Lecture # 12

- Greedy Algorithms
- Knapsack Problem

- **Definition:** An algorithm that always takes the best immediate, or local (optimum choice at each stage with the hope of finding the global optimum) solution while finding an answer. Greedy algorithms find the overall, or globally, *optimal solution* for some *optimization problems*, but may find less-than-optimal solutions for some instances of other problems.
- If there is no greedy algorithm that always finds the optimal solution for a problem, one may have to search (exponentially) many possible solutions to find the optimum. Greedy algorithms are usually quicker, since they don't consider the details of possible alternatives.

- How can one tell if a greedy algorithm will solve a particular optimization problem?

- There is no way in general, but the greedy choice property and optimal substructure are the two key ingredients.

Greedy-choice property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
- In other words when we consider which choice to make, we make a choice that looks best in the current problem, without considering the results of from sub problems.

- Here is where greedy algorithms differ from dynamic programming.
- In DP, we make a choice at each step, but the choice *usually depends on the solutions to the sub problems*. Consequently, we typically solve DP problems in a *bottom up* manner, progressing from smaller sub problems to larger sub problems.
- In greedy algorithm, we make whatever choice seems good at the moment and then solve the sub problem arising after the choice is made. The choice made by the *greedy algorithm may depend on the choices so far*, but it *cannot depend* on any future choices or on the *solutions to the sub problems*.

- Thus unlike dynamic programming, which solves the sub problems bottom up, greedy algorithm usually progress in top-down fashion, making one greedy choice after another, reducing each given problem instance to a smaller one.

Optimal Substructure

- A problem exhibits Optimal Substructure if an optimal solution to the problem contains within it optimal solutions to sub problems.
- This property is the key ingredient of DP as well as Greedy algorithms.
- All we need to argue is that an optimal solution to the sub problem, combined with the greedy choice already made, yields an optimal solution.

Knapsack Problem

- **Statement :** A thief robbing a store and can carry a maximal weight of w into their knapsack. There are n items and i^{th} item weigh w_i and is worth v_i dollars.

- What items should thief take?

- There are two versions of the problem.

- Fractional knapsack problem:

  The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of x_i of item i, where $0 \leq x_i \leq 1$.

  - Exhibit greedy choice property.
    - Greedy algorithm exists.
  - Exhibit optimal substructure property.
    - ?????

- The optimal solution to this problem is to sort by the value of the item in decreasing order. Then pick up the most valuable item which also has a least weight.
- First, if its weight is less than the total weight that can be carried. Then deduct the total weight that can be carried by the weight of the item just pick.
- The second item to pick is the most valuable item among those remaining. Keep follow the same strategy until thief cannot carry more item (due to weight).

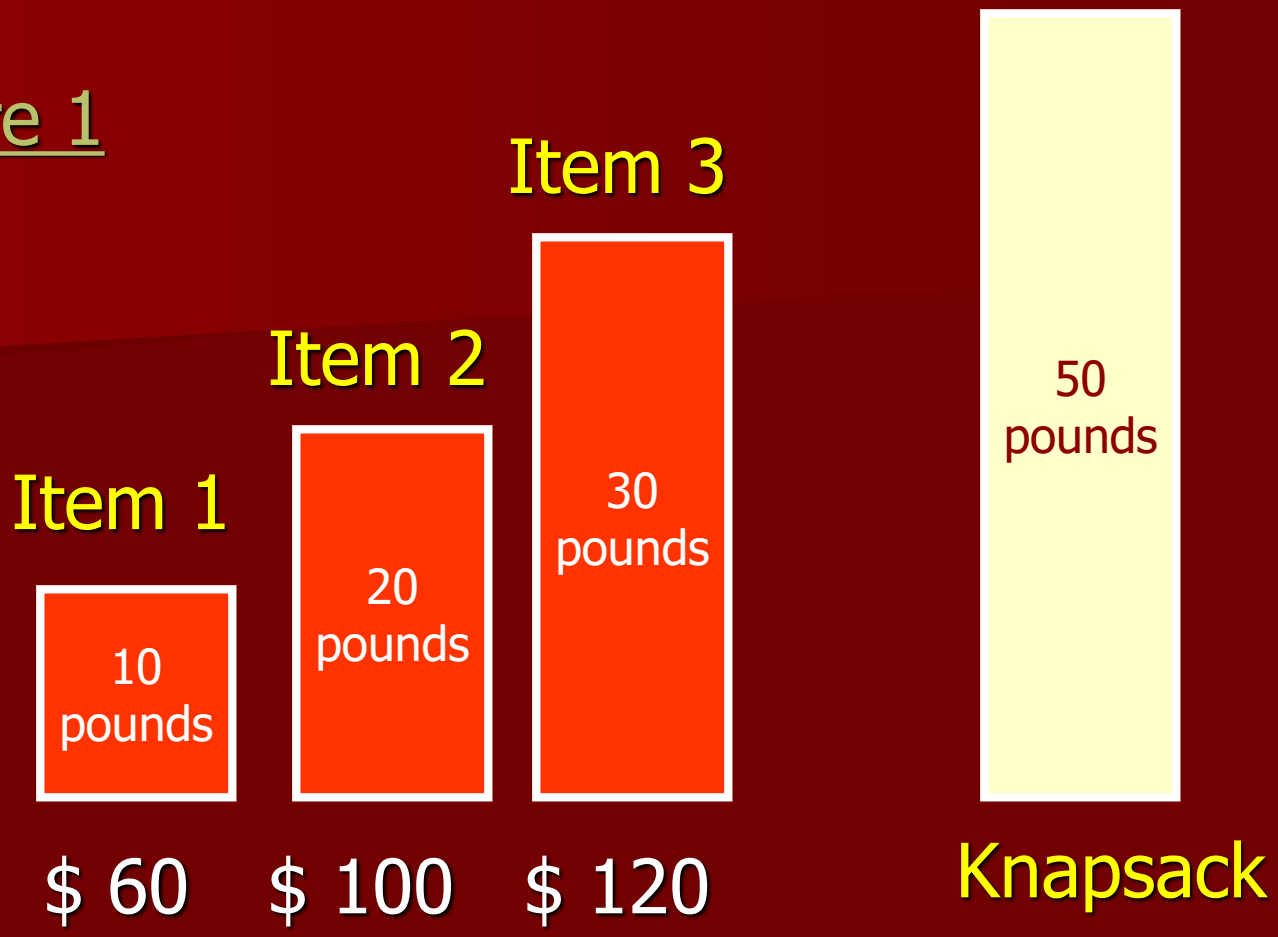
- The items has to be sorted into decreasing order of v_i/w_i . Therefore, the total time including the sort is in $O(n \log n)$.

- 0-1 knapsack problem:

  The setup is the same, but the items may not be broken into smaller pieces, so thief may decide either to take an item or to leave it (binary choice), but may not take a fraction of an item.

  - Exhibit No greedy choice property.
    - No greedy algorithm exists.
  - Exhibit optimal substructure property.
    - Only dynamic programming algorithm exists.

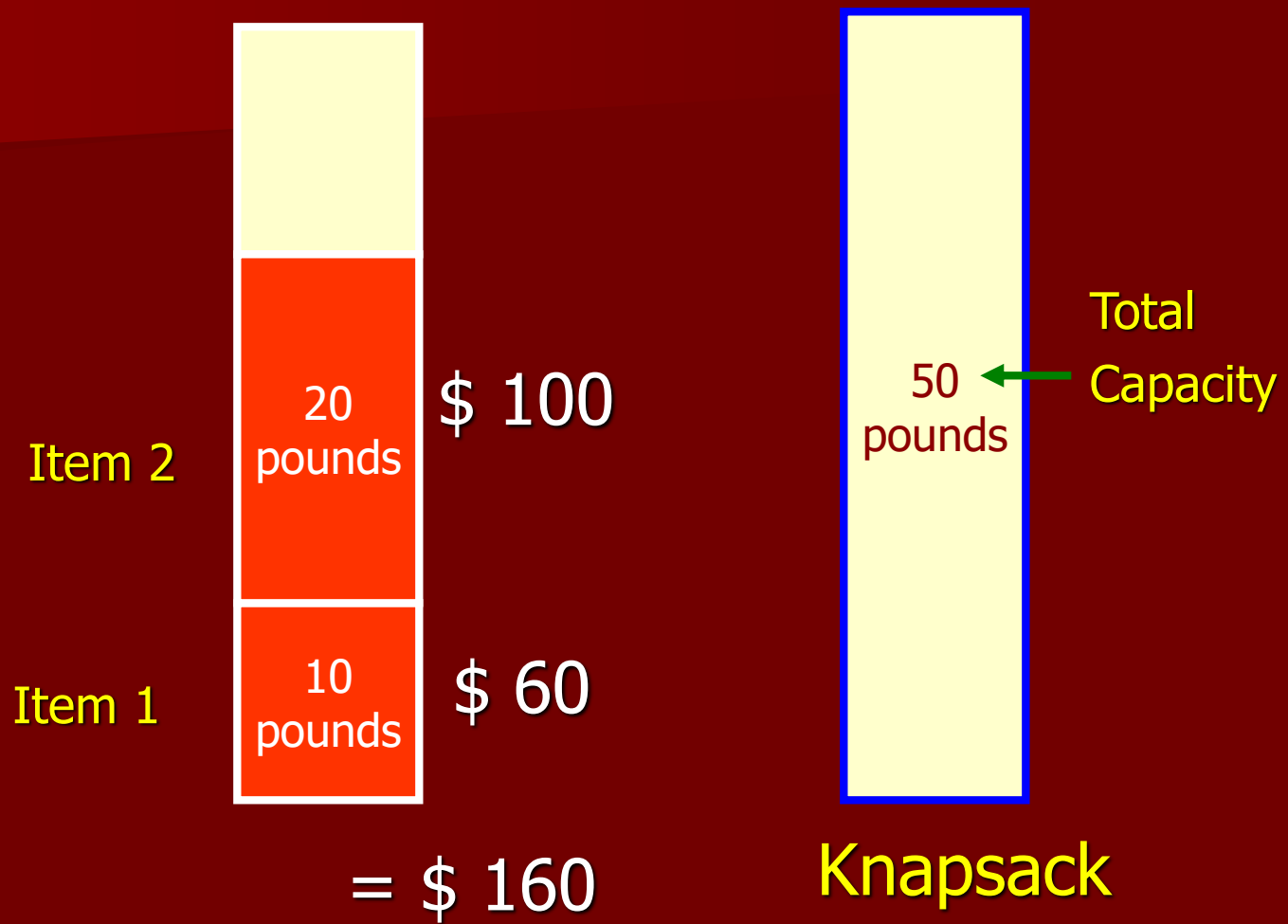
Figure 1



- What would be Greedy strategy to fill the knapsack?

Greedy Strategy – (0-1 Knapsack)

Sub Optimal solution



- No Greedy solution exists but DP solution exists.

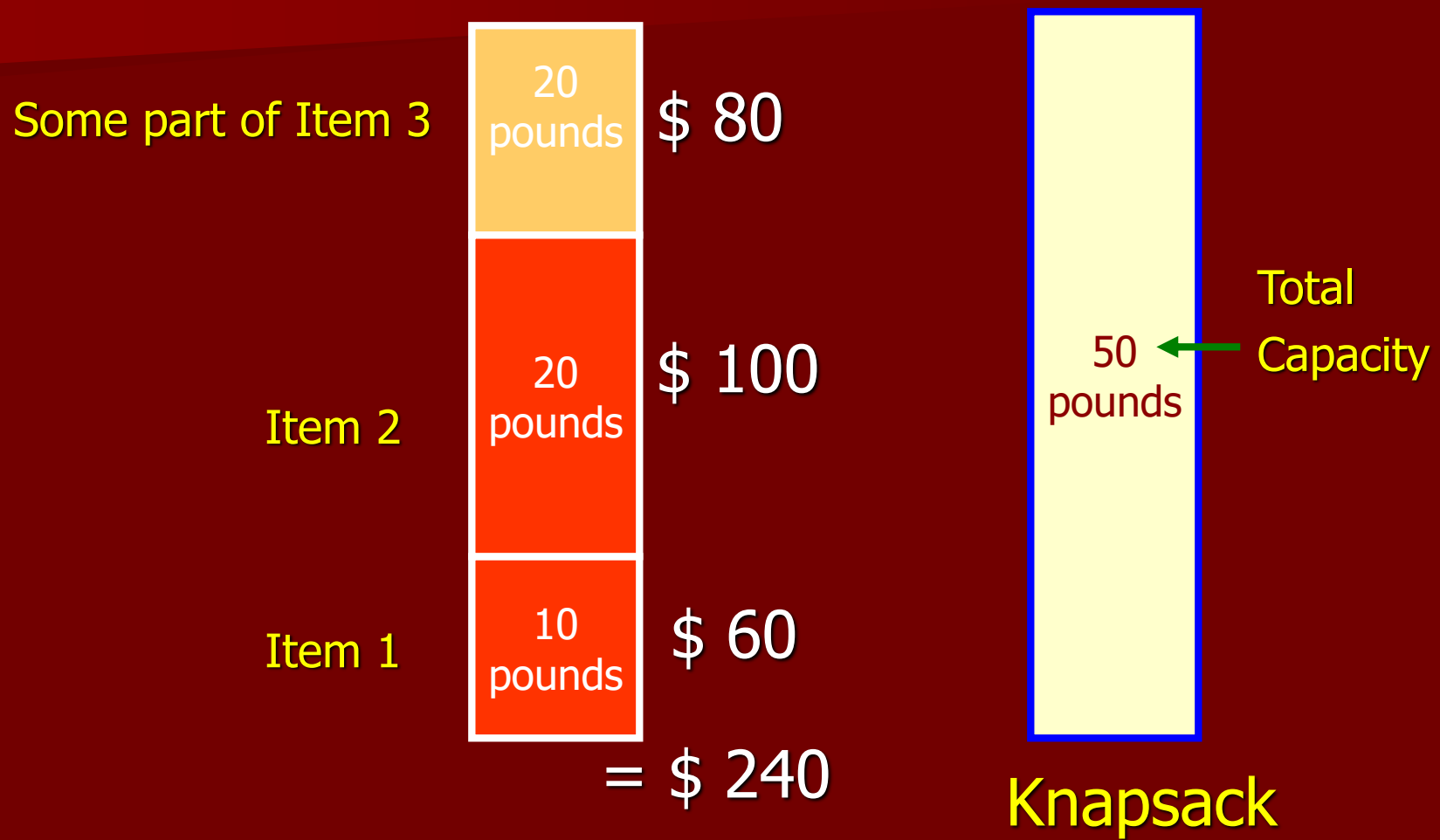
- Optimal solution would take item 2 and item 3. the two possible solutions that involve item 1 are sub optimal

Sub Optimal solution →

- Optimal solution would take item 2 and item 3. the two possible solutions that involve item 1 are sub optimal

Optimal solution →

Greedy Strategy – (Fractional Knapsack)



- Greedy solution exists.

- For fractional version of above problem , the greedy strategy yields an optimal solution.

- Taking item 1 first doesn't work in the 0-1 problem, because thief is unable to fill his knapsack to the capacity it has.

- In 0-1 problem when we consider an item for inclusion in the knapsack, we must compare the solution to the sub problem in which the item is included with the solution to the sub problem in which the item is excluded before we can make the choice.

- The problem formulated in this way gives rise to many overlapping sub problems – a hallmark of DP, and *indeed*, DP can be used to solve 0-1 problem.