CS314 - More recursive algorithms

Note Title 9/5/2013

Announcements

- HW 1 up, due in class next Friday (may still work in groups)
- Thursday office hours next week may move - stay tuned...

Recursion: Quicksort

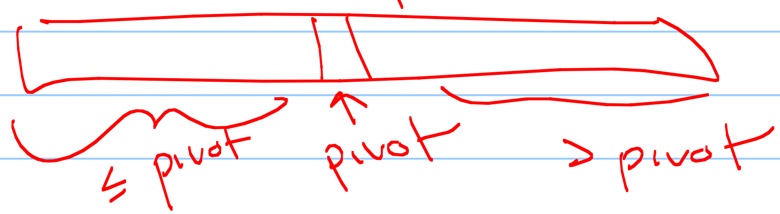
Downside of MergeSort — space!

Hard to do in place.

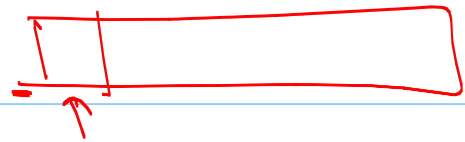
(Also harder to code...)

Simpler divide + conquer: Quicksort

Idea? select a "pivot"



Pseudocode



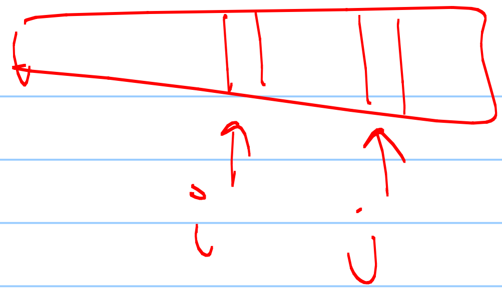
```
QuickSort(A[1..n]):
   if (n > 1)
      Choose a pivot element A[p]
      k ← Partition(A, p)
      QuickSort(A[1..k − 1])
      QuickSort(A[k + 1..n])
```

55 3 2 6 100 91 11

But how to partition?

PARTITION($A[1..n], p$):

  if $(p \neq n)$
    swap $A[p] \longleftrightarrow A[n]$ $O(1)$
  $i \leftarrow 0$; $j \leftarrow n$
  while $(i < j)$
    repeat $i \leftarrow i + 1$ until $(i = j$ or $A[i] \geq A[n])$
    repeat $j \leftarrow j - 1$ until $(i = j$ or $A[j] \leq A[n])$
    if $(i < j)$
      swap $A[i] \longleftrightarrow A[j]$
  if $(i \neq n)$
    swap $A[i] \longleftrightarrow A[n]$
  return i



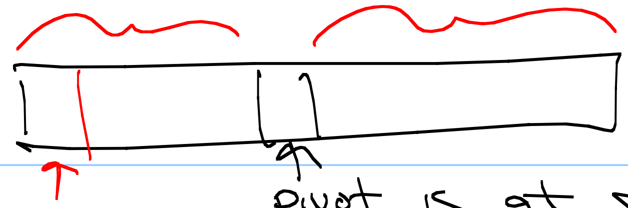
$\Rightarrow \Theta(n)$

Proof: (sketch)

Very similar to mergesort:

- First show partition works given any array as input + any p.
- Then use induction on entire array.

Analysis:



pivot is at spot k

Depends on choice of pivot:

$$Q(n) = O(n) + Q(k-1) + Q(n-k)$$

Worst case:

$$Q(n) = O(n) + \cancel{Q(0)} + Q(n-1)$$

$$= n + (n-1) + \dots + 1$$

$$= O(n^2)$$

A recursive strategy: backtracking

Idea: Build up a solution iteratively.

Setting: an algorithm needs to try multiple possibilities.

Strategy: make a recursive call on each possibility.

Downside: slow

Ex: Subset Sum

Given a set X of positive integers and a target t, is there a subset of X which sums to t?

Ex: $X = \{8, 6, 7, 5, 3, 10, 9\}$

$t = 15$

Yes

$\{8, 7\}$
$\{10, 5\}$
$\{6, 9\}$
$\vdots$

Ex: $t = 20$

Yes

$\{8, 7, 5\}$

How could we look at things incrementally (or recursively)?

Set up: take an item $x \in X$.

Two possibilities:

- x is in subset
- not

$X - \{x\}$
t

$X - \{x\}$
$t - x$

Careful —
that is the recursive case!

What is missing?
(ie when are we done?)

- if X is empty, can't hit any $t \neq 0$

- if $t < 0$, done

Pseudo code

```
SubsetSum(X[1..n], T):
    if T = 0
        return True
    else if T < 0 or n = 0
        return False
    else
        return (SubsetSum(X[2..n], T) ∨ SubsetSum(X[2..n], T − X[1]))
```

(tail recursion)

Correctness

IS: Either $X[1]$ is in subset or not (if subset summing to t exists).

My code tries both possibilities.

→ Base case: $T=0 \Rightarrow$ true
$\emptyset$ sums to 0

$T<0$ → no set sums to T

X is empty — can't hit positive T

Runtime:

$$S(n) = 5 + 2S(n-1)$$

$$\hookrightarrow \quad \underbrace{s_n = 2s_n}_{x-2=0} + 5$$

poly of degree 0

$$S(n) = c_1 2^n + c_2$$

(use base cases to check $c_1 \neq 0$)

$$c_1 \cdot 2^0 + c_2 = 1$$
$$c_1 2^1 + c_2 = 5$$

$$S(n) = O(2^n)$$

Side note: brute force

- try every subset (2^n)
- for each, sum values & check $= T$

$O(n)$

$\Rightarrow O(n2^n)$