

# Greedy Algorithm

Kuan-Yu Chen (陳冠宇)

2019/06/12 @ TR-310-1, NTUST

# Review

---

- For many optimization problems, using dynamic programming to determine the best choices is overkill
  - More efficient algorithms will do
- A *greedy algorithm* always makes the choice that looks best at the moment
  - It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution
  - **Greedy algorithms do not always yield optimal solutions,** but for many problems they do

# Recursive Greedy Algorithm

- The recursive greedy algorithm
  - It takes the start and finish times of the activities, i.e.,  $s$  and  $f$
  - The index  $k$  that defines the subproblem  $S_k$  it is to solve
    - $S_k = \{a_i \in S: s_i \geq f_k\}$
  - The size  $n$  of the original problem

RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )

```
1   $m = k + 1$ 
2  while  $m \leq n$  and  $s[m] < f[k]$            // find the first activity in  $S_k$  to finish
3       $m = m + 1$ 
4  if  $m \leq n$ 
5      return  $\{a_m\} \cup \text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6  else return  $\emptyset$ 
```

- In order to start, we add the fictitious activity  $a_0$  with  $f_0 = 0$ , so that subproblem  $S_0$  is the entire set of activities  $S$ 
  - The initial call, which solves the entire problem, is RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )

$k$	$s_k$	$f_k$
0	-	0

1	1	4
---	---	---

2	3	5
---	---	---

3	0	6
---	---	---

4	5	7
---	---	---

5	3	9
---	---	---

6	5	9
---	---	---

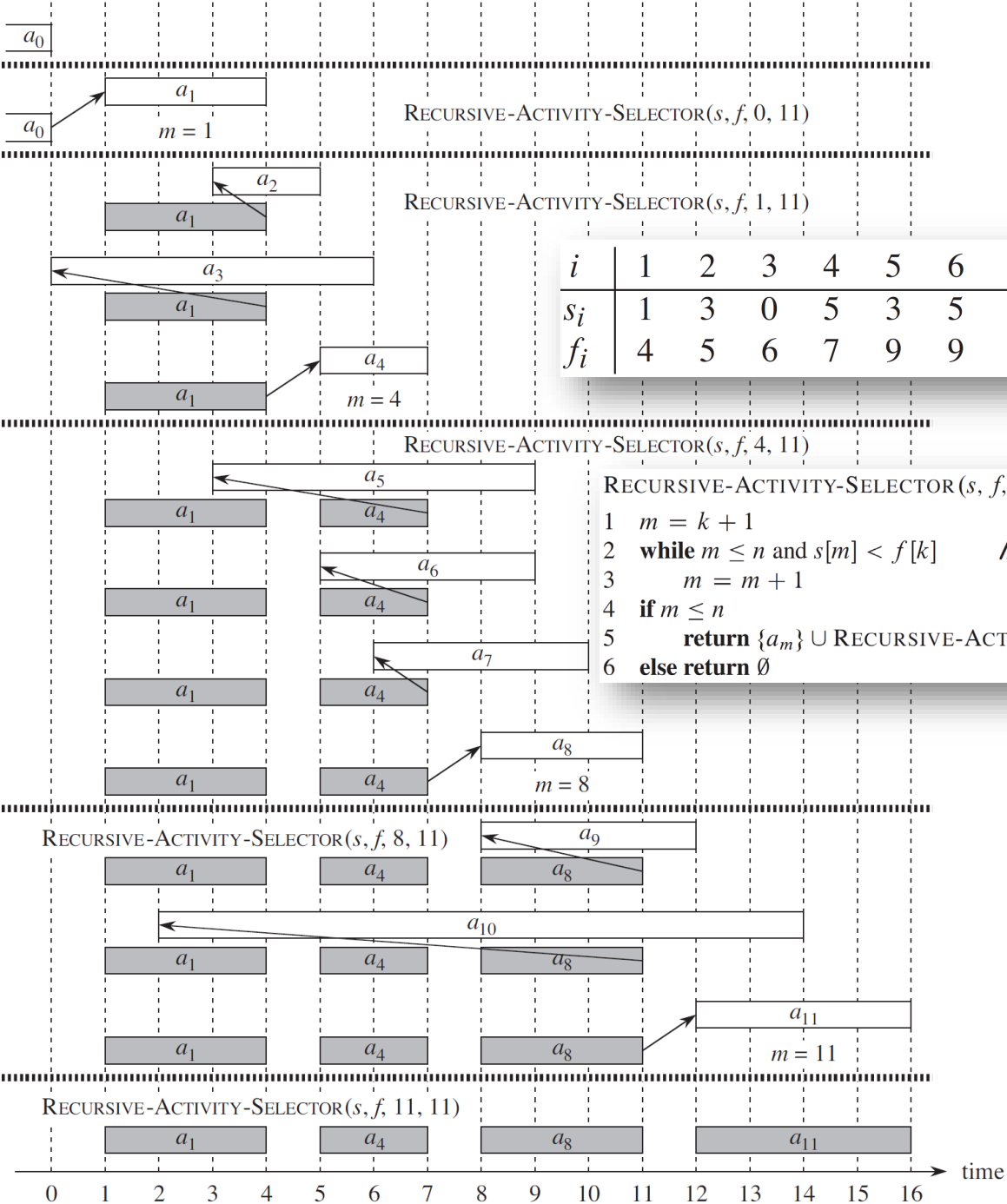
7	6	10
---	---	----

8	8	11
---	---	----

9	8	12
---	---	----

10	2	14
----	---	----

11	12	16
----	----	----



$i$	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	9	9	10	11	12	14	16

RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )

```

1   $m = k + 1$ 
2  while  $m \leq n$  and  $s[m] < f[k]$       // find the first activity in  $S_k$  to finish
3       $m = m + 1$ 
4  if  $m \leq n$ 
5      return  $\{a_m\} \cup \text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6  else return  $\emptyset$ 
```

# Iterative Greedy Algorithm

- We can convert our recursive procedure to an iterative one
  - The procedure RECURSIVE-ACTIVITY-SELECTOR is almost “tail recursive”
  - In fact, some compilers for certain programming languages perform this task automatically
- The procedure GREEDY-ACTIVITY-SELECTOR assumes that the input activities are ordered by monotonically increasing finish time

$i$	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	9	9	10	11	12	14	16

$i$	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	9	9	10	11	12	14	16

$i$	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	9	9	10	11	12	14	16

GREEDY-ACTIVITY-SELECTOR( $s, f$ )


```

1   $n = s.length$ 
2   $A = \{a_1\}$ 
3   $k = 1$ 
4  for  $m = 2$  to  $n$ 
5      if  $s[m] \geq f[k]$ 
6           $A = A \cup \{a_m\}$ 
7           $k = m$ 
8  return  $A$ 
```

# Appendix


## Direct Recursion

```
1 function A()  
2 ▼ {  
3     ...  
4  
5     A() ;  
6  
7     ...  
8 }  
9
```



## Indirect Recursion


```
1 function A()  
2 {  
3     ...  
4  
5     B() ;  
6  
7     ...  
8 }  
9  
10 function B()  
11 {  
12     ...  
13  
14     A() ;  
15  
16     ...  
17 }
```



calling cycle

## Tail Recursion

```
1 function A()  
2 ▼ {  
3     ...  
4  
5     A() ;  
6 }
```



# Property

---

- At each decision point, the greedy algorithm makes choice that seems best at the moment
  - This heuristic strategy does not always produce an optimal solution
  - But as we saw in the activity-selection problem, sometimes it does!
- How can we tell whether a greedy algorithm will solve a particular optimization problem?
  - **Greedy-choice property** and **optimal substructure** are the two key ingredients
  - If we can demonstrate that the problem has these properties, then we are well on the way to developing a greedy algorithm for it

# Two Key Ingredients

---

- Greedy-choice property
  - We can assemble a globally optimal solution by making locally optimal (greedy) choices
    - In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems
- Optimal substructure
  - A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems



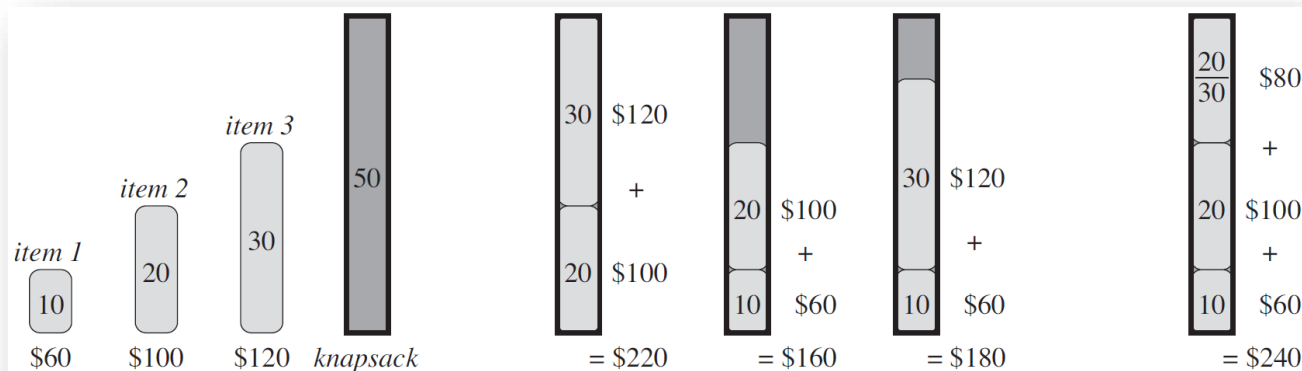
# DP & Greedy Algorithm.

---

- In dynamic programming, we make a choice at each step, but the choice usually depends on the solutions to subproblems
  - Typically in a bottom-up manner
- In a greedy algorithm, we make whatever choice seems best at the moment and then solve the subproblem that remains
  - The choice may depend on choices so far, but it cannot depend on any future choices or on the solutions to subproblems

# DP & Greedy Algorithm..

- Let us investigate two variants of a classical optimization problem
  - The **0-1 knapsack problem**
    - A thief robbing a store finds  $n$  items. The  $i^{th}$  item is worth  $i$  dollars and weighs  $w_i$  pounds, where  $i$  and  $w_i$  are integers. The thief wants to take as valuable a load as possible, but he can carry at most  $W$  pounds in his knapsack, for some integer  $W$ . Which items should he take?
  - The **fractional knapsack problem**
    - The setup is the same, but the thief can take fractions of items, rather than having to make a binary (0-1) choice for each item



# In This Semester.

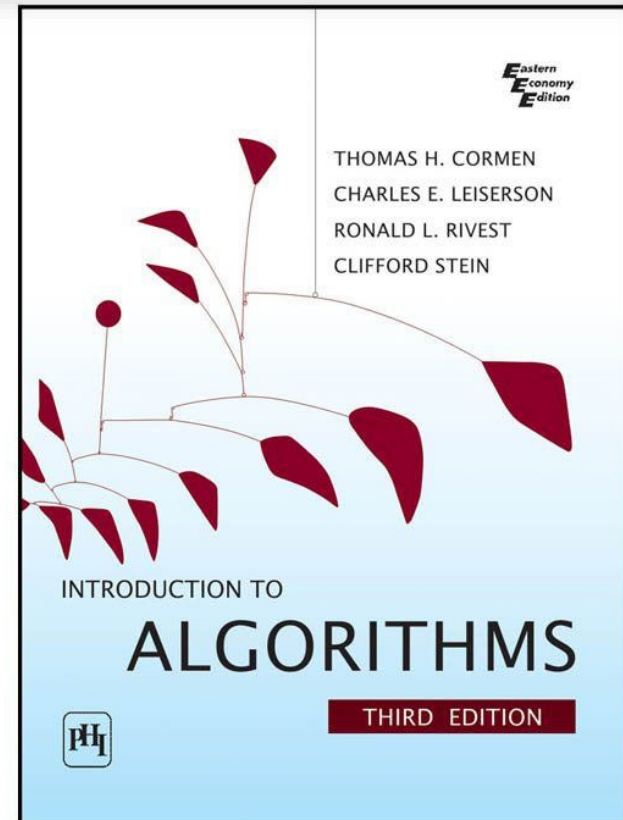
<b>1</b>	<b>The Role of Algorithms in Computing</b>	<b>5</b>
1.1	Algorithms	5
1.2	Algorithms as a technology	11
<b>2</b>	<b>Getting Started</b>	<b>16</b>
2.1	Insertion sort	16
2.2	Analyzing algorithms	23
2.3	Designing algorithms	29
<b>3</b>	<b>Growth of Functions</b>	<b>43</b>
3.1	Asymptotic notation	43
3.2	Standard notations and common functions	53
<b>4</b>	<b>Divide-and-Conquer</b>	<b>65</b>
4.1	The maximum-subarray problem	68
4.2	Strassen's algorithm for matrix multiplication	75
4.3	The substitution method for solving recurrences	83
4.4	The recursion-tree method for solving recurrences	88
4.5	The master method for solving recurrences	93
★ 4.6	Proof of the master theorem	97

<b>6</b>	<b>Heapsort</b>	<b>151</b>
6.1	Heaps	151
6.2	Maintaining the heap property	154
6.3	Building a heap	156
6.4	The heapsort algorithm	159
6.5	Priority queues	162
<b>7</b>	<b>Quicksort</b>	<b>170</b>
7.1	Description of quicksort	170
7.2	Performance of quicksort	174
7.3	A randomized version of quicksort	179
7.4	Analysis of quicksort	180
<b>8</b>	<b>Sorting in Linear Time</b>	<b>191</b>
8.1	Lower bounds for sorting	191
8.2	Counting sort	194
8.3	Radix sort	197
8.4	Bucket sort	200

# In This Semester..

- 15 Dynamic Programming 359**
  - 15.1 Rod cutting 360
  - 15.2 Matrix-chain multiplication 370
  - 15.3 Elements of dynamic programming 378
  - 15.4 Longest common subsequence 390
  - 15.5 Optimal binary search trees 397
- 16 Greedy Algorithms 414**
  - 16.1 An activity-selection problem 415
  - 16.2 Elements of the greedy strategy 423
  - 16.3 Huffman codes 428
- 21 Data Structures for Disjoint Sets 561**
  - 21.1 Disjoint-set operations 561
  - 21.2 Linked-list representation of disjoint sets 564
  - 21.3 Disjoint-set forests 568
- 22 Elementary Graph Algorithms 589**
  - 22.1 Representations of graphs 589
  - 22.2 Breadth-first search 594
  - 22.3 Depth-first search 603
  - 22.4 Topological sort 612
  - 22.5 Strongly connected components 615
- 23 Minimum Spanning Trees 624**
  - 23.1 Growing a minimum spanning tree 625
  - 23.2 The algorithms of Kruskal and Prim 631

- 24 Single-Source Shortest Paths 643**
  - 24.1 The Bellman-Ford algorithm 651
  - 24.2 Single-source shortest paths in directed acyclic graphs 655
  - 24.3 Dijkstra's algorithm 658
  - 24.4 Difference constraints and shortest paths 664
  - 24.5 Proofs of shortest-paths properties 671



<http://nlp.csie.ntust.edu.tw/sws2019/>

LOADING

# 2019 SPEECH SIGNAL PROCESSING WORKSHOP

2019/06/25

國立臺灣科技大學國際大樓  
1樓IB-101會議室

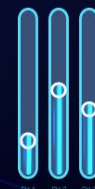


Formosa  
Grand  
Challenge

科技大擂台



REGULATOR





# <http://nlp.csie.ntust.edu.tw/sws2019/>

---



# Questions?

---



[kychen@mail.ntust.edu.tw](mailto:kychen@mail.ntust.edu.tw)