Box Packing

**Input:** $n$ boxes of capacities $c_1, c_2, \ldots, c_n$

$m$ items of sizes $s_1, s_2, \ldots, s_m$

Can put **at most 1** item in a box

Item $j$ can be put into box $i$ if $s_j \leq c_i$

**Output:** A way to put as many items as possible in the boxes.
Box Packing

**Input:** \( n \) boxes of capacities \( c_1, c_2, \cdots, c_n \)
\( m \) items of sizes \( s_1, s_2, \cdots, s_m \)

Can put at most 1 item in a box

Item \( j \) can be put into box \( i \) if \( s_j \leq c_i \)

**Output:** A way to put as many items as possible in the boxes.

**Example:**
- Box capacities: 60, 40, 25, 15, 12
- Item sizes: 45, 42, 20, 19, 16
- Can put 3 items in boxes: 45 \(\rightarrow\) 60, 20 \(\rightarrow\) 40, 19 \(\rightarrow\) 25
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an irrevocable decision using a “reasonable” strategy

Designing a Reasonable Strategy for Box Packing

- Q: Take box 1. Which item should we put in box 1?
Greedy Algorithm
- Build up the solutions in steps
- At each step, make an *irrevocable* decision using a “reasonable” strategy

Designing a Reasonable Strategy for Box Packing
- Q: Take box 1. Which item should we put in box 1?
- A: The item of the largest size that can be put into the box.
Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma**  The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.
Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma**  The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.

- Intuition: putting the item gives us the easiest residual problem.
Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma** The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.

- **Intuition**: putting the item gives us the easiest residual problem.
- **Formal proof via exchanging argument**: 
Lemma  There is an optimum solution in which box 1 contains the largest item it can hold.
Lemma  There is an optimum solution in which box 1 contains the largest item it can hold.

Proof.

- Let $j =$ largest item that box 1 can hold.
Lemma  There is an optimum solution in which box 1 contains the largest item it can hold.

Proof.
- Let $j =$ largest item that box 1 can hold.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
Lemma There is an optimum solution in which box 1 contains the largest item it can hold.

Proof.

- Let $j = $ largest item that box 1 can hold.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$:

$$\begin{align*}
S: \quad & \quad \text{box 1} \\
& \quad \text{item } j \\
& \quad \text{......}
\end{align*}$$
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j =$ largest item that box 1 can hold.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$: $s_{j'} \leq s_j$, and swapping gives another solution $S'$.
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j = \text{largest item that box 1 can hold}$.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$:

  $S'$:
  - box 1
  - item $j'$
  - item $j$
  - $s_{j'} \leq s_j$, and swapping gives another solution $S'$
  - $S'$ is also an optimum solution. In $S'$, $j$ is put into Box 1.
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.

**Analysis of Greedy Algorithm**

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.

**Analysis of Greedy Algorithm**

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
- **Trivial**: we decided to put Item $j$ into Box 1, and the remaining instance is obtained by removing Item $j$ and Box 1.
### Generic Greedy Algorithm
1. **while** the instance is non-trivial **do**
2. make the choice using the greedy strategy
3. reduce the instance

### Greedy Algorithm for Box Packing
1. \( T \leftarrow \{1, 2, 3, \ldots, m\} \)
2. **for** \( i \leftarrow 1 \) to \( n \) **do**
3. **if** some item in \( T \) can be put into box \( i \) **then**
4. \( j \leftarrow \) the largest item in \( T \) that can be put into box \( i \)
5. print(“put item \( j \) in box \( i \)”)
6. \( T \leftarrow T \setminus \{j\} \)
Why “Safety” + “Self-reduce” \(\implies\) Optimality?

- Let \(\text{BP}(B, T)\) denote a box-packing instance.
- \(\phi(1, 2, \ldots, m) \mapsto \{1, 2, \ldots, n, \text{NULL}\}\) denote packing strategy. e.g., \(\phi(2) = 3\) means item 2 is put into box 3.
- \(\text{val}(\phi) := \) the number of items packed by \(\phi\).
- \(\phi_g\): the packing strategy obtained by greedy algorithm.

Proof.

- Base case: When \(|B| = 1\) or \(|T| = 1\).
- Inductive case: (Hypothesis) Assume Greedy alg solves \(\text{BP}(B', T')\) optimally for \(|B'| = n - 1\) and \(|T'| = m - 1\).
Why “Safety” + “Self-reduce” $\iff$ Optimality?

**Proof.**

- (Induction) Wlog, let $\pi$ be the optimal solution matches our greedy sol on $\text{BP}(B, T)$, saying $\pi(j) = 1$.
- By self-reduce: $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$ is a smaller BP instance.
- $\pi$ and $\phi_g$ onto $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$, denoted as $\pi'$ and $\phi'_g$.
- By Inductive hypothesis, $\phi'_g$ is the optimal sol for $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$.
- $\text{val}(\pi) \geq \text{val}(\phi_g) = 1 + \text{val}(\phi'_g) \geq 1 + \text{val}(\pi') = \text{val}(\pi)$. 

\[\square\]
Running time

Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Greedy Algorithm for Box Packing

1: $T \leftarrow \{1, 2, 3, \cdots, m\}$
2: for $i \leftarrow 1$ to $n$ do
3: if some item in $T$ can be put into box $i$ then
4: $j \leftarrow$ the largest item in $T$ that can be put into box $i$
5: print(“put item $j$ in box $i$”)
6: $T \leftarrow T \setminus \{j\}$
Running time

**Generic Greedy Algorithm**

1: while the instance is non-trivial do  
2: make the choice using the greedy strategy  
3: reduce the instance

**Greedy Algorithm for Box Packing**

1: \( T \leftarrow \{1, 2, 3, \cdots, m\} \)  
2: for \( i \leftarrow 1 \) to \( n \) do  
3: if some item in \( T \) can be put into box \( i \) then  
4: \( j \leftarrow \) the largest item in \( T \) that can be put into box \( i \)  
5: print("put item \( j \) in box \( i \)"")  
6: \( T \leftarrow T \setminus \{j\} \)
Running time

**Generic Greedy Algorithm**

1. **while** the instance is non-trivial **do**
2. make the choice using the greedy strategy
3. reduce the instance

**Greedy Algorithm for Box Packing**

1. $T \leftarrow \{1, 2, 3, \ldots, m\}$
2. **for** $i \leftarrow 1$ to $n$ **do**
3. **if** some item in $T$ can be put into box $i$ **then**
4. $j \leftarrow$ the largest item in $T$ that can be put into box $i$
5. print(“put item $j$ in box $i$”)
6. $T \leftarrow T \setminus \{j\}$

- With sorted item-sizes and box-capacities, running time is $O(\max\{n, m\})$. 
GenericGreedyAlgorithm

1: \textbf{while} the instance is non-trivial \textbf{do} \\
2: make the choice using the greedy strategy \\
3: reduce the instance \\

\textbf{Lemma} Generic algorithm is correct \textbf{if and only if} the greedy strategy is safe.
Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Lemma  Generic algorithm is correct if and only if the greedy strategy is safe.

- Greedy strategy is safe: we will not miss the optimum solution
Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Lemma  Generic algorithm is correct if and only if the greedy strategy is safe.

- Greedy strategy is safe: we will not miss the optimum solution
- Greedy strategy is not safe: we will miss the optimum solution for some instance, since the choices we made are irrevocable.
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an irreversible decision using a “reasonable” strategy
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Def.** As strategy is “safe” if there is an optimal solution that is “consistent” with the decision made according to the strategy.
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an *irrevocable* decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

Def. A strategy is “safe” if there is always an optimum solution that is “consistent” with the decision made according to the strategy.
l et $S$ be an arbitrary optimum solution.

if $S$ is consistent with the greedy choice, done.

otherwise, show that it can be modified to another optimum solution $S'$ that is consistent with the choice.
let $S$ be an arbitrary optimum solution.

if $S$ is consistent with the greedy choice, done.

otherwise, show that it can be modified to another optimum solution $S'$ that is consistent with the choice.

The procedure is not a part of the algorithm.
Outline

1 Toy Example: Box Packing

2 Interval Scheduling

3 Offline Caching
   - Heap: Concrete Data Structure for Priority Queue

4 Data Compression and Huffman Code

5 Summary
Interval Scheduling

**Input:** $n$ jobs, job $i$ with start time $s_i$ and finish time $f_i$

$i$ and $j$ are **compatible** if $[s_i, f_i)$ and $[s_j, f_j)$ are disjoint

**Output:** A maximum-size subset of mutually compatible jobs
Interval Scheduling

**Input:** \( n \) jobs, job \( i \) with start time \( s_i \) and finish time \( f_i \). 

\( i \) and \( j \) are compatible if \([s_i, f_i)\) and \([s_j, f_j)\) are disjoint.

**Output:** A maximum-size subset of mutually compatible jobs.
Which of the following strategies are safe?
Which of the following strategies are safe?

Schedule the job with the smallest size?
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?

Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!

![Diagram showing intervals and scheduling]

0 1 2 3 4 5 6 7 8 9

- Interval scheduling
- Job scheduling
- Greedy algorithm
- Interval overlap
- Resource allocation

22/85
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?

- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs?
Which of the following strategies are safe?

- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!

![Diagram of intervals]

23/85
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
- Schedule the job with the earliest finish time?
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
  - Schedule the job conflicting with smallest number of other jobs? No!
  - Schedule the job with the earliest finish time? Yes!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
  - Schedule the job conflicting with smallest number of other jobs? No!
  - Schedule the job with the earliest finish time? Yes!

![Diagram showing intervals and scheduling decisions](image-url)
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
- Schedule the job with the earliest finish time? Yes!
Greedy Algorithm for Interval Scheduling

**Lemma** It is safe to schedule the job \( j \) with the earliest finish time: There is an optimum solution where the job \( j \) with the earliest finish time is scheduled.

**Proof.**
**Lemma** It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**
- Take an arbitrary optimum solution $S$
**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**
- Take an arbitrary optimum solution $S$
- If it contains $j$, done

$S$:
Greedy Algorithm for Interval Scheduling

**Lemma** It is safe to schedule the job \( j \) with the earliest finish time: There is an optimum solution where the job \( j \) with the earliest finish time is scheduled.

**Proof.**

- Take an arbitrary optimum solution \( S \)
- If it contains \( j \), done

\[
S: \quad \text{[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]}
\]

\[
j: \quad \text{[ ] [ ]}
\]
**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**

- Take an arbitrary optimum solution $S$
- If it contains $j$, done
- Otherwise, replace the first job in $S$ with $j$ to obtain another optimum schedule $S'$. 

$S$: 

```
                      
```

$j$: 

```
      
```
**Lemma** It is safe to schedule the job \( j \) with the earliest finish time: There is an optimum solution where the job \( j \) with the earliest finish time is scheduled.

**Proof.**

- Take an arbitrary optimum solution \( S \)
- If it contains \( j \), done
- Otherwise, replace the first job in \( S \) with \( j \) to obtain another optimum schedule \( S' \).
**Lemma** It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

- What is the remaining task after we decided to schedule $j$?
- Is it another instance of interval scheduling problem?

![Diagram of interval scheduling](image)
Greedy Algorithm for Interval Scheduling

Lemma It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

- What is the remaining task after we decided to schedule $j$?
- Is it another instance of interval scheduling problem? Yes!
**Lemma** It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

- What is the remaining task after we decided to schedule $j$?
- Is it another instance of interval scheduling problem? Yes!
**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

- What is the remaining task after we decided to schedule $j$?
- Is it another instance of interval scheduling problem? Yes!
Greedy Algorithm for Interval Scheduling

Schedule($s, f, n$)

1: $A \leftarrow \{1, 2, \cdots , n\}, S \leftarrow \emptyset$
2: while $A \neq \emptyset$ do
3: \hspace{1em} $j \leftarrow \arg \min_{j' \in A} f_{j'}$
4: \hspace{1em} $S \leftarrow S \cup \{j\}$; $A \leftarrow \{j' \in A : s_{j'} \geq f_{j}\}$
5: return $S$
Greedy Algorithm for Interval Scheduling

Schedule\((s, f, n)\)

1: \(A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset\)
2: \textbf{while} \(A \neq \emptyset\) \textbf{do}
3: \(j \leftarrow \text{arg min}_{j' \in A} f_{j'}\)
4: \(S \leftarrow S \cup \{j\}; \ A \leftarrow \{j' \in A : s_{j'} \geq f_j\}\)
5: \textbf{return} \(S\)
Greedy Algorithm for Interval Scheduling

**Schedule**(*s, f, n*)

1. \( A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset \)
2. **while** \( A \neq \emptyset \) **do**
3. \( j \leftarrow \arg \min_{j' \in A} f_{j'} \)
4. \( S \leftarrow S \cup \{j\}; A \leftarrow \{j' \in A : s_{j'} \geq f_j\} \)
5. **return** \( S \)
Greedy Algorithm for Interval Scheduling

Schedule\((s, f, n)\)

1: \(A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset\)
2: \(\textbf{while} \ A \neq \emptyset \ \textbf{do}\)
3: \(j \leftarrow \arg \min_{j' \in A} f_{j'}\)
4: \(S \leftarrow S \cup \{j\}; \ A \leftarrow \{j' \in A : s_{j'} \geq f_j\}\)
5: \(\textbf{return} \ S\)
Greedy Algorithm for Interval Scheduling

**Schedule**(*s, f, n*)

1. \( A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset \)
2. **while** \( A \neq \emptyset \) **do**
3. \( j \leftarrow \text{arg min}_{j' \in A} f_{j'} \)
4. \( S \leftarrow S \cup \{j\}; A \leftarrow \{j' \in A : s_{j'} \geq f_j\} \)
5. **return** \( S \)
Greedy Algorithm for Interval Scheduling

Schedule\((s, f, n)\)

1. \( A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset \)
2. \textbf{while} \( A \neq \emptyset \) \textbf{do}
3. \hspace{1em} \( j \leftarrow \arg\min_{j' \in A} f_{j'} \)
4. \hspace{1em} \( S \leftarrow S \cup \{j\}; \text{ } A \leftarrow \{j' \in A : s_{j'} \geq f_j\} \)
5. \textbf{return} \( S \)
Greedy Algorithm for Interval Scheduling

**Schedule**\((s, f, n)\)

1: \(A \leftarrow \{1, 2, \cdots, n\}, S \leftarrow \emptyset\)
2: \textbf{while} \(A \neq \emptyset\) \textbf{do}
3: \(j \leftarrow \arg \min_{j' \in A} f_{j'}\)
4: \(S \leftarrow S \cup \{j\}; A \leftarrow \{j' \in A : s_{j'} \geq f_j\}\)
5: \textbf{return} \(S\)

Running time of algorithm?
Greedy Algorithm for Interval Scheduling

\textbf{Schedule}(s, f, n)

1: \( A \leftarrow \{1, 2, \ldots, n\}, S \leftarrow \emptyset \)
2: \textbf{while} \( A \neq \emptyset \) \textbf{do}
3: \( j \leftarrow \arg \min_{j' \in A} f_{j'} \)
4: \( S \leftarrow S \cup \{j\}; A \leftarrow \{j' \in A : s_{j'} \geq f_j\} \)
5: \textbf{return} \( S \)

Running time of algorithm?
- Naive implementation: \( O(n^2) \) time