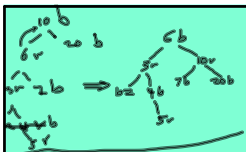


Augmenting structures

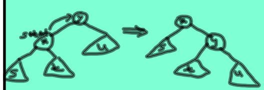
Dynamic Selection

BST

x.size = # nodes in subtree rooted at x

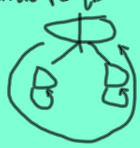

```
kth smallest(T, k)
if T.size < k then error
else if T.left.size == k-1
    return T(root)
else if T.left.size > k-1
  return kth smallest(T.left, k)
else
  return kth smallest(T.right, k-T.left.size-1)

Rank(x, T)
  r := 1
  if x.left ≠ nil then
     r := r + x.left.size
  while x.p ≠ nil & x = x.p.right
     r := r + x.p.left.size + 1
     x := x.p
return r
```



Dynamic Programming

Divide + conquer —



$F_0 := 0$

$F_1 := 1$

$F_n := F_{n-1} + F_{n-2}$ $(n>1)$

```
fib(n)
  if n ∈ {0,1} then
     return n
  else
     return fib(n-1) + fib(n-2)
```

Array $F[0..n]$

$F[i] = F_i$

```
F[0] := 0
F[1] := 1
For i := 2 to n
  F[i] := F[i-1] + F[i-2]
return F[n]
```

Optimization Problem (usually)

"Optimal Substructure"

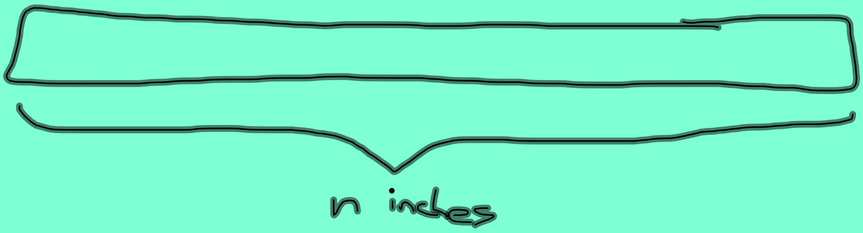
optimal solution can be obtained from optimal sol'ns to subproblems, which can be obtained independently

If space of all subproblems is reasonably small, then dynamic prog is appropriate:

Compute a table of subsolutions bottom up, new entries depending on previously computed entries.

Oct 11-12:49 PM

Rod cutting



i	0	1	2	$\cdots\cdots$	n
r_i	3	4	7	7	

How to cut the rod to maximize revenue?

Let r_n = max revenue for cutting an n-inch rod.

$$r_0 = 0$$

$$r_n = \max(\underline{p_n}, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$$