Recursion
for decision problems

Knapsack problem

Given : a set S of n objects each with a weight w_i and value v_i , $0 \leq i \leq n - 1$ and a knapsack with weight capacity W

Question : What is the maximum value that can be obtained by packing the knapsack with objects from S such that the sum of weights of chosen objects does not exceed W ?

- To answer this question you don't need to examine every possible subset of S
- **Optimality principle** : Suppose in an optimal solution, after choosing objects from subset of first m objects, capacity C remains. Then the choices for the remaining n-m objects must be an optimal solution for the subproblem of knapsack capacity C.

Knapsack examples

- Can we use a greedy approach ?
- Consider weights = {6,4,2, 1} and
  values = {6,4,5,3} and
  capacity of knapsack W = 6
- Fill objects by weights, smallest first.

  weights chosen = {1,2}, total value = 3+5= 8
- Fill objects by values, largest first.

weights chosen = {6}, total value = 6

- Best solution :

Weights chosen = {4,2}, total value = 4+5 = 9

Recursion subproblem for knapsack

- **Given**: A subset of remaining objects indexed from k (i.e. k, k+ 0,….n- 1) and remaining capacity c of knapsack where 0 ≤ c ≤ W,
- **Needed** : What is the maximum value $F_k(c)$ that can be obtained ?

Answer to original problem : Set k = 0 and c = W

Base step:

$F_n(c)$ = 0 (as no more objects available)

Recursion step (k=0, 1,2,…n- 1):

$F_k(c) = F_{k+1}(c)$ if w_i > c (object cannot be chosen)

$F_k(c) = \max(v_i + F_{k+1}(c - w_i), F_{k+1}(c))$ otherwise

i.e. choose between uselt or loselt decisions

Need $F_0(W)$ to answer the original problem.

```
public static int knapsack(int [] weights, int [] values, int start, int
capacity) {
    if (start == weights.length) {
        return 0;
    }
    if (weights[start] > capacity) {
        return knapsack(weights, values, start+ 1, capacity);
    }
    return Math.max(knapsack(weights, values, start+ 1, capacity),
            values[start] + knapsack(weights, values, start+ 1, capacity-
weights[start]));
}

public static int knapsack(int [] weights, int [] values, int capacity) {
    return knapsack(weights, values, 0, capacity);
}
```

Some of recursive calls starting from K=0, C = 6

Order of calls shown with alphabets, dotted lines show call returns, numbers show values returned.

Remaining knapsack capacity C ----→