$

Divide & conquer strategy:

$$(10^m a + b)(10^m c + d) = 10^{2m} \underline{ac} + 10^m (bc + ad) + \underline{bd}$$

$$\underbrace{963}_{a},\underbrace{245}_{b} \quad m=3 \quad = 10^3 \cdot 963 + 245$$

How to turn this into an algorithm?

$$\underbrace{624}_{c}\,\underbrace{197}_{d}$$

$ac = 963 \cdot 624$

$bd = 245 \cdot 197$

$bc + ad =$

Pseudo code

```
MULTIPLY(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m
      d ← ⌊y/10^m⌋;  c ← y mod 10^m
      e ← MULTIPLY(a, c, m)
      f ← MULTIPLY(b, d, m)
      g ← MULTIPLY(b, c, m)
      h ← MULTIPLY(a, d, m)
      return 10^{2m} e + 10^m (g + h) + f
```

O(1)

4 recursive calls

← adding n bit #'s: $O(n)$

Runtime?

$T(1) = 1$

$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

$aT\left(\frac{n}{b}\right) + f(n)$

$f(n)$ to $n^{\log_b a}$

$O(n)$ $\quad n^{\log_2 4}$

$\quad\quad\quad\;\; \parallel$

$\quad\quad\quad\;\; n^2$

$= O(n^2)$

Hrm... not better after all...

Another trick:

$$ac + bd - (a-b)(c-d) = \underbrace{bc + ad}$$

Why will this help?

Recall:

$$(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc+ad) + bd$$

Now, pseudo code only has 3 recursive calls!

```
FastMultiply(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m
      d ← ⌊y/10^m⌋;  c ← y mod 10^m
      e ← FastMultiply(a, c, m)
      f ← FastMultiply(b, d, m)
      g ← FastMultiply(a − b, c − d, m)
      return 10^(2m) e + 10^m (e + f − g) + f
```

!!

Runtime?

$$T(n) = 3T(\tfrac{n}{2}) + O(n)$$

(4n instead of n)

$$T(n) = O(n^{\log_2 3})$$

$$\log_2 3 < 2$$

$$= n^{1.5\ldots}$$

Notes:

- In practice, this is done in binary - replace 10's with 2's.
- This idea can be broken down recursively even further, for an eventual $O(n \log n)$ time.

(Ever heard of Fast Fourier transforms?)