CS 201
Greedy method

Format of Greedy algorithms:

```
for each item X in some order
    decide whether/how to include item X in the solution;
```

Examples of Greedy algorithms:

Sorting: insertion sort, selection sort
Minimum spanning tree: Kruskal, Prim
Single-source shortest paths: Dijkstra

For many problems it's easy to design a greedy algorithm.
We'll see some more examples of greedy algorithms that solve various problems.

Unfortunately some "obvious" greedy algorithms are incorrect.
We'll also see examples of these and how to identify them.

Problem: Activity Selection

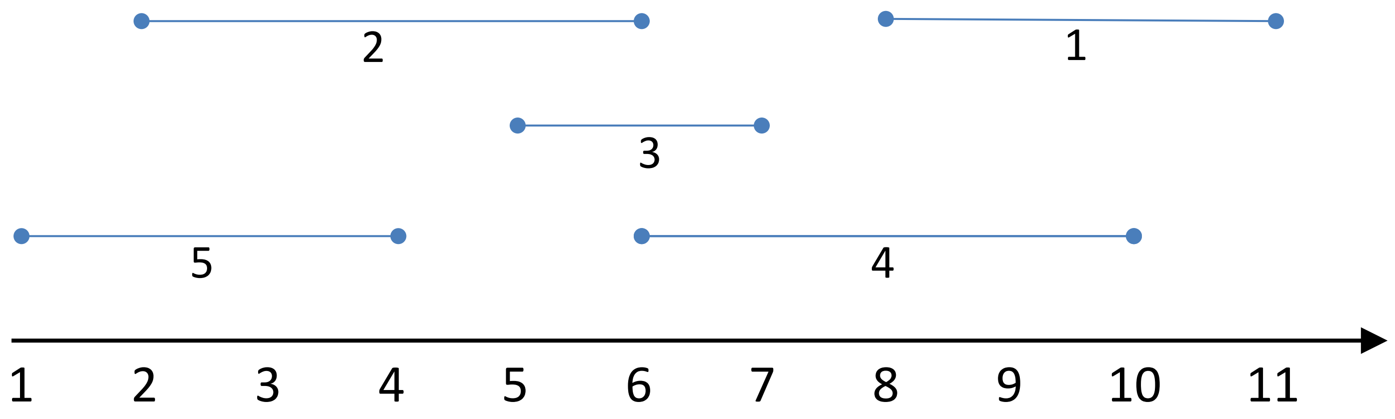
Activities {1…n}
Start times S[1…n]
Finish times F[1…n]

- Activities j and k are <u>compatible</u> if either F[j] ≤ S[k] or F[k] ≤ S[j].
- Otherwise activities j and k <u>overlap</u> or <u>conflict</u>.

Goal: Choose largest subset of mutually compatible activities.

Example:

	1	2	3	4	5
S	8	2	5	6	1
F	11	6	7	10	4



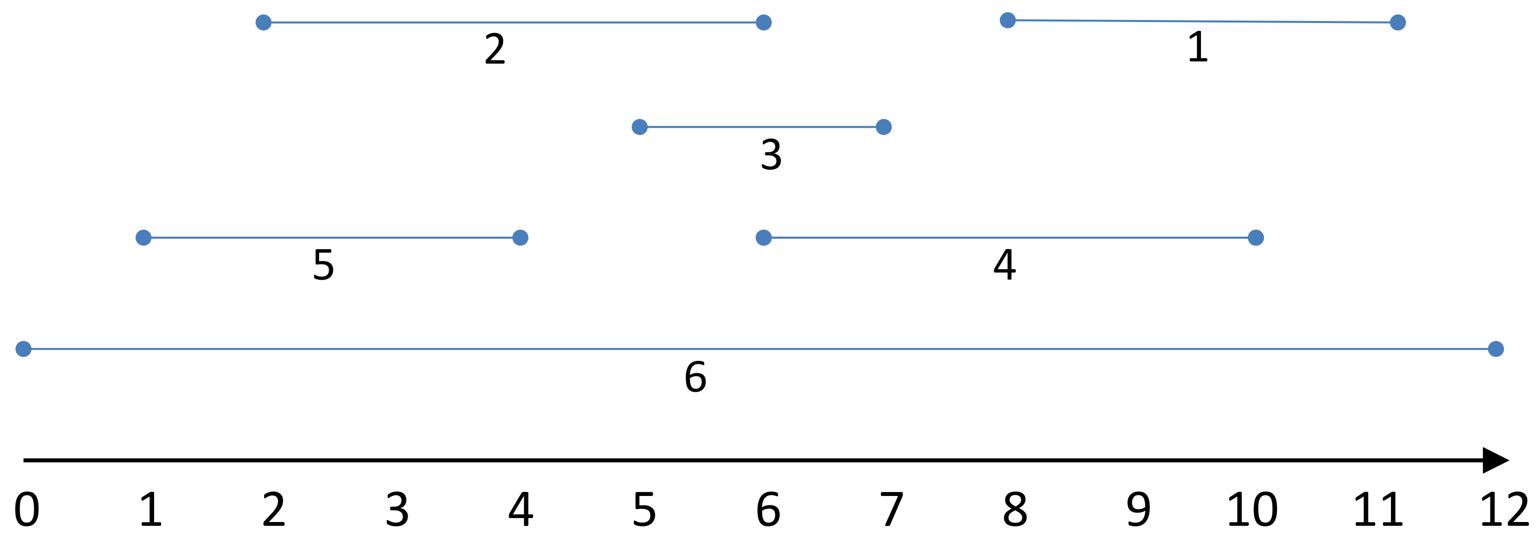
Greedy Algorithm 1 for Activity Selection:

```
A = { };
for each activity k in order of ascending S[k]
    if (activity k is compatible with all activities in A)
        A = A ∪ {k};
```

Correct for previous example, but not for all possible inputs.

Counterexample: add new 6th activity as shown

	1	2	3	4	5	6
S	8	2	5	6	1	0
F	11	6	7	10	4	12



So Greedy Algorithm 1 above is NOT correct.

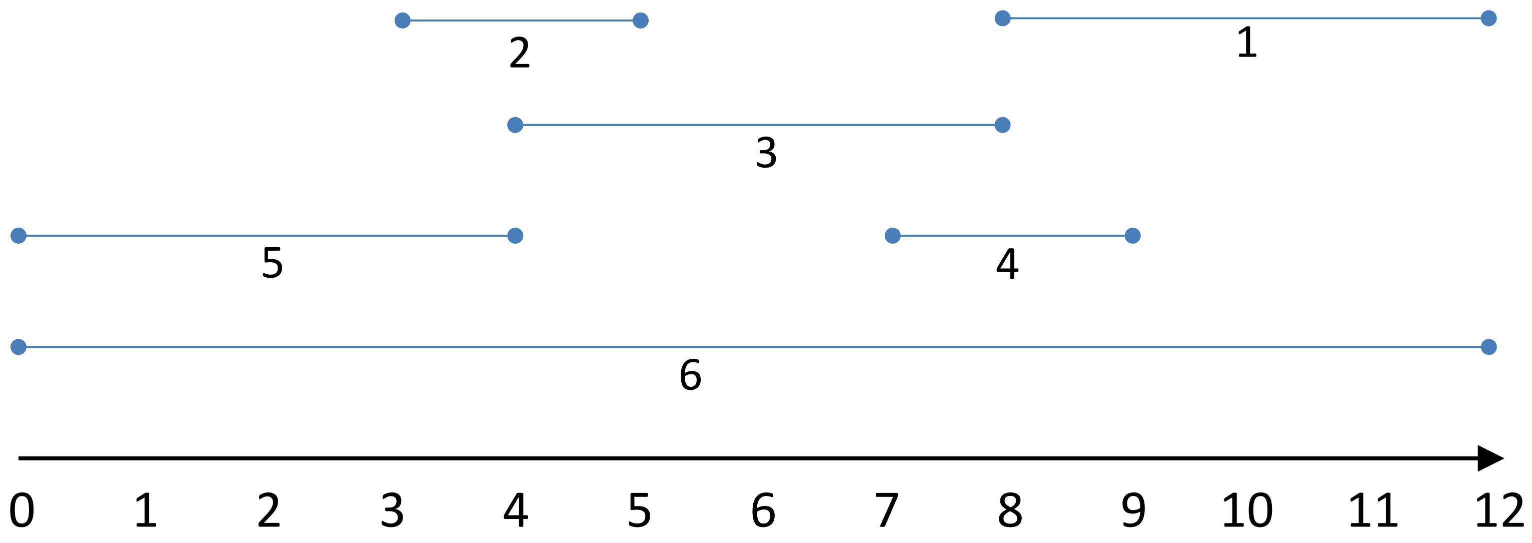
Greedy Algorithm 2 for Activity Selection:

```
    A = { };
    for each activity k in order of ascending lengths F[k]–S[k]
        if (activity k is compatible with all activities in A)
            A = A ∪ {k};
```

Correct for previous examples, but not for all possible inputs.

Counterexample: stretch 1, 3, 5 and shrink 2, 4 as shown

	1	2	3	4	5	6
S	8	3	4	7	0	0
F	12	5	8	9	4	12



So Greedy Algorithm 2 above is NOT correct.

Greedy Algorithm 3 for Activity Selection:

```
A = { };
for each activity k in order of **ascending F[k]**
    if (activity k is compatible with all activities in A)
        A = A ∪ {k};
```

Correct for previous examples.

Greedy algorithm 3 is ALWAYS correct for all possible inputs.
Why?
("No known counterexample" isn't a sufficient justification.)

Contradiction argument:
Suppose Greedy Algorithm 3 was not correct.
Consider its first incorrect choice of some activity k.
So there must be a better next choice, say activity k',
which leads to a larger solution subset A'.
By compatibility, every future activity k'' in A' has $F[k'] \le S[k'']$.
Also $F[k] \le F[k']$ because k precedes k' in ascending order of F.
Hence $F[k] \le S[k'']$.
Therefore solution $A' - \{k'\} \cup \{k\}$ is an equally good solution,
thus contradicting that k' is better choice than k.

So Greedy Algorithm 3 above is indeed correct.

Problem: Fractional Knapsack

Objects {1…n}
Profits P[1…n]
Weights W[1…n]
Maximum weight capacity M

For any fraction $0 \le f \le 1$, if we take f*W[k] weight of object k, then we will obtain f*P[k] profit.

Goal: Determine the fractions F[1…n] for each object so that $\Sigma_{1\le k\le n}$ F[k]*W[k] ≤ M, and $\Sigma_{1\le k\le n}$ F[k]*P[k] is as large as possible.

Example:

	1	2	3	4	5	6	7	8	9
P	14	30	36	9	33	40	12	42	35
W	4	5	8	2	6	10	3	12	7
F									

M = 16

Greedy Algorithm 1 for Fractional Knapsack:

```
Total = 0
for each object k in order of descending P[k] {
    F[k] = min (M/W[k], 1);
    M = M – F[k]*W[k];
    Total = Total + F[k]*P[k];
}
```

	1	2	3	4	5	6	7	8	9
P	14	30	36	9	33	40	12	42	35
W	4	5	8	2	6	10	3	12	7
F						2/5		1	

M	Total	k	F[k]
16	0	8	1
4	42	6	2/5
0	58		

All other F[k] = 0

Total profit = 58   (not optimal)

Greedy Algorithm 1 above is NOT correct.

Greedy Algorithm 2 for Fractional Knapsack:

```
Total = 0
for each object k in order of ascending W[k] {
    F[k] = min (M/W[k], 1);
    M = M – F[k]*W[k];
    Total = Total + F[k]*P[k];
}
```

	1	2	3	4	5	6	7	8	9
P	14	30	36	9	33	40	12	42	35
W	4	5	8	2	6	10	3	12	7
F	1	1		1	1/3		1		

M	Total	k	F[k]
16	0	4	1
14	9	7	1
11	21	1	1
7	35	2	1
2	65	5	1/3
0	76		

All other F[k] = 0

Total profit = 76   (not optimal)

Greedy Algorithm 2 above is NOT correct.

Greedy Algorithm 3 for Fractional Knapsack:

```
Total = 0
for each object k in order of descending ratios P[k]/W[k] {
    F[k] = min (M/W[k], 1);
    M = M – F[k]*W[k];
    Total = Total + F[k]*P[k];
}
```

	1	2	3	4	5	6	7	8	9
P	14	30	36	9	33	40	12	42	35
W	4	5	8	2	6	10	3	12	7
R	3.5	6	4.5	4.5	5.5	4	4	3.5	5
F		1			1				5/7

M	Total	k	F[k]
16	0	2	1
11	30	5	1
5	63	9	5/7
0	88		

All other F[k] = 0

Total profit = 88 (optimal)

Another example of tracing Greedy Algorithm 3 for the Fractional Knapsack problem:

Suppose n=4, M=8

	1	2	3	4
P	12	15	16	18
W	2	3	4	6
Ratio	6	5	4	3
Fraction	1	1	0.75	0

M	Total	k	F[k]
8	0	1	1
6	12	2	1
3	27	3	3/4
0	39		

Greedy algorithm 3 is ALWAYS correct for all possible inputs. Why?

Contradiction argument:

Suppose Greedy Algorithm 3 was not optimal.
That is, suppose it takes too much of some object k, and too little of some other object k’, where k precedes k’ in order of descending ratio.
So the optimal solution must have $F[k]<1$ and $F[k']>0$, where $P[k]/W[k] \geq P[k']/W[k']$.

Now we can increase the amount of object k by Z units, and also decrease the amount of object k’ by Z units, until either F[k] reaches 1 or F[k’] reaches 0.
Here $Z = \min((1–F[k])*W[k], F[k']*W[k'])$.
This change would increase the total profit by
$Z*(P[k]/W[k] – P[k']/W[k']) \geq 0$.
So either Greedy Algorithm 3 does not take too much of object k, or it does not take too little of other object k’, or both.

Contradiction, so Greedy Algorithm 3 is correct.

Problem: Huffman Coding

Alphabet A[1…n] with n characters
Relative frequencies F[1…n] of each character

Example:

A	a	b	c	d	e	f	g	h	i	j
F	9	2	5	6	12	3	4	7	8	1

- Assign binary codes C[1…n] to each character (either fixed-length or variable-length codes)
- To encode a message, replace each character A[k] by its corresponding binary code C[k]
- To decode a message, replace each binary code C[k] by its corresponding character A[k]

- We'll use variable-length binary codes to obtain shortest encoded message length
- Larger frequency $\Rightarrow$ shorter binary code, and smaller frequency $\Rightarrow$ longer binary code
- To decode efficiently, the codes must satisfy the prefix property: No character's code C[k] is a prefix of any other character's code C[k']

Goal: Map the characters to variable-length binary codes C[1…n] that satisfy the prefix property and that yield the shortest encoded message lengths

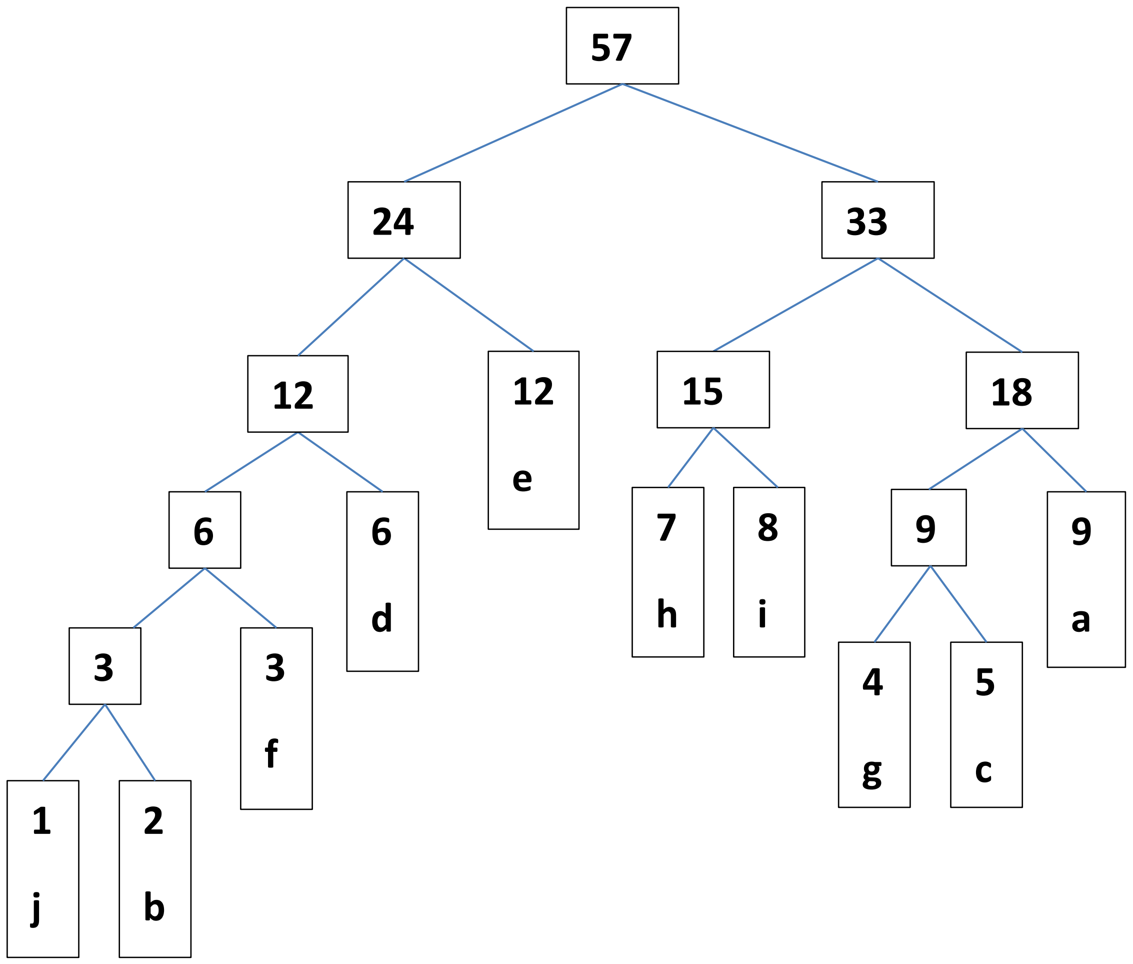
Example:

A	a	b	c	d	e	f	g	h	i	j
F	9	2	5	6	12	3	4	7	8	1
C										

Greedy algorithm for Huffman coding:

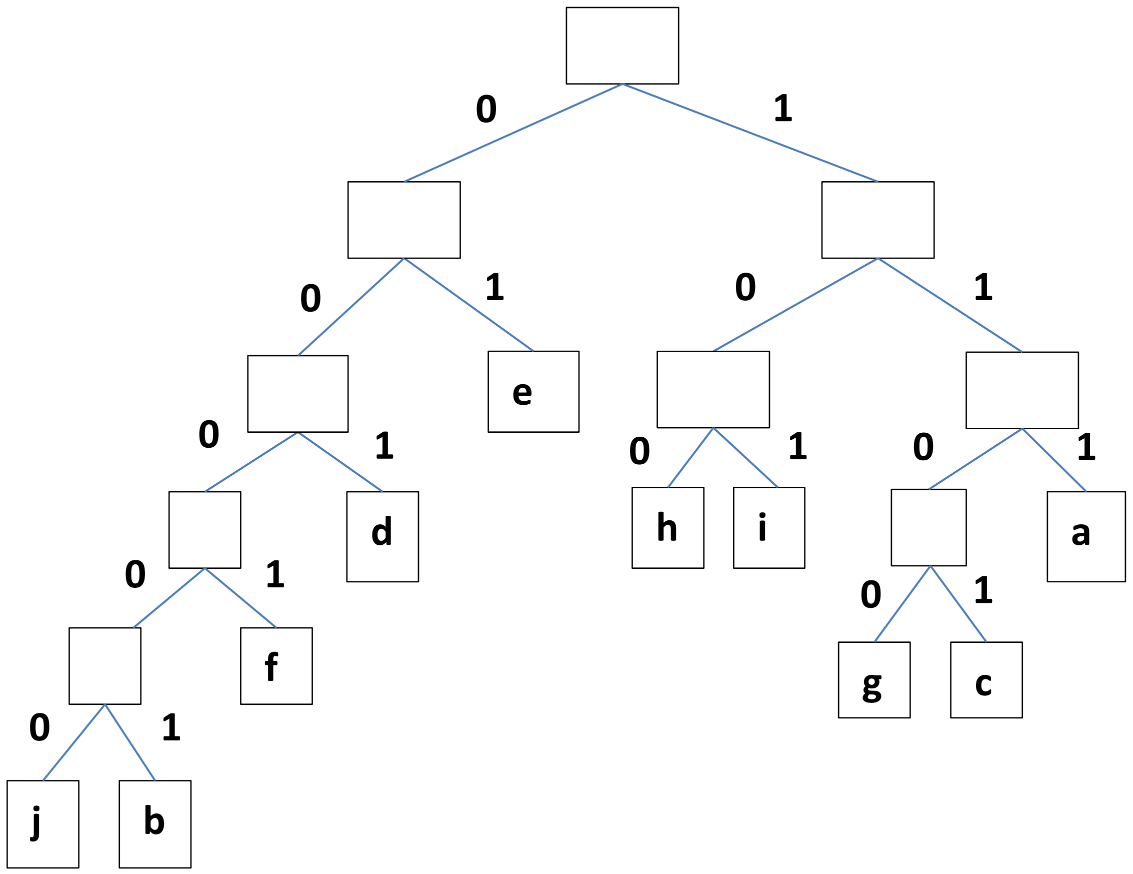
(1) First construct a Huffman tree (binary tree) as follows:

```
H = new Heap( );                // min-ordered heap
for j = 1 to n
    H.insert (new Leaf(F[j], A[j]));         // F[j] is key
while (H.size( ) >= 2) {
    left = H.removeMin( );
    right = H.removeMin( );
    sum = left.key + right.key;
    H.insert (new Node(sum, left, right));  // sum is key
}
root = H.removeMin( );
```



(2) Next assign each character a binary code as follows:

Label each link to left child with 0.
Label each link to right child with 1.
Follow the path from root to each leaf node.
Concatenate the labels to obtain the binary code.



A	a	b	c	d	e	f	g	h	i	j
F	9	2	5	6	12	3	4	7	8	1
C	111	00001	1101	001	01	0001	1100	100	101	00000

Encode a message:

abigchefahead

111 00001 101 1100 1101 100 01 0001 111 100 01 111 001

Decode the message:

1110000110111001101100010001111110001111001

?

Why is prefix property needed to decode the message?

Why does Huffman code always satisfy the prefix property?

Why does Huffman coding yield shorter encoded messages?

- Smaller frequency $\Rightarrow$ added to tree earlier $\Rightarrow$ deeper in tree $\Rightarrow$ longer binary code.
- Larger frequency $\Rightarrow$ added to tree later $\Rightarrow$ shallower in tree $\Rightarrow$ shorter binary code.

Problem: Task Scheduling

Tasks {1…n} that each take one unit of time to perform
Deadlines D[1…n] such that each D[k] ≤ n
Penalties P[1…n]

Example:

	1	2	3	4	5	6
D	1	3	2	3	2	5
P	5	8	3	10	7	4

Current time is 0.
Schedule each task k in a distinct unit interval (S[k], F[k])
such that 0 ≤ S[k] ≤ n–1 and F[k] = S[k]+1.
Task k is <u>late</u> if it is not finished by its deadline D[k], that is,
if F[k] > D[k], or equivalently S[k] ≥ D[k].
Task k is <u>early</u> if it does finish by its deadline D[k], that is,
if F[k] ≤ D[k], or equivalently S[k] < D[k].
If task k is late, it incurs penalty P[k].

Goal: Minimize the total penalty incurred by all late tasks.
(Same as maximizing the penalty avoided by all early tasks.)

Greedy Algorithm 1 for Task Scheduling:

schedule tasks in order of **ascending D[k]**,
and break ties in order of **descending P[k]**

	1	5	3	4	2	6
D	1	2	2	3	3	5
P	5	7	3	10	8	4

S	0	1	2	3	4	5
F	1	2	3	4	5	6

Late tasks = {3,4,2,6}
Total penalty of late tasks = 3 + 10 + 8 + 4 = 25
Not optimum solution
Incorrect algorithm

Greedy Algorithm 2 for Task Scheduling:

schedule tasks in order of **descending P[k]**,
and break ties in order of **ascending D[k]**

	4	2	5	1	6	3
D	3	3	2	1	5	2
P	10	8	7	5	4	3

S	0	1	2	3	4	5
F	1	2	3	4	5	6

Late tasks = {5,1,3}
Total penalty of late tasks = 7 + 5 + 3 = 15
Not optimum solution
Incorrect algorithm

Greedy Algorithm 3 for Task Scheduling:

```
for each task k in order of descending P[k],
    place task k in the latest unassigned interval so that
    it finishes by its deadline if that is possible,
    or otherwise place it in the latest unassigned interval
```

	4	2	5	1	6	3
D	3	3	2	1	5	2
P	10	8	7	5	4	3

S	2	1	0	5	4	3
F	3	2	1	6	5	4

Late tasks = {1,3}
Total penalty of late tasks = 5 + 3 = 8
Correct algorithm
Always yields an optimum solution
Can you justify why?