Greedy Algorithms

for Optimization Problems

Alexandra Stefan

Optimization Problems

- Knapsack problem
- Job Scheduling (a.k.a. Weighted Interval Scheduling)
- *Other problems – not covered here*
  - *Huffman coding*
    - File compression: encode symbols in a text file s.t. the file has the smallest size possible.
  - *Graphs – Minimum Spanning Tree (MST-Prim's Algorithm) – see later*
- Terminology:
  - *Problem* - general
  - *Variations of a problem* – additional specification to the general problem
  - *Instance of a problem* – specific data (what I use in examples are instances of the problem being discussed problem)

Greedy Method for Optimization Problems

- Optimization problem
  - multiple possible solutions, pick the one that gives the most value or lowest cost
- Discuss
  - Knapsack problem
  - Job Scheduling
- Greedy:
  - Method:
    - **Decide on a criterion** that reflects the measure you are optimizing for (value or cost)
      E.g. for Knapsack maximize the money (value)
    - **take the action that is best now** (out of the current options) **according to your criterion** (i.e. pick a local optimal) and keep it (it will be part of the final solution).
  - **It may miss the optimal solution**
  - **Builds the solution as it goes.**
    - No choice is undone
    - No need to back trace it.
- Terminology:
  - *Problem* - general
  - *Variations of a problem* – additional specification to the general problem
  - *Instance of a problem* – specific data (what I use in examples are instances of the problem being discussed problem)

The 0/1 Knapsack Problem

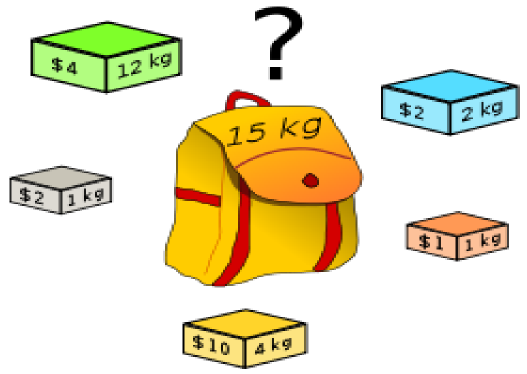


Image from [Wikipedia]()

A thief breaks into a store.

- The maximum total weight that he can carry is *W*. (Here W = 15kg)
- There are *N* types of items at the store. (Here N = 5)
- Each type t_i has a value v_i and a weight w_i.
  (Here, clockwise: ($2, 2kg) , ($1, 1kg) , ($10, 4kg) , ($2, 1kg) , ($4, 12kg)
- What is the <u>maximum total **value**</u> that he can carry out?
  - Cannot load more than maximum weight.
  - Ok if did not fill to full weight(e.g. ok if items add to 12 kg, not 15kg)
- <u>What items should they pick</u> to obtain this maximum value?

0/1 Knapsack - Only one of each object type j.

Other versions:
- a limited number, c_j, of each item type j.
- an unlimited number of each item type j.

0/1 Knapsack – Greedy solution

- What would a greedy thief do for 0/1 Knapsack with W, N, (v_j, w_j)?
  - Criterion: value/weight ratio
  - Method:

```
Sort the N items in decreasing order of ratio,      // O(N lg N), SC: Θ(1)
While (still have objects and knapsack not full )   // O(N)
    if next item, j, fits, take it:
        update total value: res = res + $v_j$          O(1)     O(N)
        update the remaining weight: wrem = wrem – $w_j$
    else skip it
```

  - Does NOT give an optimal solution. See next page.
  - Time complexity: O(N lg N)
  - Space complexity: O(1)

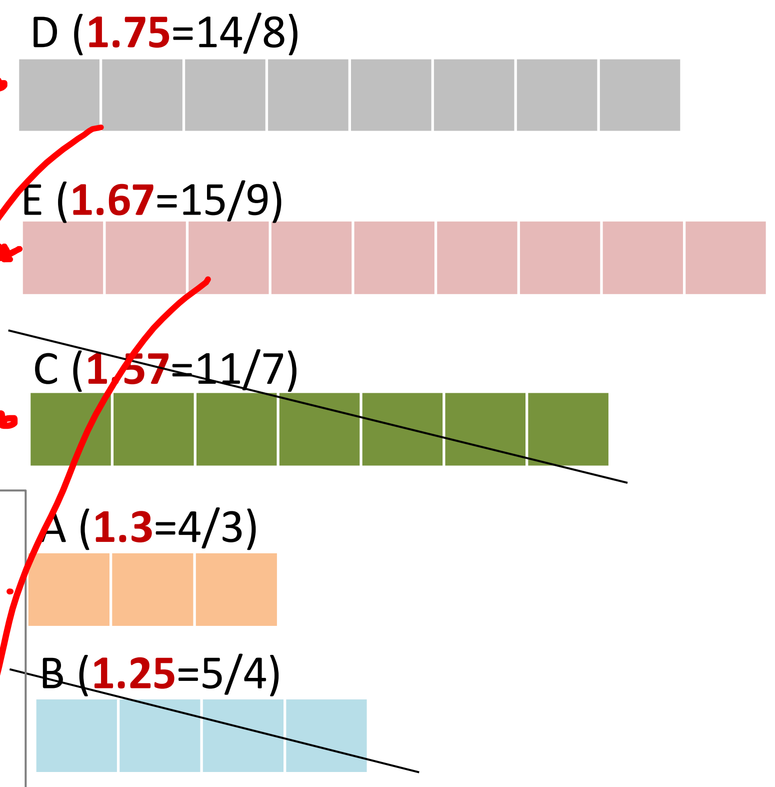
- What if we could take fractions of an item? Does Greedy with ratio give the optimal solution? (Prove or give counter-example)

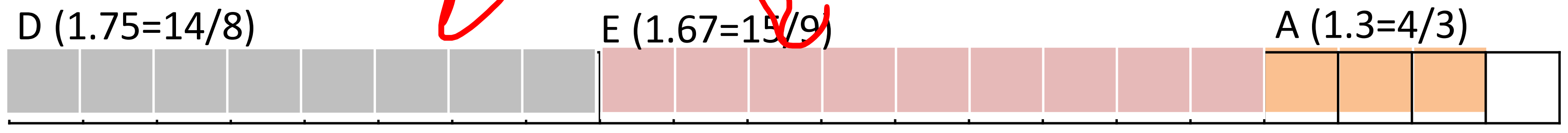
0/1 Knapsack, Greedy, Ratio

Item	A	B	C	D	E
Value	4	5	11	14	15
Weight	3	4	7	8	9
Ratio	4/3= 1.3	5/4= 1.25	11/7= 1.57	14/8= 1.75	15/9= 1.67
Reordered decreasing by **ratio**:	D, 1.75	E, 1.67	C, 1.57	A, 1.3	B 1.25

```
Sort items in decreasing order of ratio,
While (still have objects and knapsack not full )
    if next item, j, fits, take it:
        update total value: $res = res + v_j$
        update the remaining weight $W = W-w_j$
    else skip it
```

Reordered decreasing by **ratio**:





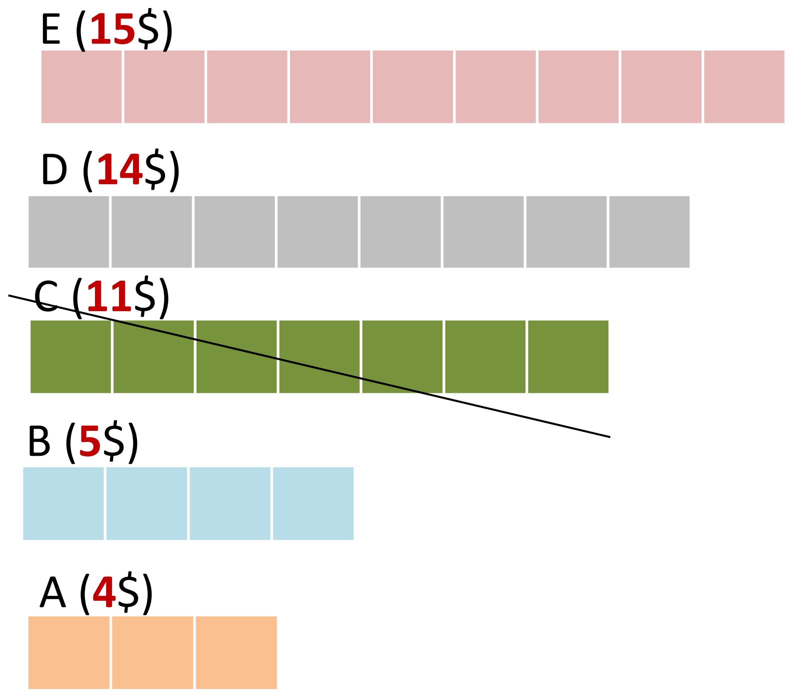
W=21 (knapsack capacity is 21)

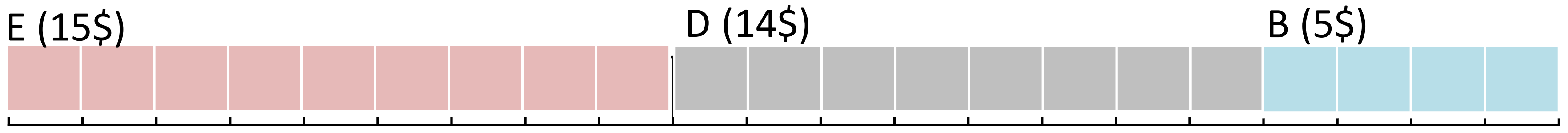
Total value = value(D) + value(E) + value(A) =14 + 15 + 4 =33

0/1 Knapsack , Greedy, **Value**

Item	**A**	**B**	**C**	**D**	**E**
Value	**4**	**5**	**11**	**14**	**15**
Weight	**3**	**4**	**7**	**8**	**9**
Reordered decreasing by **value**: **E, D, C, B, A**					

Reordered decreasing by **value**:





0/1
NF

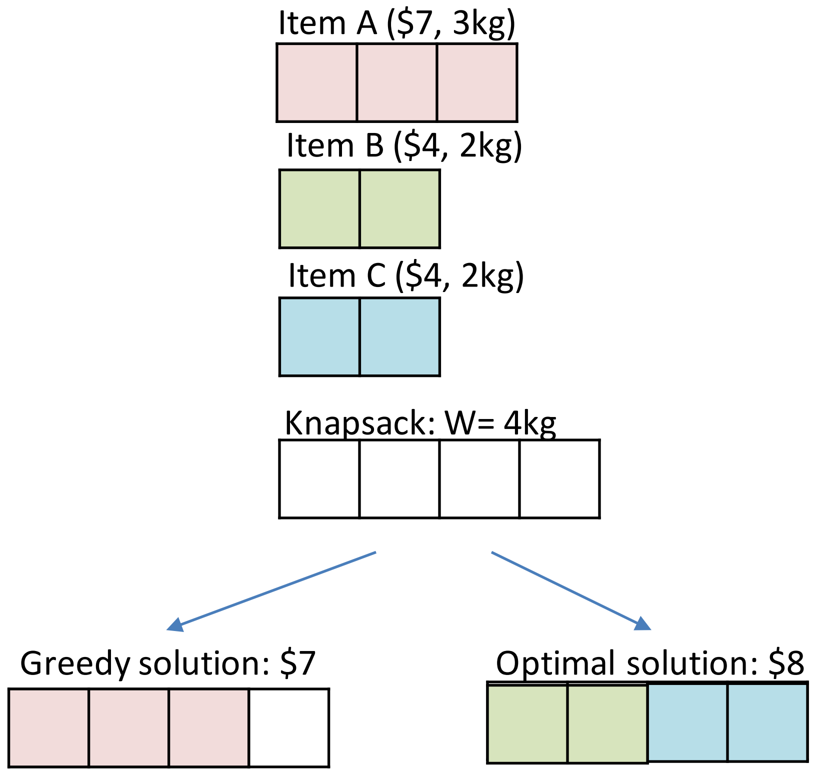
W=21 (knapsack capacity is 21)

Greedy does NOT always give the optimum solution - Proof by counter example

- Goal: construct a **0/1** Knapsack instance where Greedy (with the **ratio**) does not give the optimal answer. Produce: W, N, (v_j,w_j)
- Intuition: We want Greedy to pick only one item,A, when in fact two other items, B and C, can be picked and together give a higher value
  - Start with B and C: make them same weight, same value
  - choose A s.t. it has higher ration than B (or C) , but value smaller than their sum
  - Try W = 2 => fails
  - Try W = 4, B = C = (2kg, $2) => fails
  - Try W = 4, B = C = (2kg, $4) => can make it work:
    - Item B: 2kg, $4 => ratio = 4/2 = 2
    - Item C: 2kg, $4
    - Item A: 3kg, $7 (pick value s.t. ratio > ratio(B) = 2)
    - Knapsack max weight: 4
    - => (W = 4, N = 3, ($7, 3kg), ($4, 2kg), ($4, 2kg)

  - Greedy: picks A, nothing else fits => $7
  - Optimal: B&C => $8

  - !!! You must double-check that Greedy would pick item A
    => check the ratios: 7/3 > 4/2 (2.33 > 2).
    - *If item A had value 5, Greedy would have picked B and C (optimal solution).*

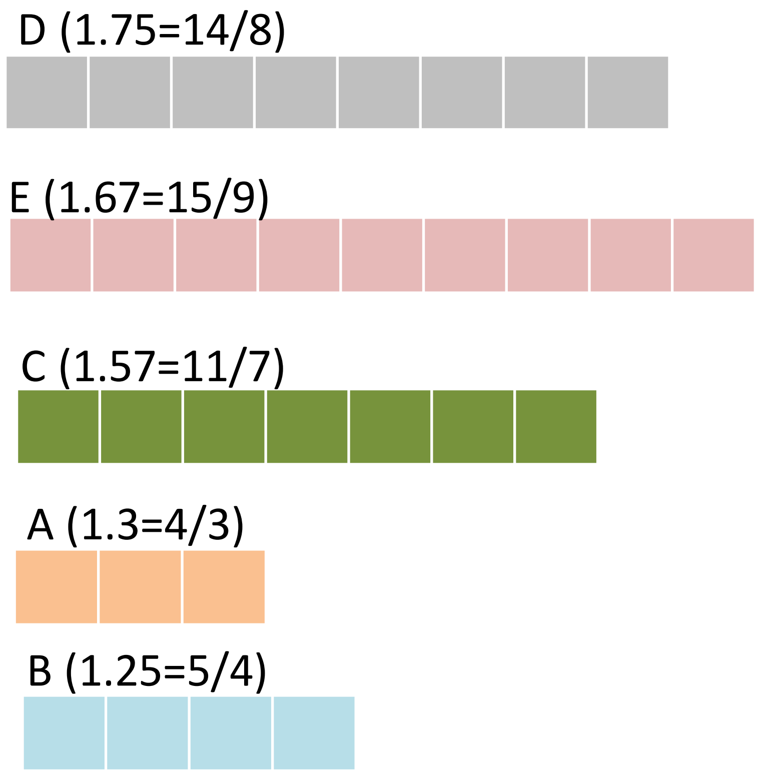


Variation: allow fractions of items => Greedy with ratio is optimal

Item	A	B	C	D	E
Value	**4**	**5**	**11**	**14**	**15**
Weight	**3**	**4**	**7**	**8**	**9**
Ratio	4/3= **1.3**	5/4= **1.25**	11/7= **1.57**	14/8= **1.75**	15/9= **1.67**

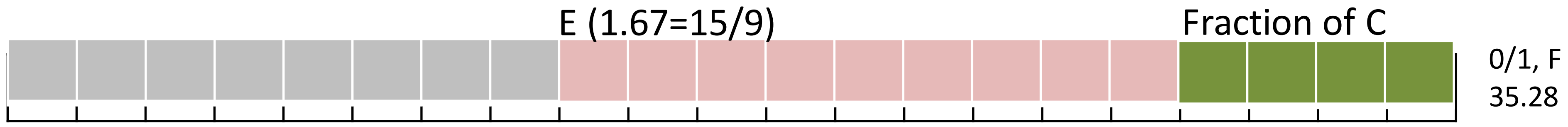
Reordered decreasing by **ratio**: **D, E, C, A, B**

1.75, 1.67, 1.57, 1.3, 1.25



Fractional

Can take fractions of items.



W=21 (knapsack capacity is 21)

0/1 fractional: Total value = value(D) + value(E) + 4 *value(C)/weight(C) = 14 + 15 + 4*(11/7) =35.28

Job Scheduling
(Weighted Interval Scheduling)

Weighted Interval Scheduling
(a.k.a. Job Scheduling)

Problem:

Given n jobs where each job has a start time, finish time and value, (s_j,f_j,v_j) select a subset of them that do not overlap and give the largest total value.

Back-to-back jobs are ok. E. (5,   6, \$3) and g (6,   8, \$2).

Overlapping jobs cannot be both picked. E.g. (2,   5, \$6) and (3, 11, \$5) overlap from 3 to 5.

E.g.:

(start, end, value)
(6,   8, \$2)
(2,   5, \$6)
(3, 11, \$5)
(5,   6, \$3)
(1,   4, \$5)
(4,   7, \$2)

Weighted Interval Scheduling

(a.k.a. Job Scheduling)

E.g.:

(start,	end,	value)
(6,	8,	$2)
(2,	5,	$6)
(3,	11,	$5)
(5,	6,	$3)
(1,	4,	$5)
(4,	7,	$2)

Sort by
finish time

JobId	(start, **end**, value)
0	(
1	(
2	(
3	(
4	(
5	(
6	(

Preprocessing:

- Sort jobs by finish time (increasing).

Weighted Interval Scheduling

(a.k.a. Job Scheduling)

E.g.:

start	end	value
6	8	$2
2	5	$6
3	11	$5
5	6	$3
1	4	$5
4	7	$2

Sort by finish time

JobId	start	**end**	value
0	0	**0**	$0
1	1	**4**	$5
2	2	**5**	$6
3	5	**6**	$3
4	4	**7**	$2
5	6	**8**	$2
6	3	**11**	$5

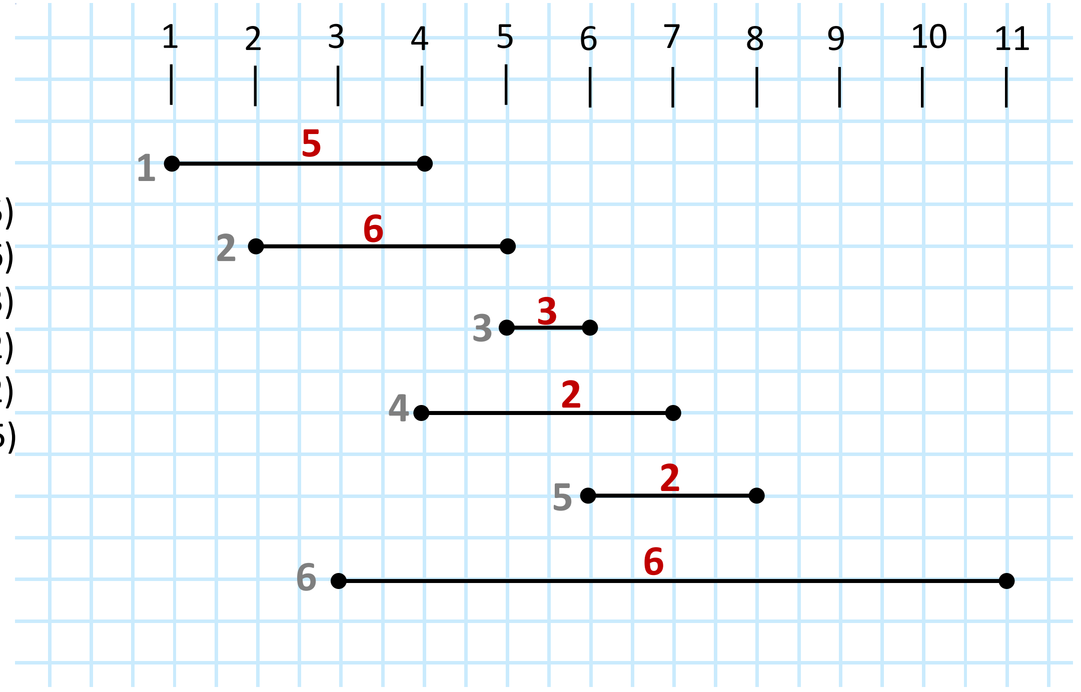
Preprocessing:

- Sort jobs by finish time (increasing).

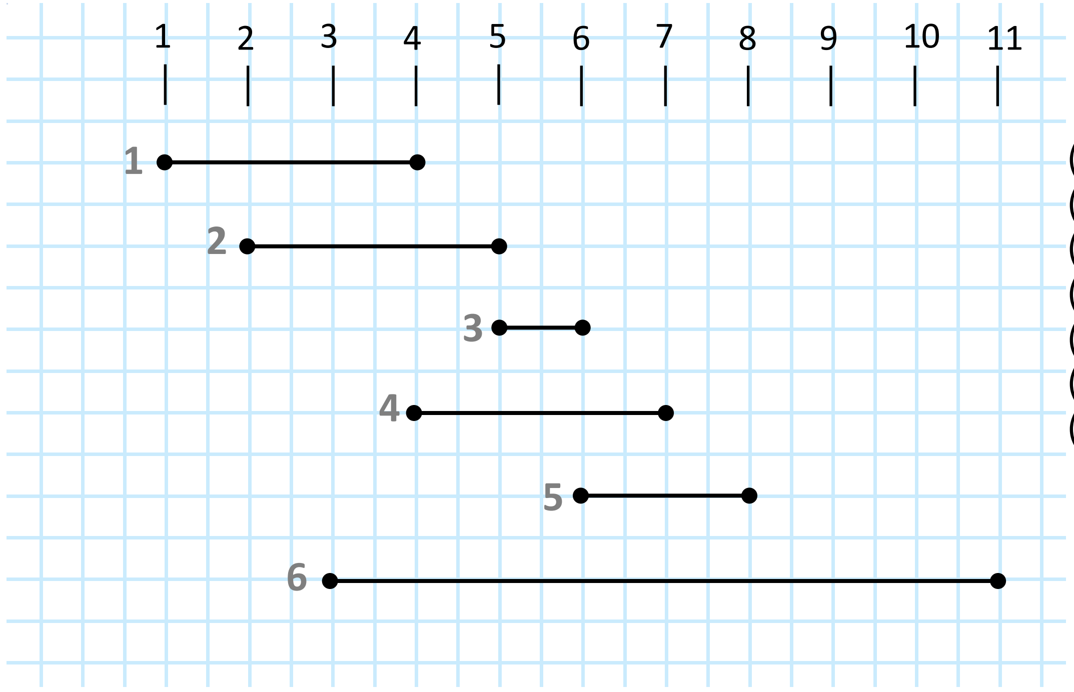
Job Scheduling variations

Jobs have *different values*.
Maximize **total value**.

(s,	f,	$)
(1,	4,	$5)
(2,	5,	$6)
(5,	6,	$3)
(4,	7,	$2)
(6,	8,	$2)
(3,	11,	$5)



Jobs have the *same value*.
Maximize **number of jobs**.



(s,	f)
(1,	4)
(2,	5)
(5,	6)
(4,	7)
(6,	8)
(3,	11)

How would you solve each of these problems with a Greedy method?

Criterion: ______________
Method:

Criterion: ______________
Method:

Class work – Greedy with Finish First to Max Number of Jobs

- Criterion: job that Finishes First (FF)
- Algorithm Fill in your answers

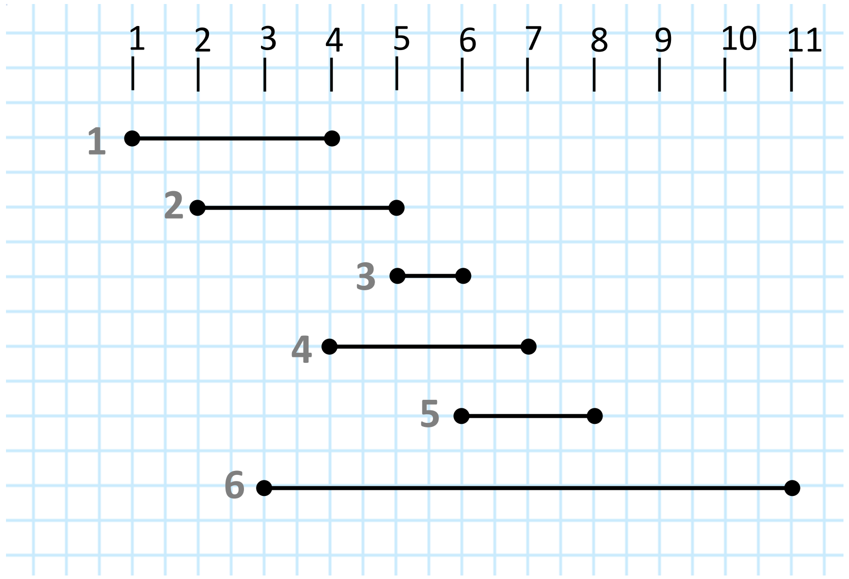
```
// assume jobs are sorted by finish time (increasing)
// and stored at indexes 1 to N
// TC = ____      SC = ______
// You can use Queue with methods: newQueue(), Q.add(v), Q.remove()
Queue GreedyFF(int N, int* start, int *finish)
    sol = newQueue()




    return sol
```

- optimal or not (if not, can we build a counter example?)
  - For **max number of jobs** problem – ___________
  - For **max total value** problem – ____________
- Preprocessing, if needed: Sort jobs in increasing order of their finish time.

max number of jobs

Jobs are already sorted by finish time

	(start, finish)
1	(1, 4)
2	(2, 5)
3	(5, 6)
4	(4, 7)
5	(6, 8)
6	(3, 11)



Class work –Greedy with Finish First to Max Number of Jobs - Solution

- Criterion: job that Finishes First (FF)

- Algorithm

```
// assume jobs are sorted by finish time (increasing)
// and stored at indexes 1 to N
// TC = __Θ(N)__      SC = __Θ(1)__
// You can use Queue with methods: newQueue(), Q.add(v), Q.remove()
Queue GreedyFF(int N, int* start, int *finish)
    sol = newQueue()
    sol.add(1)
    last = 1
    for(k = 2 to N)
        if (finish[last]<=start[k])
            sol.add(k)
            last = k
    return sol
```

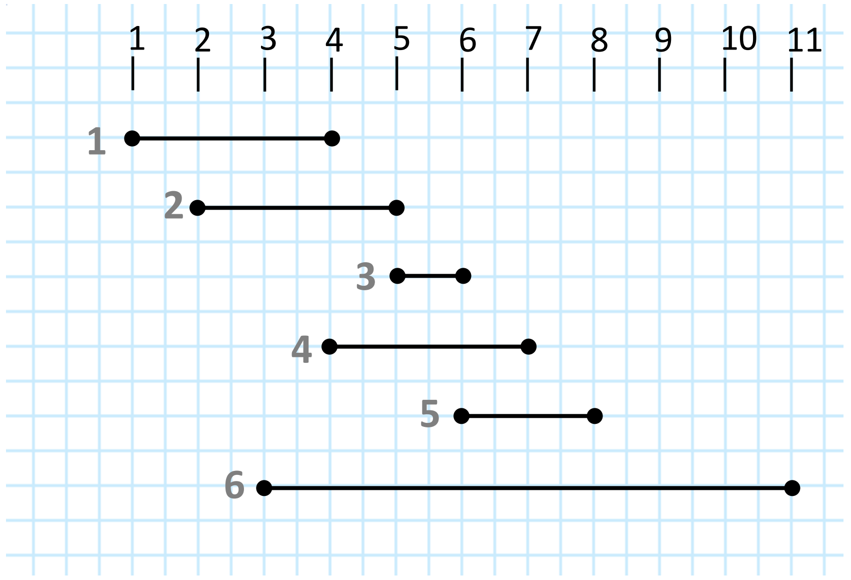
- optimal or not (if not, can we build a counter example?)
  - For **max number of jobs** problem – ____ **optimal**_________
  - For **max total value** problem – ______**not optimal**_______

- Preprocessing, if needed: Sort jobs in increasing order of their finish time. Depends on sorting alg.

max number of jobs

Jobs are already sorted by finish time

	(start,	finish)
1	(1,	4)
2	(2,	5)
3	(5,	6)
4	(4,	7)
5	(6,	8)
6	(3,	11)



Criteria, Optimal solution?

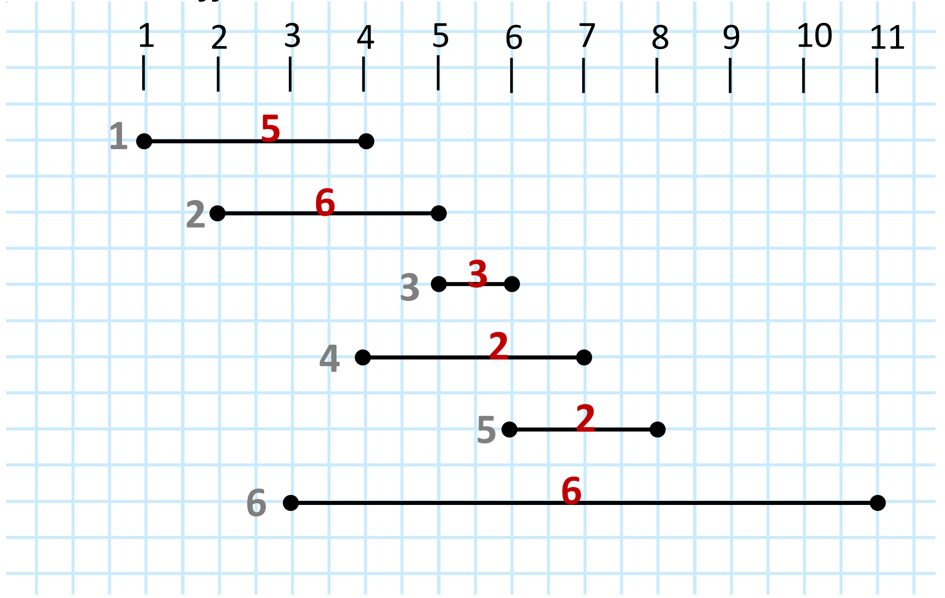
The table below considers the different criteria that can be used for Greedy and our two problem types.

For each pair, say if you think Greedy would find an optimal solution or not.

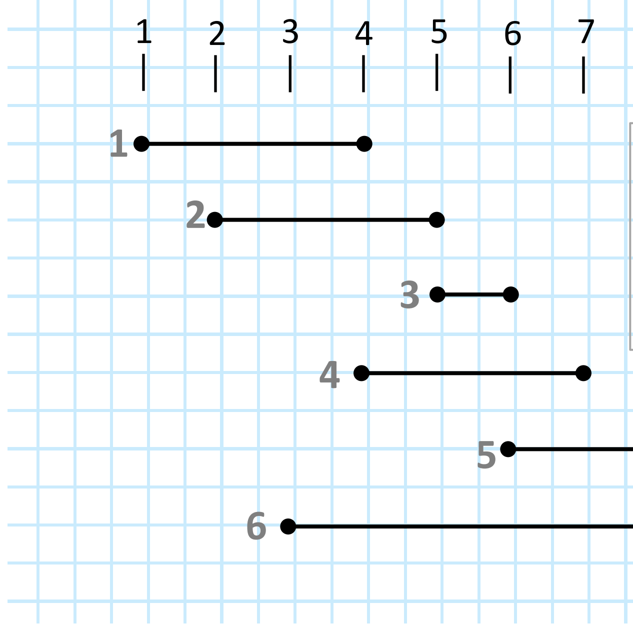
If the answer is not, can you give a counter example?

Criteria	**Max total value**	**Max number of jobs**
Max value/length		
Max value		
Min length		
Finishes last		
Starts first		
Finishes first		
Starts last		

Jobs have *different values*. Maximize **total value**.



Jobs have the *same value*. Maximize **number of jobs**.



What jobs does Greedy with finishes 1st criterion, pick?

Criteria, Optimal solution? Answer

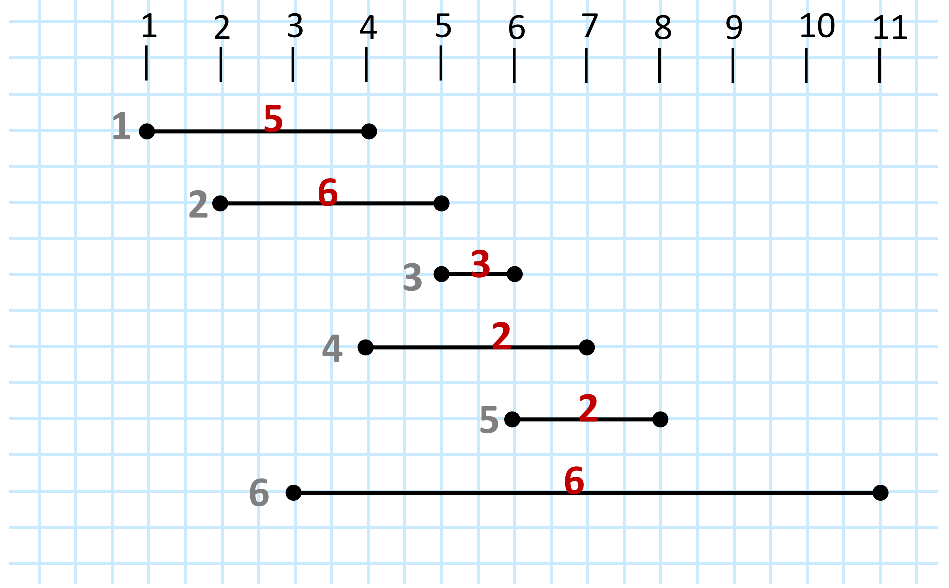
The table below considers the different criteria that can be used for Greedy and our two problem types.

For each pair, say if you think Greedy would find an optimal solution or not.

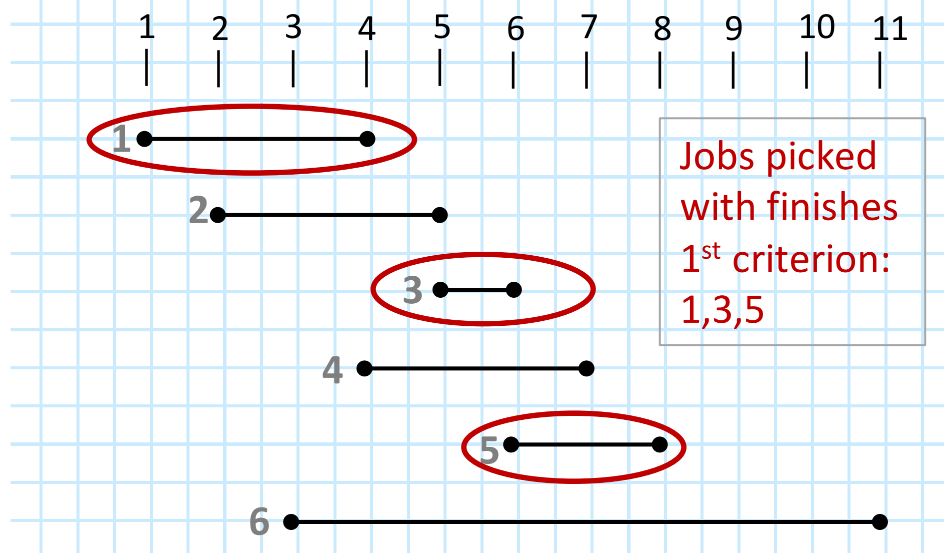
If the answer is not, can you give a counter example?

Criteria	Max total value	Max number of jobs
Max value/length	NO	NO
Max value	NO	NO
Min length	NO	NO
Finishes last	NO	NO
Starts first	NO	NO
Finishes first	NO	***YES***
Starts last	NO	***YES***

Jobs have *different values*. Maximize **total value**.



Jobs have the *same value*. Maximize **number of jobs**.



Jobs picked with finishes 1st criterion: 1,3,5

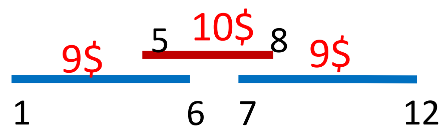
Note: Start last picks another set, that is also optimal.

Summary and Counter Examples

Max **total value**

- job = (start, finish, value)
- Dynamic Programming - optimal
- Greedy – none optimal

Example showing that Greedy with ***largest value*** does not give an optimal solution.

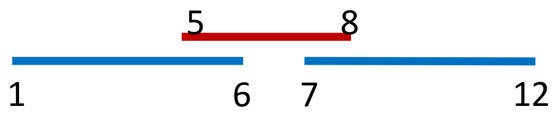


Greedy will pick the red job. Nothing else fits.
Better (optimal): the 2 blue jobs.

Max **number of jobs**

- job = (start, end)
- Dynamic Programming - optimal
- Greedy optimal:
  - finish first (see CLRS for proof)
  - start last
- Greedy NOT optimal:
  - finish last
  - start first
  - other

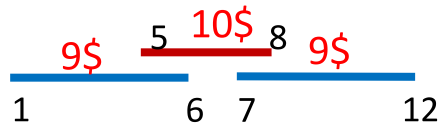
Example showing that ***min length*** Does not give an optimal solution.



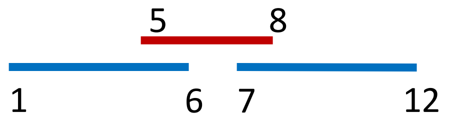
Greedy will pick the red job. Nothing else fits.
Better (optimal): the 2 blue jobs.

Greedy Algorithms

- Greedy does NOT solve optimally:
  - 0/1 Knapsack
  - Job Scheduling with Max Profit



- Greedy solves optimally
  - Knapsack fractional
  - Job Scheduling with Max Number of Jobs
    - All jobs have the same value, want to maximize number of picked jobs
  - Huffman (see if time permits)
  - Minimum Spanning Tree in Graphs (see soon)



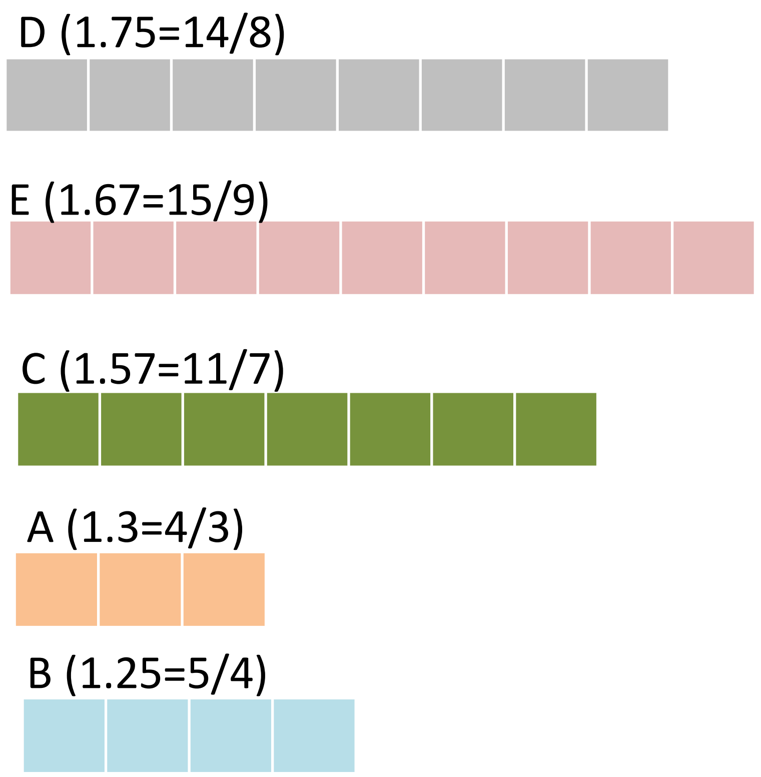
- Difference between Greedy and Dynamic Programming:
  - In DP you will compute two or more answers for the current problem and pick the best of those.
    - You identify and use the answer for at least two smaller problems.
  - In greedy, you make the greedy choice and commit to it
    - You do not undo any choice.
    - (You only have one smaller problem left.)

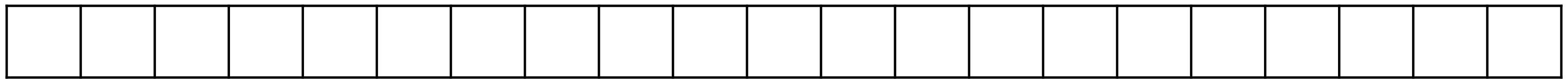
Worksheet (make copies)

Item	A	B	C	D	E
Value	4	5	11	14	15
Weight	3	4	7	8	9
Ratio	4/3= 1.3	5/4= 1.25	11/7= 1.57	14/8= 1.75	15/9= 1.67

Reordered decreasing by **ratio**: **D, E, C, A, B**

1.75, 1.67, 1.57, 1.3, 1.25





W=21 (knapsack capacity is 21)