COMP 550
Algorithm and Analysis

Greedy Algorithms

Based on CLRS Sec. 15.1 and 15.2

Some slides are adapted from ones by prior instructors Prof. Plaisted and Prof. Osborne

Coin Change

- You have unlimited quantities of pennies (1 cent), nickels (5 cents), dimes (10 cents), and quarters (25 cents)
- You need to provide change of x cents. How to determine the minimum number of coins to equal x?
- Fill the table for $x = 61$.

Coin	Number	Value
Quarters (25¢)		
Dimes (10¢)		
Nickels (5¢)		
Pennies (1¢)		

Coin Change

- You have unlimited quantities of pennies (1 cent), nickels (5 cents), dimes (10 cents), and quarters (25 cents)
- You need to provide change of x cents. How to determine the minimum number of coins to equal x?
- Fill the table for $x = 61$ cents.

Coin	Number	Value
Quarters (25¢)	2	50¢
Dimes (10¢)	1	10¢
Nickels (5¢)	0	0¢
Pennies (1¢)	1	1¢

Coin Change

- What strategy did you use to solve this?
- Mine is a greedy strategy
  - Pick 25¢ until the remaining value is less than 25¢
  - After that, pick 10¢ until the remaining value is less than 10¢
  - After that, pick 5¢ until the remaining value is less than 5¢
  - After that, pick 1¢ until the remaining value is less than 1¢

Greedy Algorithm for Coin Change

- Input: A list of coins C, and a $target$
- Output: List of coins that equals $target$

```
  Iterative-Coin-Change(C, target)
1.     Let A be an empty list
2.     Sort C in descending order (not
                    needed if presorted)
3.     while (target ≠ 0)
4.        c = C[1] //largest coin in C
5.        if (c ≤ target)
6.           A.append(c)
7.           target = target – c
8.        else
9.           C.remove(c)
10.    return C
```

Make a greedy choice

Set up a subproblem to solve in the next iterations

Greedy Algorithm for Coin Change

- Input: A list of coins C, and a $target$
- Output: List of coins that equals $target$

```
   Coin-Change(C, target)
1.    Let A be an empty list
2.    Sort C in descending order (not needed if presorted)
3.    return Recursive-Coin-Change(C, A, target)
```

```
   Recursive-Coin-Change(C, A, target)
1.    c = C[1] //largest coin in C
2.    if (c ≤ target)
3.       A.append(c)
4.       return Recursive-Coin-Change(C, A, target-c)
5.    else
6.       return Recursive-Coin-Change(C.remove(c), A, target)
```

Greedy Strategies

- The choice that seems best at the moment is the one we pick
  - Locally optimal choice leads to a globally optimal solution
- This strategy will work if
  - The problem has optimal substructure property
  - There is a greedy choice with greedy-choice property

Optimal Substructure

- Optimal substructure property
  - Optimal solution of a problem contains optimal solutions of its subproblems
- Does coin change problem has optimal substructure property?
  - Optimal solution for change of 61¢ was $< 25¢, 25¢, 10¢, 1¢ >$
  - What is the optimal solution for 36¢?
    - $< 25¢, 10¢, 1¢ >$

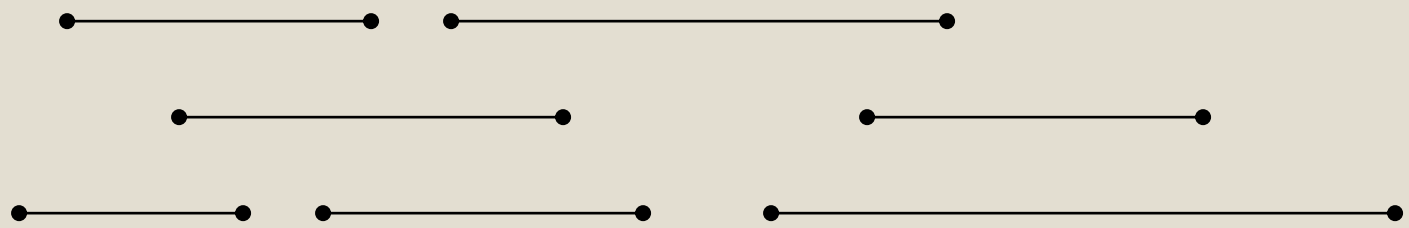
Greedy-Choice Property

- Greedy-Choice Property
  - An item picked by a greedy choice is guaranteed be in an optimal solution
- Does the coin change problem satisfy greedy-choice property?
  - The largest coin less than or equal to target value is part of an optimal solution
  - If target is $61¢$, then there is an optimal solution with at least one 25¢

Activity-Selection Problem

- Also known as *Interval Scheduling* problem
- Input: Set *S* of *n* activities, $a_1, a_2, \ldots, an$.
  - s_i = start time of activity *i*.
  - f_i = finish time of activity *i*.
  - Activity a_i takes place during $[s_i, f_i)$
- Output: Subset *A* of maximum number of compatible activities.
  - Two activities are compatible, if their intervals don't overlap.

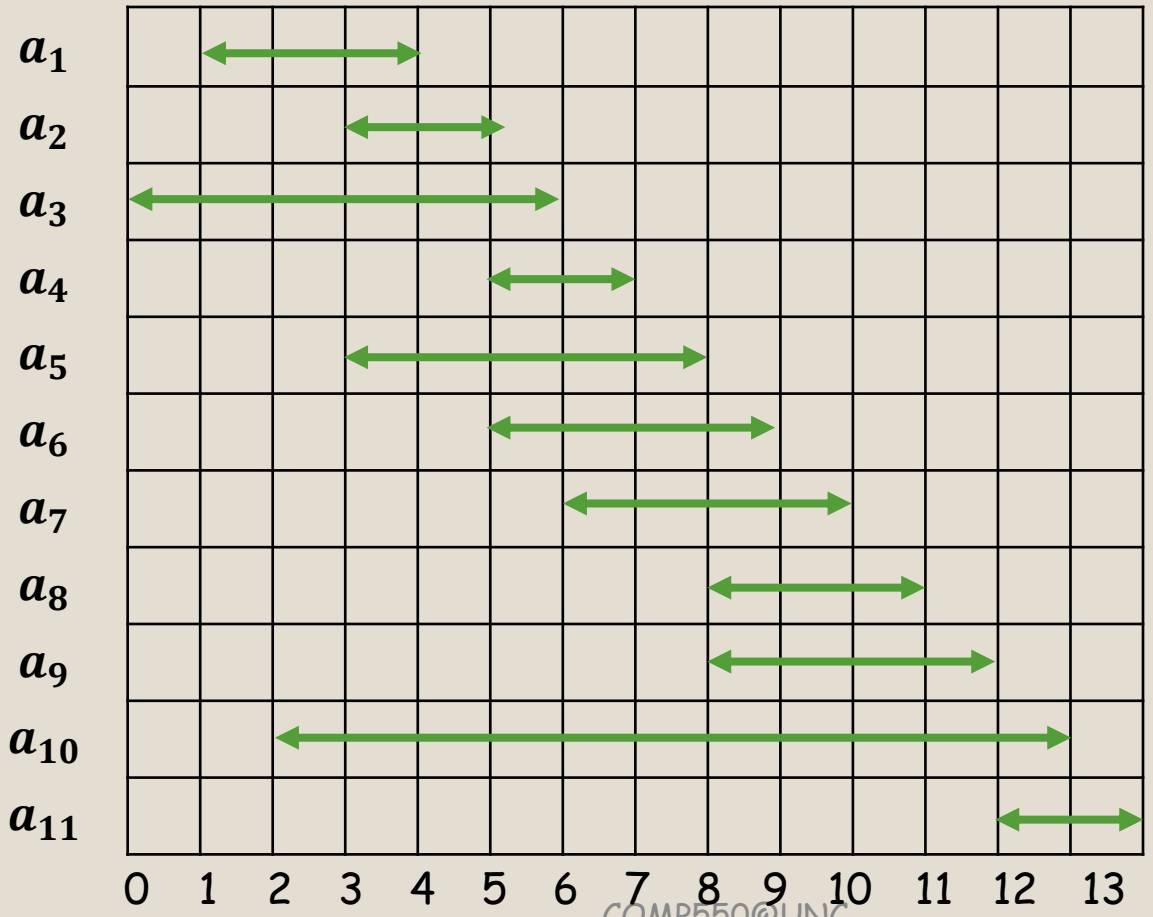
Example: Activities in each line are compatible.



Activity-Selection Problem

- Example:

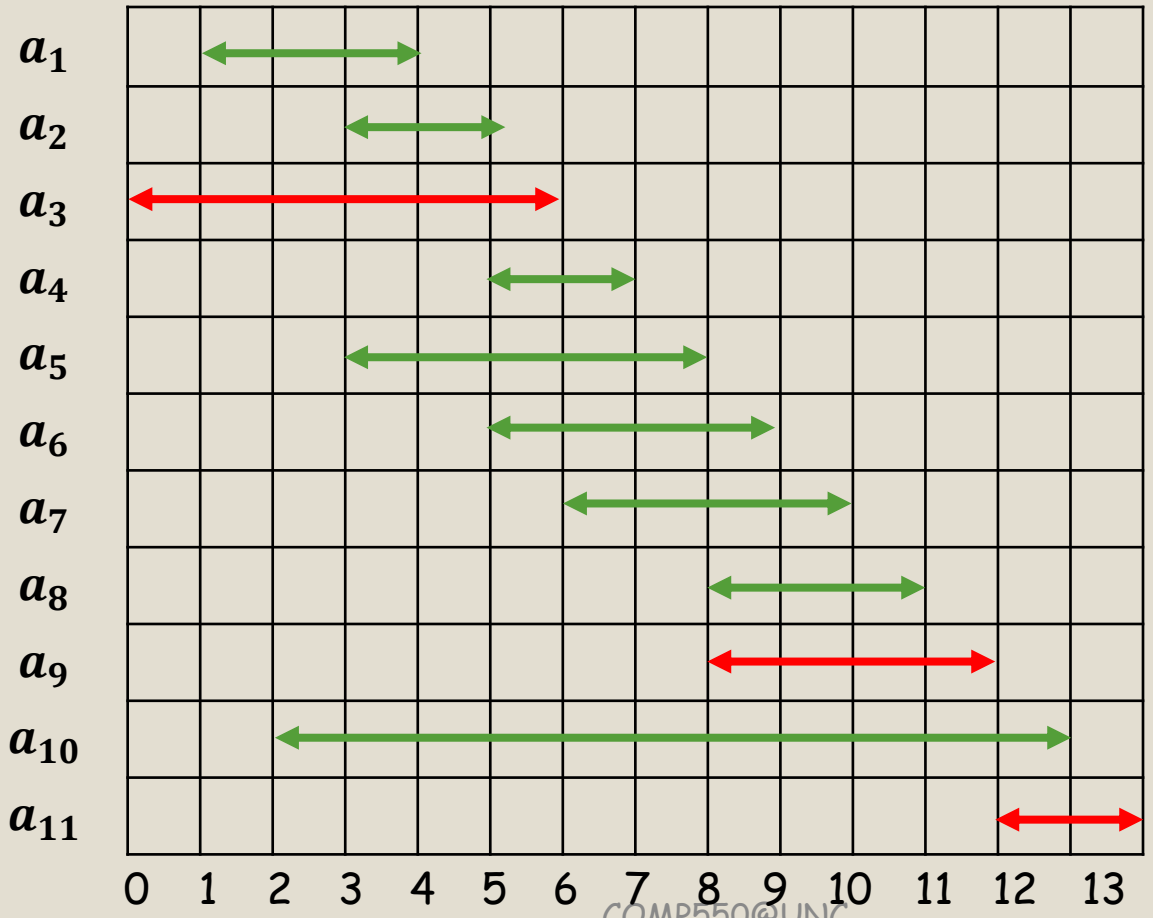
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14



Activity-Selection Problem

- Example:

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



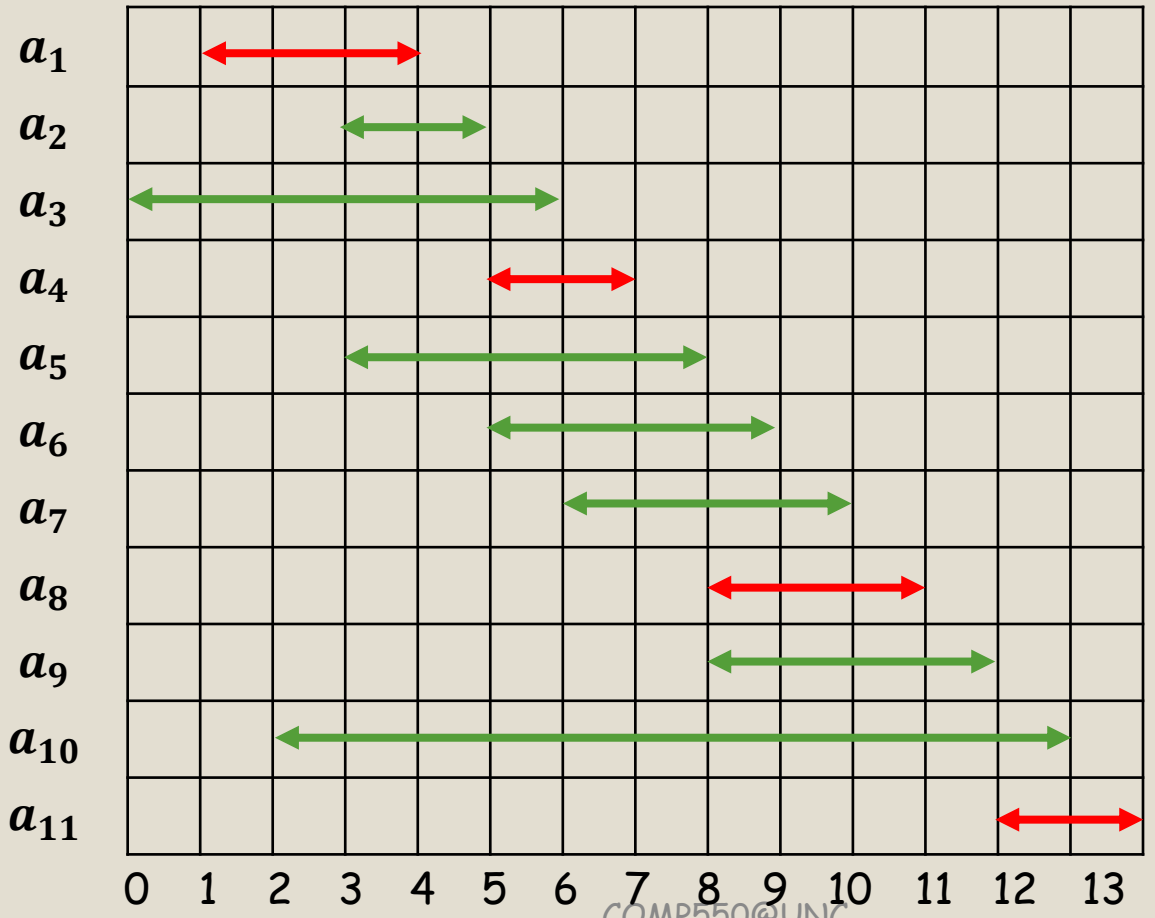
One solution: $\{a_3, a_9, a_{11}\}$

Here, solution means the set satisfies constraints of the problem, each pair in $\{a_3, a_9, a_{11}\}$ are compatible

Activity-Selection Problem

- Example:

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



Another solution:
$\{a_1, a_4, a_8, a_{11}\}$

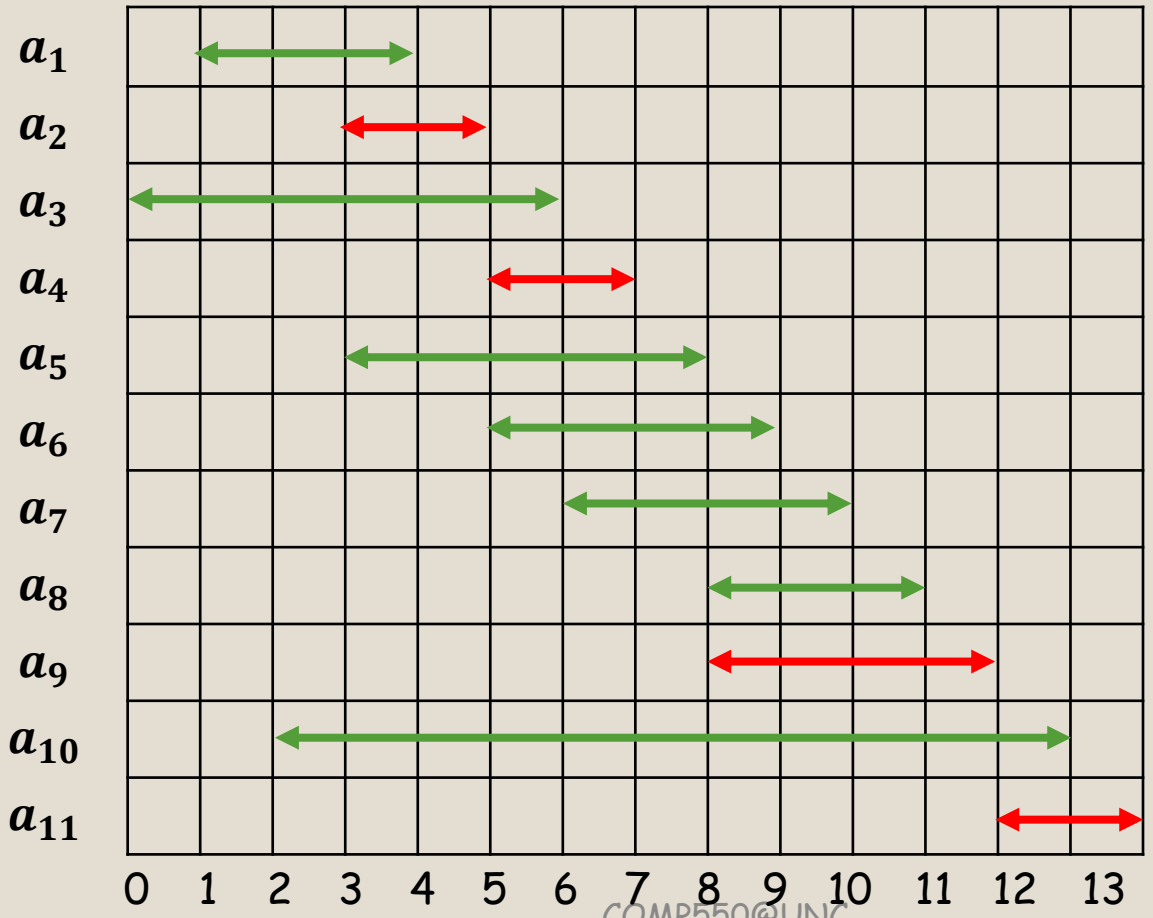
Is there a larger set?

No, this is an optimal solution

Activity-Selection Problem

- Example:

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



Another optimal solution:
$\{a_2, a_4, a_9, a_{11}\}$

Activity-Selection Problem

- What should a greedy algorithm look like for this problem?

```
Iterative-Activity-Selection(S)
1. A = an empty list (to store result)
2. while (S ≠ ∅)
3.     a = An activity from S according
           to a greedy choice
4.     A = A ∪ { a }
5.     S_a = all activities in S that
            overlap with a
6.     S = S \ S_a
7. return A
```

```
Recursive-Activity-Selection(S)
1. If S = ∅
2.     return ∅
3. a = An activity from S according
         to a greedy choice
4. S_a = all activities in S that
         overlap with a (including a)
5. return Recursive-Activity-
           Selection(S \ S_a) ∪ { a }
```

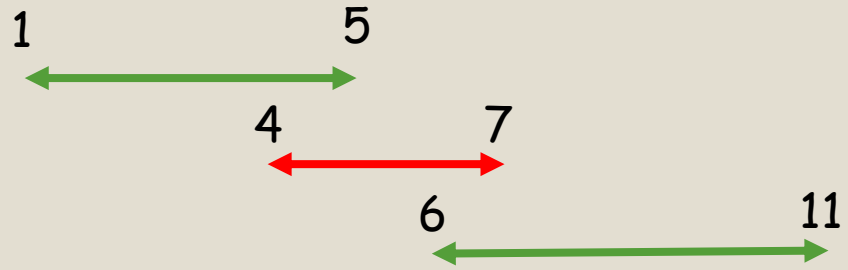
Activity-Selection Problem

- Can we figure out a greedy choice (leading to an optimal solution)?
  - Recall: For greedy algorithm to work, item picked by greedy choice must be part of an optimal solution
- Option 1: Shortest interval (pick the shortest interval that does not overlap with previously-selected intervals)
  - Question: Does it lead to an optimal solution?

Activity-Selection Problem

- Option 1: Shortest interval (pick the shortest interval that does not overlap with previously-selected intervals)
  - Question: Does it lead to an optimal solution?

- Proof of non-optimality



- [4,7) is the shortest interval. If [4,7) is picked, none other can be picked
- [1,5) and [6,11) is optimal subset of activities.

Activity-Selection Problem

- Option 2: Least number of conflicts (pick the activity that does not overlap with the least)
  - Non-optimal (Can you give an example showing this?)
- Option 3: Earliest start time (pick the earliest-starting activity)
  - Non-optimal (Can you give an example showing this?)
- Option 4: Earliest finish time (pick the earliest-finishing activity)
  - Optimal

Why Earliest Finish Time Works?

- Need to argue that "Earliest Finish Time" satisfies greedy-choice property
  - An activity picked by earliest-finish-time greedy choice is always in an optimal solution.
  - Can you write a theorem statement for this?

Theorem 15.1. Consider any non-empty set of activities S_k. Let a_m be an activity in S_k with the earliest finish time. Then, a_m is included in some maximum subset of mutually compatible activities in S_k (In other words, a_m is included in an optimal solution).

Why Earliest Finish Time Works?

Theorem 15.1. Consider any non-empty set of activities S_k. Let a_m be an activity in S_k with the earliest finish time. Then, a_m is included in some maximum subset of mutually compatible activities in S_k.

Proof by exchange argument.

Key idea:

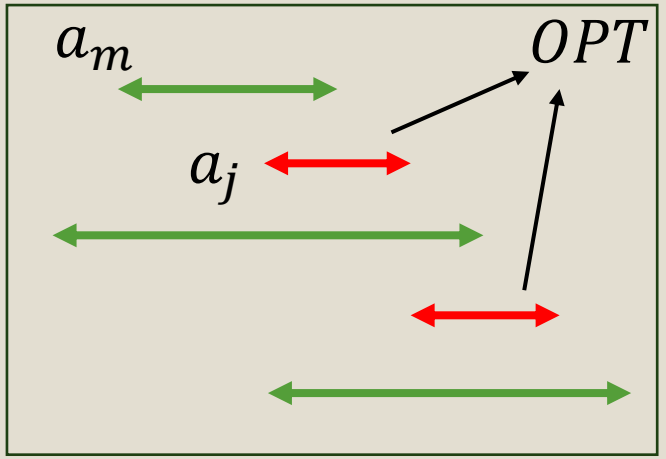
- My solution is no worse than yours
- Exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse

Why Earliest Finish Time Works?

Theorem 15.1. Consider any non-empty set of activities S_k. Let a_m be an activity in S_k with the earliest finish time. Then, a_m is included in some maximum subset of mutually compatible activities in S_k.

Proof by exchange argument.

- Let OPT be a maximum subset of mutually compatible activities.
- Let a_j be the activity in OPT with the earliest finish time, i.e., among all activities in OPT, a_j has the earliest finish time.
- If $a_j = a_m$, we are done. So, assume $a_j \neq a_m$.

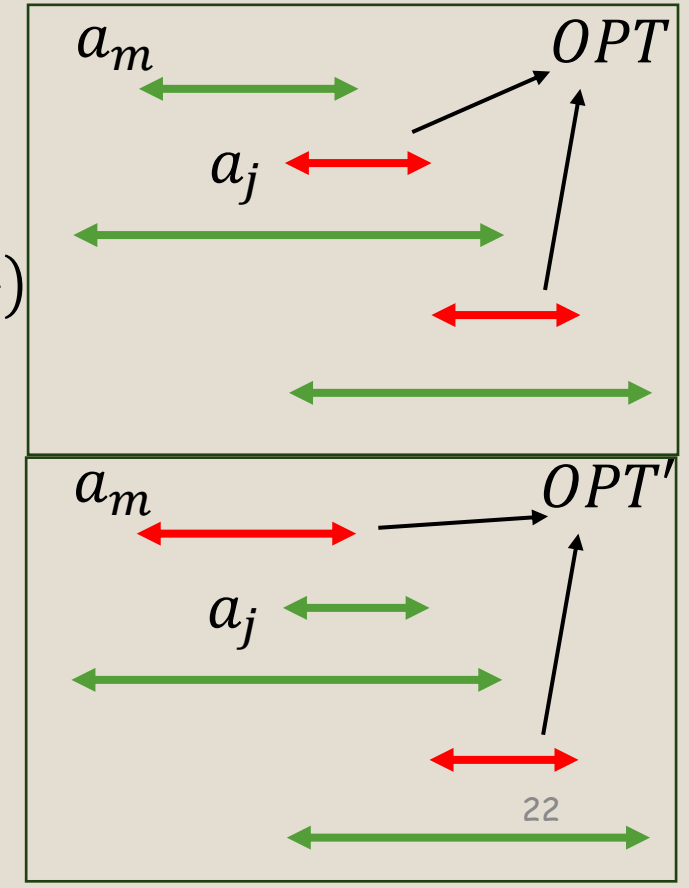


Why Earliest Finish Time Works?

Theorem 15.1. Consider any non-empty set of activities S_k. Let a_m be an activity in S_k with the earliest finish time. Then, a_m is included in some maximum subset of mutually compatible activities in S_k.

Proof by exchange argument.

- Activities in OPT other than a_j must start after a_j ends.
- Since $f_m \leq f_j, a_m$ does not overlap with activities in $(OPT \setminus \{a_j\})$
- So, we can exchange a_j with a_m and create a new subset $OPT' = (OPT \setminus \{a_j\}) \cup \{a_m\}$ of mutually compatible activities.
- $|OPT'| = |\mathrm{OPT}|$.



Optimal Substructure

Let a_m be the activity with the earliest finish time and S_m be the set of activities that overlap with a (including a). Then,

optimal solution for $S = \{a_m\} \cup$ optimal solution for $S \backslash S_m$

Proof by contradiction.

- Let A be a solution for S by taking $\{a_m\} \cup$ optimal solution for $S \backslash S_m$.
- For contradiction, assume A is not optimal.
- Let OPT be an optimal solution for S that includes a_m.
  - By Theorem 15.1 (greedy-choice property), such an optimal solution exists.
- Remove $\{a_m\}$ from OPT. This $OPT \setminus \{a_m\}$ should be larger than the optimal solution for $S \backslash S_m$, contradiction.

Greedy Algorithm for Activity Selection

```
  Iterative-Activity-Selection(S)
1. Sort S be f_i values
2. A = {a_1}
3. k = 1  //k denotes index of the last selected activity
4. for (i = 2 to n)
5.     if (f_k <= s_i)
6.         A = A ∪ { a_i }
7.         k = i
8. return A
```

Correctness: Immediate from greedy-choice and optimal substructure properties

Running time: $\Theta(n \lg n)$

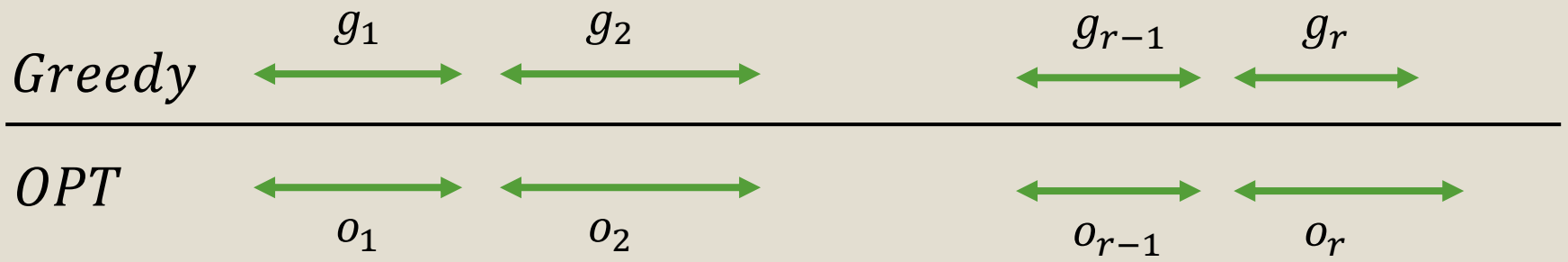
If input array S is pre-sorted by finish time, then $\Theta(n)$

Another Optimality Proof

Theorem. The algorithm returns a largest subset of compatible activities.

Proof Sketch: Let $Greedy$ be our solution.

- Assume $Greedy$ is not optimal
- Assume $Greedy = \{g_1, g_2, \dots, g_s\}$
- Assume $Opt = \{o_1, o_2, \dots, o_t\}$ is an optimal solution with activities such that $o_1 = g_1, o_2 = g_2, \dots, o_{r-1} = g_{r-1}, o_r \neq g_r$ holds with largest r value.
  - There can be many optimal solution, we took the one that has the most consecutive matching with $Greedy$ from the first selected activity

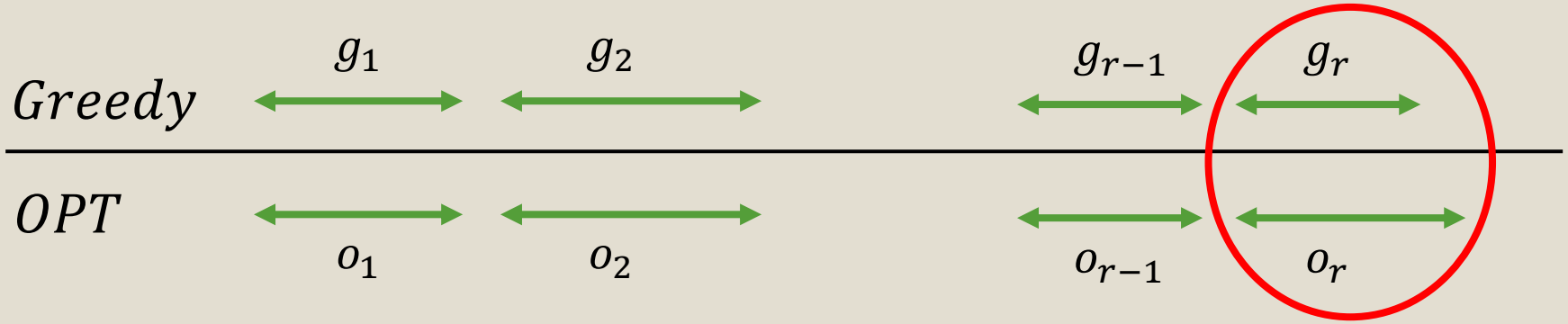


Another Optimality Proof

Theorem. The algorithm returns a largest subset of compatible activities.

Proof Sketch: Let $Greedy$ is our solution and OPT is an optimal solution.

- Case 1: $r > s$. Then, g_r does not exist, but greedy algorithm would pick another activity (e.g., it could pick o_r without causing overlaps)
- Case 2: $r \leq s$. Exchange o_s with g_r in OPT (As we did in Theorem 15.1). The new OPT shows that the old OPT does not have the largest r value, contradiction.



Yet Another Optimality Proof

Theorem. The algorithm returns a largest subset of compatible activities.

The previous proof is based on exchange argument

Another technique to prove optimality of greedy algorithms: 'Greedy Stays Ahead'

Show that, for all i, the i^{th} activity by Greedy has earlier finish time than the i^{th} choice of an arbitrary optimal solution

Thank You!