THE GREEDY PARADIGM

What is a greedy algorithm?

Optimization Problems

- For most optimization problems you want to find, not just ***a*** solution, but the ***best*** solution.
- A ***greedy algorithm*** sometimes works well for optimization problems. It works in phases. At each phase:
    - You take the best you can get right now, without regard for future consequences.
    - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum.

THE GREEDY PARADIGM

Commit to choices one-at-a-time,

never look back,

and hope for the best.

Greedy Approach

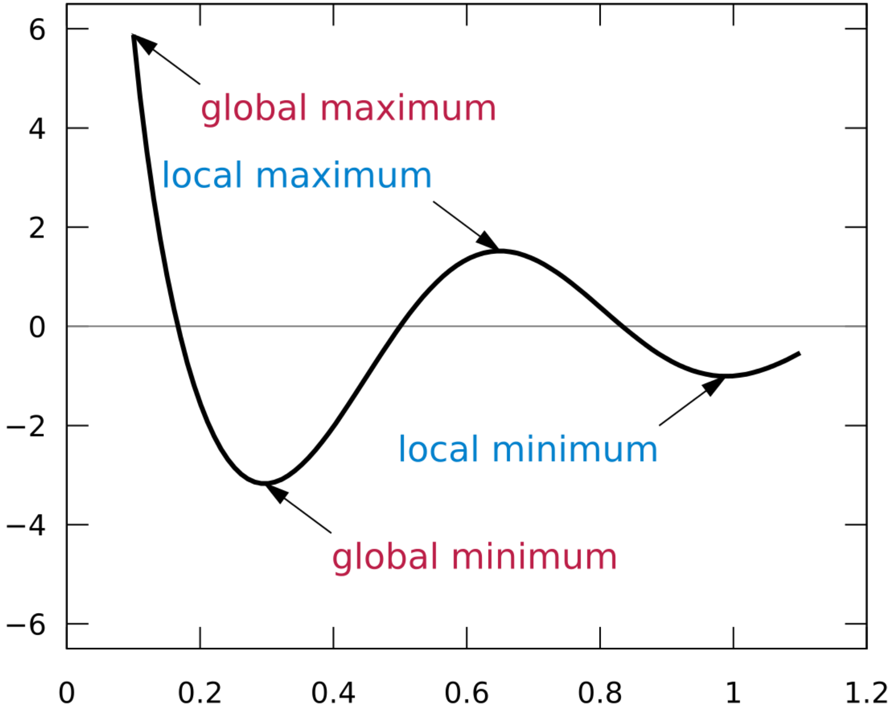
- Like dynamic programming, used to solve optimization problems.
- Problems exhibit optimal substructure (like DP).
- Problems also exhibit the **greedy-choice** property.
    - When we have a choice to make, make the one that looks best *right now*.
    - Make a **locally optimal choice** in hope of getting a **globally optimal solution**

Greedy Approach

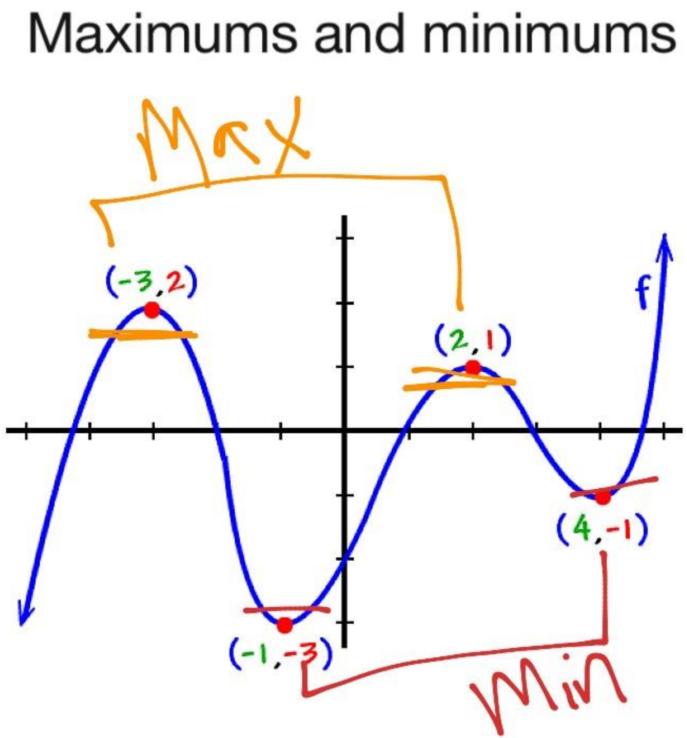
- Does **greedy-choice** property lead us to optimize solution?
- 

- **Greedy algorithms** mostly (but not **always**) fail to find the globally **optimal solution** because they usually **do** not operate exhaustively on all the data.

Greedy Approach



Greedy Approach



ACTIVITY SELECTION

An example where greedy works!

ACTIVITY SELECTION: THE TASK

Input: n activities with start times and finish times
Constraint: All activities are equally important, but you can only do 1 activity at a time!
Output: A way to maximize the number of activities you can do

Activity Selection problem: Greedy Approach

An Activity Selection Problem (Conference Scheduling Problem)

- **Input: A set of activities S = $\{a_1, \ldots, a_n\}$**
- Each activity has start time and a finish time
  - $a_i = (s_i, f_i)$
- Two activities are compatible if and only if their interval does not overlap
- **Output: a maximum-size subset of mutually compatible activities**

Activity Selection problem: Greedy Approach

Here are a set of start and finish times

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

What is the maximum number of activities that can be completed?

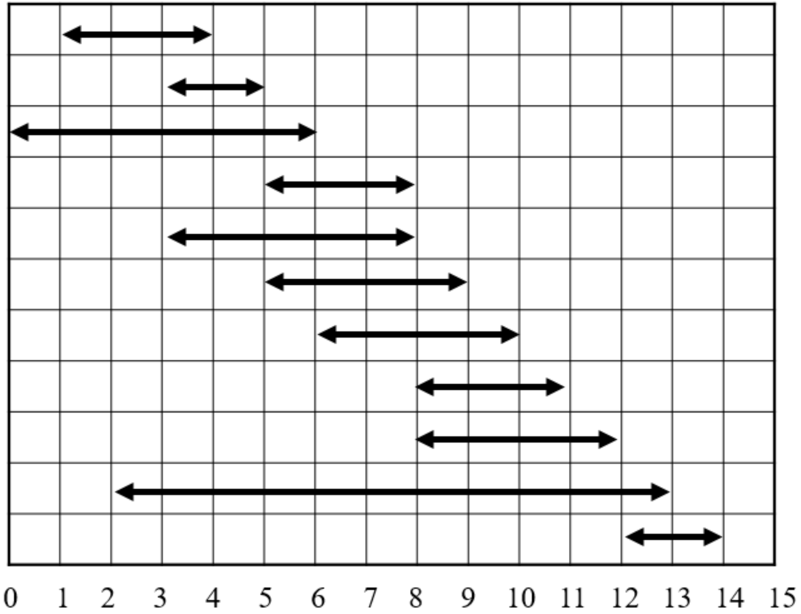
Activity Selection problem: Greedy Approach

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of s_j.
- [Earliest finish time] Consider jobs in ascending order of f_j.
- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

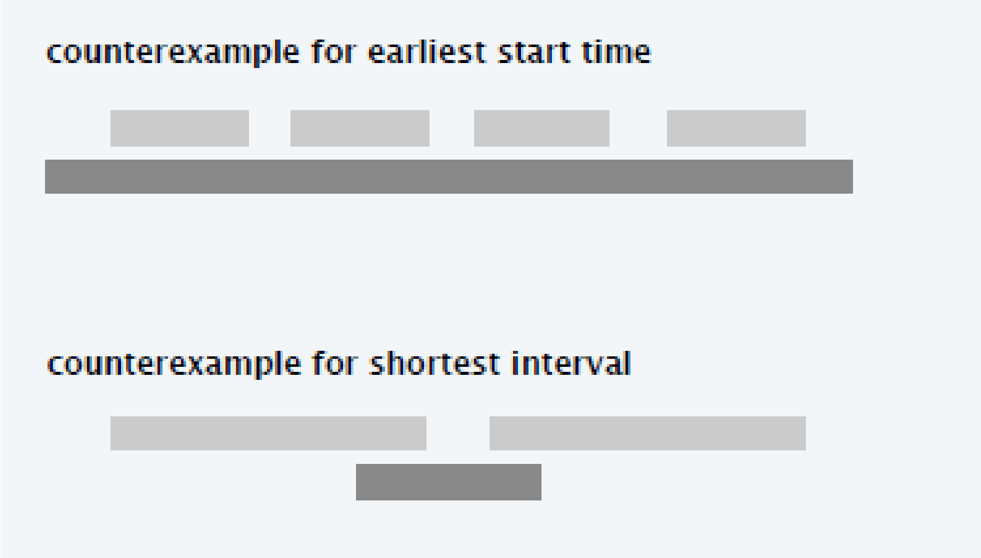
Activity Selection problem: Greedy Approach

- All activities :



Activity Selection problem: Greedy Approach

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.

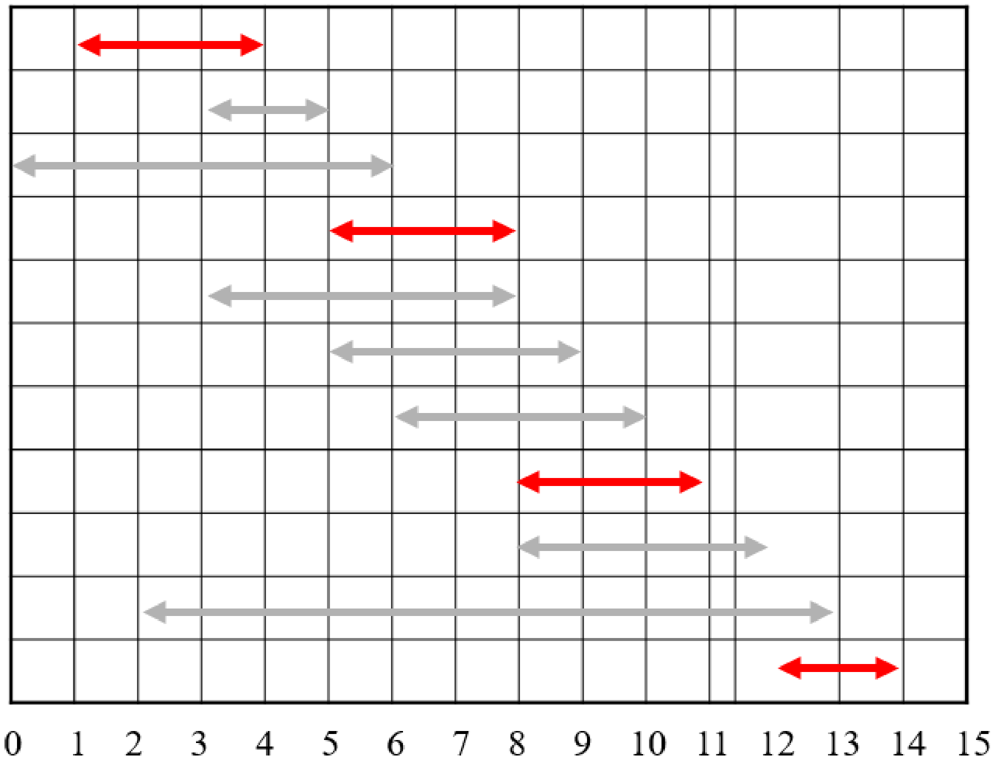


Activity Selection problem: Greedy Approach

- Greedy approach is :
  - Select the activity with the earliest finish
  - Eliminate the activities that could not be scheduled
  - Repeat!

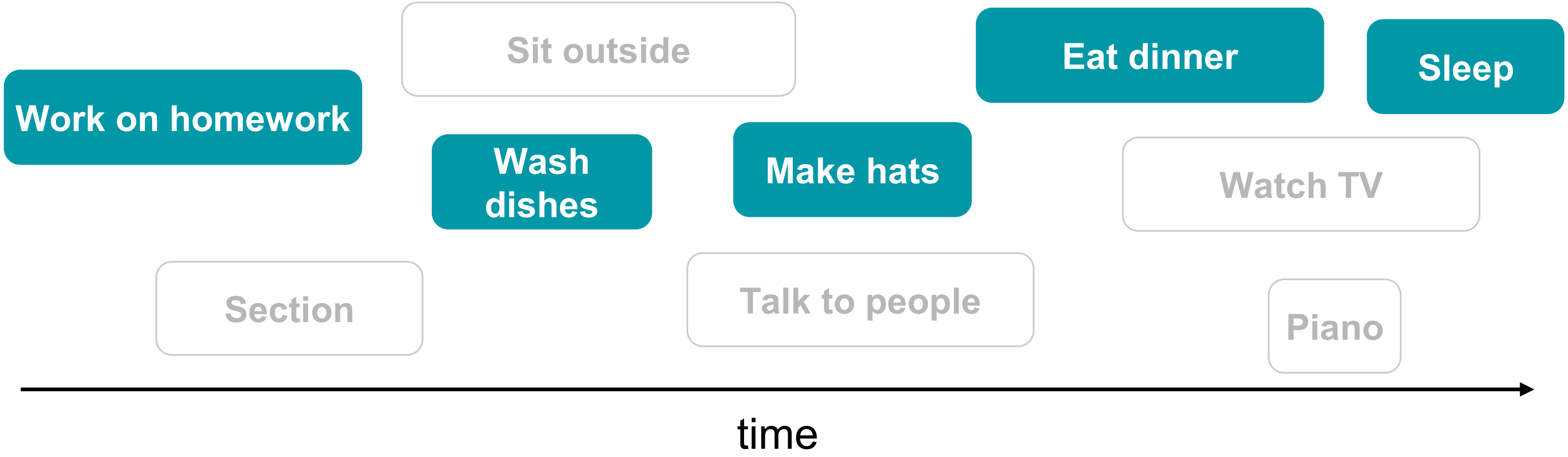
Activity Selection problem: Greedy Approach

- Selected activities:



OUR GREEDY ALGORITHM

Pick an available activity with the smallest finish time & repeat



EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish time so that $f_1 \leq f_2 \leq \ldots \leq f_n$

$A \leftarrow \phi$ ← set of jobs selected

FOR $j = 1$ TO n

  IF job j is compatible with A

    $A \leftarrow A \cup \{j\}$

RETURN A

ACTIVITY SELECTION: PSEUDOCODE

```
ACTIVITY_SELECTION(activities A with start and finish times):
  A = MERGESORT_BY_FINISHTIMES(A)
  result = {}
  busy_until = 0
  for a in A:
    if a.start >= busy_until:
      result.add(a)
      busy_until = a.finish
  return result
```

Runtime: O(n log n)

WHY IS IT GREEDY?

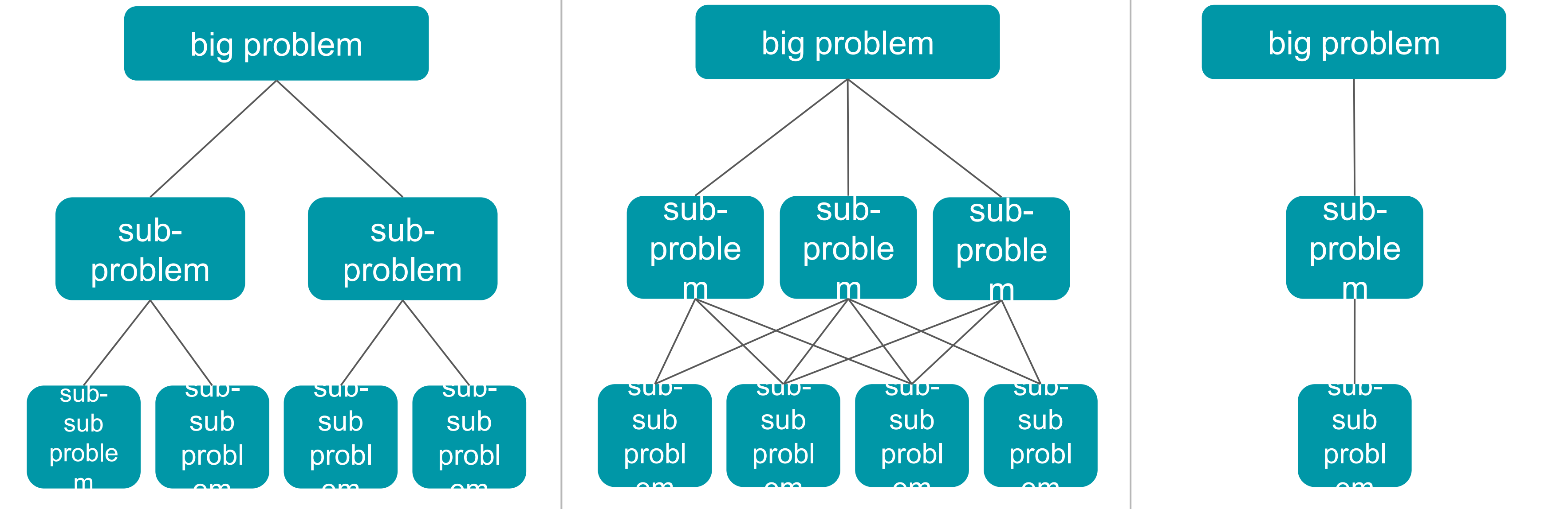
What makes our algorithm a **greedy** algorithm?

At each step in the algorithm, we make a choice (pick the available activity with the smallest finish time) and never look back.

How do we know that this greedy algorithm is correct?
(Proving correctness is the hard part!)

D&C vs. DP vs. GREEDY

DIVIDE-AND-CONQUER	DYNAMIC PROGRAMMING	GREEDY



Coin Change Problem

Another problem with a greedy solution!

Coin Change Problem

- Goal: Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

- A greedy algorithm to do this would be:
  At each step, take the largest possible bill or coin that does not overshoot.
- For US money, the greedy algorithm always gives the optimum solution

Coin Change Problem

- Goal: Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex. 34¢.

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex. $2.89.

Coin Change Problem

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT n coin denominations so that $c_1 < c_2 < \ldots < c_n$

$S \leftarrow \phi$ ⟵ set of coins selected

WHILE $x > 0$

  $k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$

  IF no such k, RETURN "no solution"

  ELSE

    $x \leftarrow x - c_k$

    $S \leftarrow S \cup \{ k \}$

RETURN S

Knapsack Problem : Greedy approach

- Greedy approach is also good for fractional knapsack problem but not for 0/1 knapsack.

- **Fractional knapsack** In which you can take fraction of item if you want

- **0/1 knapsack** In which you can only take complete item or leave it but you cannot take fraction of it

Knapsack Problem : Greedy approach

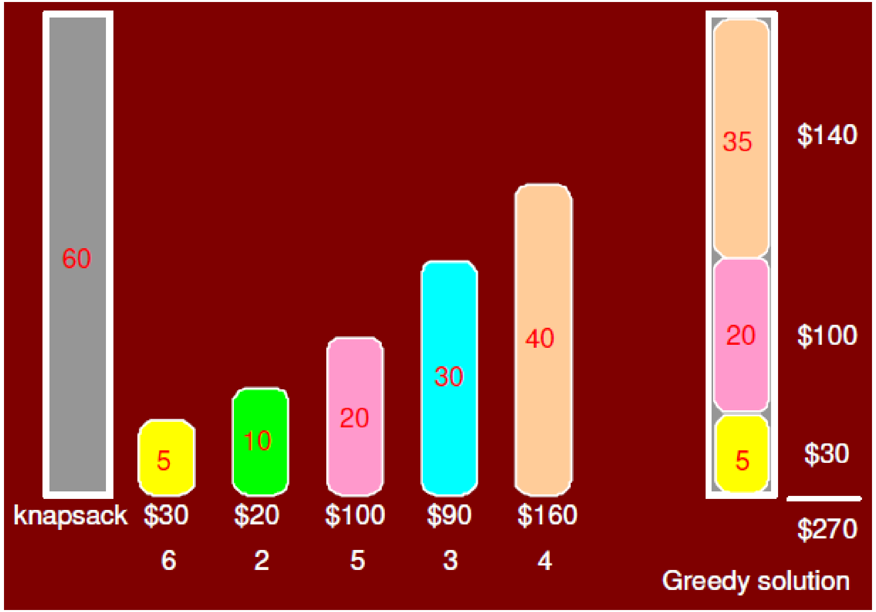
❖ Suppose "i" item has value "v(i)" and weight "w(i)". Capacity of knapsack (bag) is W. Then greedy approach would be :

- Sort in decreasing order of value/weight i-e v(i)/w(i)
- Now start selecting items till you fill the bag.

❖ In fractional knapsack, you utilize complete capacity W because you can take fraction of items but in 0/1 knapsack, you may or may not.

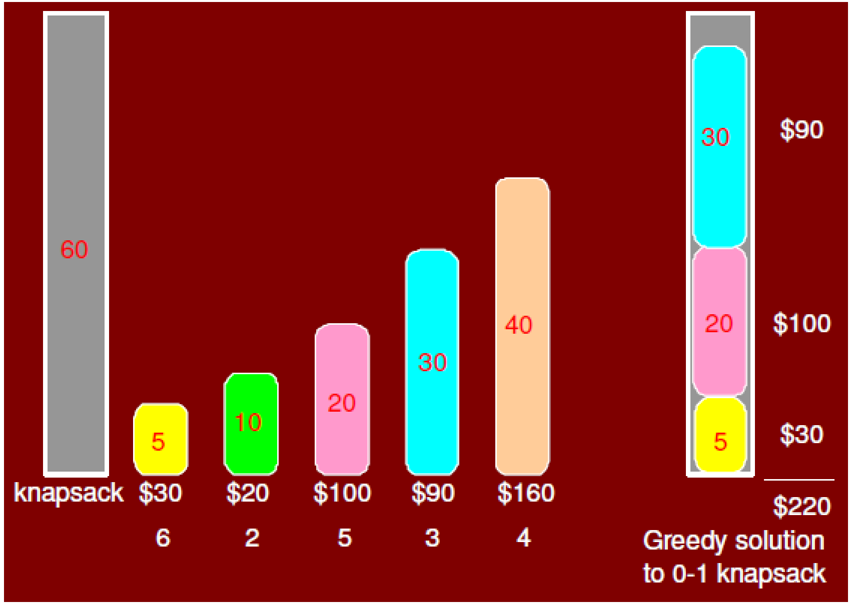
Knapsack Problem : Greedy approach

Greedy/optimal solution to fractional knapsack is :



Knapsack Problem : Greedy approach

- Greedy solution to 0/1 knapsack :



Knapsack Problem : Greedy approach

- Optimal/non-greedy solution to 0/1 knapsack :