Approximation Algorithms
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Diagram of a triangle with labeled sides](image)
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Graph showing triangle inequality](image)

**Approximation Algorithm:**
Find the Minimum Cost Network
**Approximation Algorithms**

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Graph with vertices and edges labeled with costs](image)

**Approximation Algorithm:**

Find the Minimum Cost Network
Lay out a plan for traversing the entire network
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Graph showing the triangle inequality with distances labeled: 32, 24, 24, 48, 40, 54, 56, 58, 60.]

**Approximation Algorithm:**
- Find the Minimum Cost Network
- Lay out a plan for traversing the network
- Move from vertex to vertex according to plan
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Approximation Algorithm:
Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Approximation Algorithm:

Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Approximation Algorithm:
Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan – take short cut
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Approximation Algorithm:
Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan – take short cut
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

```
Approximation Algorithm:
Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan - finish
```
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Approximation Algorithm:
Find the Minimum Cost Network
Lay out a plan for traversing the network
Move from vertex to vertex according to plan - finish
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Observe: cost of minimum cost network of a connected graph is less than the cost of a minimum cost Hamiltonian cycle.

Cost = a + b + c + d + e + f
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Observe: cost of minimum cost network of a connected graph is less than the cost of a minimum cost Hamiltonian cycle.

Cost = $a + b + c + d + e$

This is a spanning tree of cost no less than that of a minimum cost network.
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Observe: cost of this tour is no greater than cost of this tour ... (triangle inequality)

... and cost of this tour is less than double best TSP tour
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Example that almost hits twice optimum:

Optimum solution has cost $7 - 2\epsilon$ – follow red lines
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Example that almost hits twice optimum:

Minimum cost network – follow blue lines
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Example that almost hits twice optimum:

Minimum cost network – path to follow
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Example that almost hits twice optimum:

Minimum cost network – take shortcuts – cost is 11-\(\epsilon\)
Approximation Algorithms

How far away from the optimal can the approximation algorithm get?

Example that almost hits twice optimum:

Optimum cost: $2n - 2\epsilon$
Approx alg cost: $4n - 1 - 2\epsilon$

Ratio: $2 + O(1/n)$
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofides Algorithm:
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofides Algorithm:
Find the Minimum Cost Network
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofide Algorithm:
Find the Minimum Cost Network
Find minimum perfect matching over vertices of odd degree
Combine edges
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofides Algorithm:
Find the Minimum Cost Network
Find minimum perfect matching over vertices of odd degree
Combine edges – visit vertices as before
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Graph showing traveling salesman problem and Christofides Algorithm example]

**Christofides Algorithm:**
- Find the Minimum Cost Network
- Find minimum perfect matching over vertices of odd degree
- Combine edges – visit vertices as before
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofides Algorithm:
Find the Minimum Cost Network
Find minimum perfect matching over vertices of odd degree
Combine edges – visit vertices as before
Approximation Algorithms

**Traveling Salesman:** triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

**Example:**

![Graph with edges and weights]

**Christofides Algorithm:**
Find the Minimum Cost Network
Find minimum perfect matching over vertices of odd degree
Combine edges – visit vertices as before – with shortcut
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Christofides Algorithm:
Find the Minimum Cost Network
Find minimum perfect matching over vertices of odd degree
Combine edges – visit vertices as before

40+48+24+32+56 = 200
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Analysis:
Minimum Cost Network: cost no greater than optimum
Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Analysis:
Minimum Cost Network: cost no greater than optimum
Min cost perfect matching: cost no greater than 1/2 optimum
Approximation Algorithms

Traveling Salesman: triangle inequality holds – cost of any side of a triangle is no greater than the sum of the costs of the other two sides.

Example:

Analysis:
Minimum Cost Network: cost no greater than optimum
Min cost perfect matching: cost no greater than 1/2 optimum
Both together, with shortcut: no greater than 3/2 optimum
How does it do on the example given before?
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching
How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

Start here

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

2n+1 vertices

Minimum Cost Network + Minimum Perfect Matching
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
How does it do on the example given before?

Minimum Cost Network+Minimum Perfect Matching
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2\(n+1\) vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network+Minimum Perfect Matching

2n+1 vertices
Approximation Algorithms

How does it do on the example given before?

Minimum Cost Network + Minimum Perfect Matching

\[ \text{cost} = 4\left(\frac{n-2}{2} + (n+3)\right) = 3n-1 \]

Ratio = \(3/2 + O(1/n)\)
Approximation Algorithms

Traveling Salesman, triangle inequality does not hold
Suppose there is a polytime approx algorithm $A$ for TSP where $A(I)/OPT(I) < K$, $K$ a constant
Then revisit earlier transformation from HC to TSP:

If $G$ has a HC, then $OPT(H) = n$
If $G$ has no HC then $OPT(H) > Kn$
Since $A(H) < K*OPT(H)$, if $A(H) < Kn$ then $G$ has HC
If $A(H) > Kn$, then $G$ cannot have a HC
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25

$k = 5$
Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: (first fit approximation algorithm)

Objects: $0.55$, $0.55$, $0.35$, $0.20$, $0.20$, $0.45$, $0.45$, $0.25$, $0.25$, $0.25$, $0.20$, $0.25$, $k = 5$
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example: (first fit approximation algorithm)**

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.25, 0.20, 0.25  

$k = 5$
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, **0.35**, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.25, 0.20, 0.25 $k = 5$
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into \( k \) bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25 \( k = 5 \)

![Diagram of bin packing example]
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example: (first fit approximation algorithm)**

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25 $k = 5$

![Diagram of bin packing with objects placed in bins](image)
**Approximation Algorithms**

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, **0.45**, 0.45, 0.25, 0.25, 0.25, 0.25, 0.20, 0.25 $\quad k = 5$
**Approximation Algorithms**

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, **0.45**, 0.25, 0.25, 0.25, 0.25, 0.20, 0.25  
$k = 5$
**Approximation Algorithms**

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into \( k \) bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example: (first fit approximation algorithm)**

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25 \( \quad k = 5 \)
**Approximation Algorithms**

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, **0.25**, 0.25, 0.20, 0.25  

$k = 5$
Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: (first fit approximation algorithm)

Objects: 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25

$k = 5$
**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into \( k \) bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Objects:** 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20

\( k = 5 \)
Approximation Algorithms

Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: (first fit approximation algorithm)

Objects: 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25  
$k = 5$
Approximation Algorithms

Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: Optimum Solution

Objects: 0.55, 0.55, 0.35, 0.20, 0.20, 0.45, 0.45, 0.25, 0.25, 0.25, 0.20, 0.25  
$k = 4$
Approximation Algorithms

**Bin packing:** given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

**Example:** (first fit approximation algorithm)

**Observe:** at any time it is not possible for two bins to be at most $1/2$ full

The 0.25 item would have gone into the left bin
Approximation Algorithms

Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: (first fit approximation algorithm)

Observe: So, if $k$ bins are used $\sum s_i > (k-1)/2$. But $\sum s_i$ is a lower bound on the optimum number (OPT) so $OPT > \sum s_i > (k-1)/2$ and $k < 2OPT + 1$
Bin packing: given objects of various sizes, all no greater than 1, can objects be placed into $k$ bins so that the sum of sizes of objects in each bin is no greater than 1?

Example: (first fit approximation algorithm)

Optimum = 6$m$ bins, Approx. Alg = 10$m$ bins
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20  \( k = 5 \)
Approximation Algorithms

**First fit decreasing:** consider objects in decreasing order of size, put each object into the first bin where it fits

**Example:**

**Objects:** 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \( k = 5 \)
Approximation Algorithms

**First fit decreasing:** consider objects in decreasing order of size, put each object into the first bin where it fits

**Example:**

**Objects:** 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 $k = 5$
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20  $k = 5$
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, \textbf{0.45}, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \quad k = 5
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 $k = 5$
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \( k = 5 \)
Approximation Algorithms

**First fit decreasing:** consider objects in decreasing order of size, put each object into the first bin where it fits

**Example:**

**Objects:** 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20  \( k = 5 \)
First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \( k = 5 \)
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, k = 5

0.20, 0.20, 0.20

0.25
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Example:
Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \( k = 5 \)
Approximation Algorithms

**First fit decreasing:** consider objects in decreasing order of size, put each object into the first bin where it fits

**Example:**

**Objects:** 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, **0.20**, 0.20

$k = 5$
Approximation Algorithms

First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits.

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20

$k = 5$
First fit decreasing: consider objects in decreasing order of size, put each object into the first bin where it fits

Worst case: $k < (11/9)\text{OPT}$

Example:

Approx. Alg $= 11m$ bins

Optimum $= 9m$ bins
**Approximation Algorithms**

**Best fit decreasing:** consider objects in decreasing order of size, put each object into the fullest bin that still has room.

**Example:**

**Objects:** 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \(k = 5\)
Approximation Algorithms

**Best fit decreasing:** consider objects in decreasing order of size, put each object into the fullest bin that still has room

**Worst case:** $k < (11/9)\text{OPT}$
Almost Worst fit decreasing: consider objects in decreasing order of size, put object into the 2\textsuperscript{nd} emptiest bin with room

Example:

Objects: 0.55, 0.55, 0.45, 0.45, 0.35, 0.25, 0.25, 0.25, 0.25, 0.20, 0.20, 0.20 \quad k = 5
Approximation Algorithms

The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $\text{size}: A \rightarrow \mathbb{N}^+$, a function $\text{value}: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} \text{size}(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value:size, use greedy method to place
16/3, 19/6, 9/4, 14/7, 12/8  $B = 17$
Approximation Algorithms

The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $size: A \rightarrow \mathbb{N}^+$, a function $value: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} size(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value/size, use greedy method to place $16/3, 19/6, 9/4, 14/7, 12/8$ \quad $B = 14$

\[ \text{OK} \]

Subset = \{[3;16]\}
Approximation Algorithms

The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $\text{size} : A \rightarrow \mathbb{N}^+$, a function $\text{value} : A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} \text{size}(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value/size, use greedy method to place

$16/3, 19/6, 9/4, 14/7, 12/8$  $B = 8$

OK

Subset = \{[3;16], [6;19]\}
The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $\text{size}: A \rightarrow \mathbb{N}^+$, a function $\text{value}: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} \text{size}(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value/size, use greedy method to place $16/3$, $19/6$, $9/4$, $14/7$, $12/8$ $\quad B = 4$

OK

Subset = \{[3;16],[6;19],[4;9]\}
The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $size: A \rightarrow \mathbb{N}^+$, a function $value: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} size(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value/size, use greedy method to place $16/3, 19/6, 9/4, 14/7, 12/8$  

$B = 4$

NG

Subset = \{[3;16],[6;19],[4;9]\}
The Knapsack Problem:

**Instance:** a set $A$ of objects, a function $size: A \rightarrow \mathbb{N}^+$, a function $value: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} size(a) \leq B$

**Example:** $A = \{[3;16], [6;19], [4;9], [8;12], [7;14]\}$, $B = 17$

Order by ratio value/size, use greedy method to place $16/3, 19/6, 9/4, 14/7, 12/8$  

$B = 4$

$NG$

Subset $= \{[3;16], [6;19], [4;9]\}$  
Total Value $= 44$
Approximation Algorithms

The Knapsack Problem:
Polynomial Time Approximation Scheme

**Instance:** a set $A$ of objects, a function $\text{size}: A \rightarrow \mathbb{N}^+$, a function $\text{value}: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

**Output:** a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} \text{size}(a) \leq B$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3,6,4,8,7}</td>
<td>0</td>
<td>16</td>
<td>9</td>
<td>19</td>
<td>25</td>
<td>12</td>
<td>35</td>
<td>30</td>
<td>28</td>
<td>21</td>
<td>44</td>
<td>39</td>
<td>37</td>
<td>49</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{3,6,4,8}</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>16</td>
<td>9</td>
<td>-1</td>
<td>19</td>
<td>25</td>
<td>12</td>
<td>35</td>
<td>28</td>
<td>21</td>
<td>44</td>
<td>27</td>
<td>37</td>
<td>-1</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>{3,6,4}</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>16</td>
<td>9</td>
<td>-1</td>
<td>19</td>
<td>25</td>
<td>-1</td>
<td>35</td>
<td>28</td>
<td>-1</td>
<td>-1</td>
<td>44</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>{3,6}</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>16</td>
<td>-1</td>
<td>-1</td>
<td>19</td>
<td>-1</td>
<td>-1</td>
<td>35</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>{3}</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>16</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Subset = \{[3;16],[6;19],[7;14]\}  **Complexity:** $O(nB)$
Approximation Algorithms

The Knapsack Problem:
Polynomial Time Approximation Scheme

Example: \{[3:2],[4:3],[2:5],[1:3],[3:4]\}, \(B=6\)

\[
\text{TABLE}[i,j] = \min \text{ size of subset } \{1,\ldots,i\} \text{ that gives value } j
\]

\[
\text{TABLE}[i,j] = \min\{\text{TABLE}[i-1,j], \text{TABLE}[i-1, j-value(a_i)]+size(a_i)\}
\]

| \{3,4,2,1,3\} | 0 | -1 | 3 | 1 | 3 | 2 | 5 | 4 | 3 | 5 | 6 | 7 | 6 | 10 | 9 | 10 | -1 | 13 |
| \{3,4,2,1\} | 0 | -1 | 3 | 1 | -1 | 2 | 5 | 5 | 3 | -1 | 6 | 7 | -1 | 10 | -1 | -1 | -1 | -1 |
| \{3,4,2\} | 0 | -1 | 3 | 4 | -1 | 2 | -1 | 5 | 6 | -1 | 9 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| \{3,4\} | 0 | -1 | 3 | 4 | -1 | 7 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| \{3\} | 0 | -1 | 3 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Subset = \{[2:5],[1:3],[3:4]\}  \[\text{Complexity: } O(n^2 V_{\max})\]
Approximation Algorithms

The Knapsack Problem:
Polynomial Time Approximation Scheme

Instance: a set $A$ of objects, a function $size: A \rightarrow \mathbb{N}^+$, a function $value: A \rightarrow \mathbb{N}^+$, and an integer capacity $B$

Output: a subset $A^* \subseteq A$ whose total value is maximum over all subsets of $A' \subseteq A$ such that $\sum_{a \in A'} size(a) \leq B$

Algorithm: Dynamic program as before but sizes are given as $value'(a) = \lceil value(a)/K \rceil$ for some $K$ (TBD)

So complexity is $O(n^2 V_{\text{max}} / K)$

Since optimal solution cannot contain more than $n$ items

$OPT(I) - K*OPT(I') < Kn$

Choose $K = V_{\text{max}} / (k+1)n$

$A_k(I) > OPT(I) - Kn = OPT(I) - V_{\text{max}} / (k+1)$

$OPT(I) > V_{\text{max}}$

$OPT(I) / A_k(I) < 1 + 1/k$  

Complexity: $O(n^3 (k+1))$