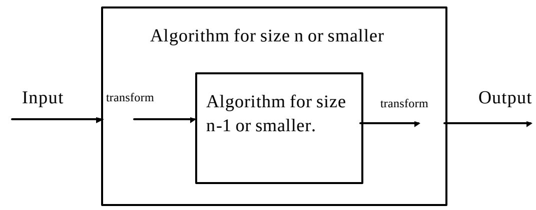
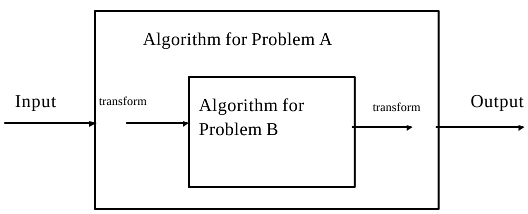
Algorithm Design Techniques

- Induction
- Divide and Conquer
- Dynamic Programming
- **Reduction**
- Greedy

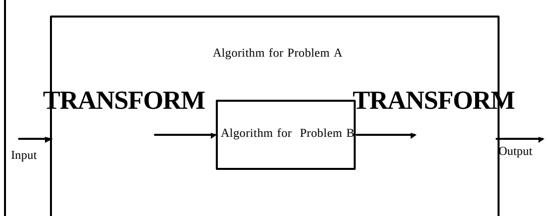
Self-Reduction



Reduction: A $\propto$ B



Reduction: A $\propto$ B



Some reductions we've seen

- Sorting $\propto$ Find-max
- General Selection $\propto$ Find-median
- Almost every dynamic programming problem

To solve A

- One Stage
  - Self-Reduction
- Two Stage
  - Define B
  - Reduce A to B
  - Solve B using self-reduction

Longest Increasing Subsequence

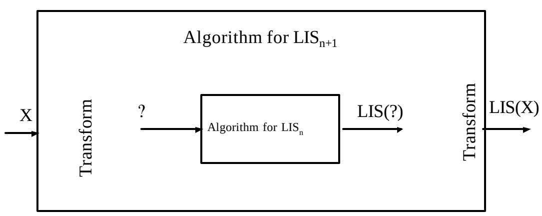
- Input: Sequence of integers X: $x_1, x_2, \ldots, x_n$
- Output: Longest increasing subsequence of X; i.e. a subsequence Z: $z_1, z_2, \ldots, z_k$ such that $z_i < z_{i+1}$ for each i:1…k-1.

Example

- 1, -3, 2, 10, 8, 23, -2, 17, 5

$LIS_{n+1} \propto LIS_n$

Don't know how to do it!!!



To solve A

- **Define B** (Strengthen the inductive hypothesis)
- Reduce A to B
- Solve B using self-reduction

LIS and Modified LIS

- Input: Sequence of integers X: $x_1, x_2, \ldots, x_n$
- Output: Longest increasing subsequence

- Input: Sequence of integers X: $x_1, x_2, \ldots, x_n$
- Output: For each i:1…n, a longest increasing subsequence of $x_1, \ldots, x_i$ that ends in x_i.

$MLIS(x_1, \ldots, x_n)$

$MLIS(x_1, \ldots, x_n)$ =

1. LIS of x_1 that ends in x_1
2. LIS of x_1, x_2 that ends in x_2

⋮

n-1. LIS of $x_1, \ldots, x_{n-1}$ that ends in x_{n-1}

n. LIS of $x_1, \ldots, x_n$ that ends in x_n

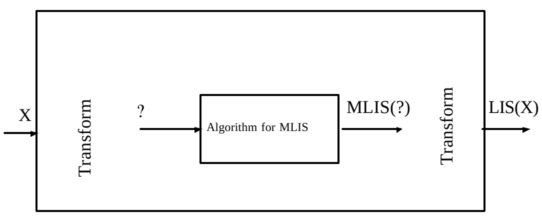
Example

- 1, -3, 2, 10, 8, 23, -2, 17, 5

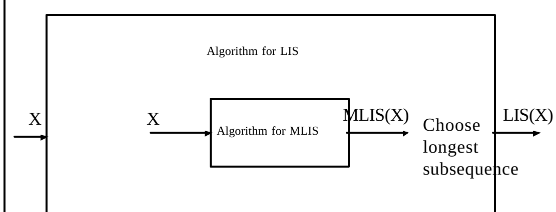
To solve A

- Define B
- **Reduce A to B**
- Solve B using self-reduction

LIS ∝ MLIS



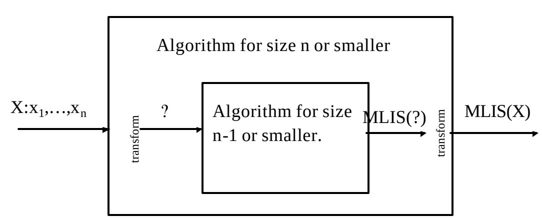
LIS ∝ MLIS



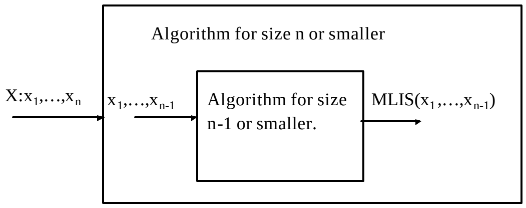
To solve A

- Define B
- Reduce A to B
- **Solve B using self-reduction**

MLIS Self-Reduction



MLIS Self-Reduction



$MLIS(x_1,\ldots,x_n)$

$MLIS(x_1,\ldots,x_n)$ =

1. LIS of x_1 that ends in x_1
2. LIS of x_1, x_2 that ends in x_2

⋮

n-1. LIS of $x_1, \ldots , x_{n-1}$ that ends in x_{n-1}
n. LIS of $x_1, \ldots , x_n$ that ends in x_n

$MLIS(x_1,\ldots,x_n)$

$MLIS(x_1,\ldots,x_n)$ = $MLIS(x_1, \ldots , x_{n-1})$

1. LIS of x_1 that ends in x_1
2. LIS of x_1, x_2 that ends in x_2

⋮

n-1. LIS of $x_1, \ldots , x_{n-1}$ that ends in x_{n-1}
n. LIS of $x_1, \ldots , x_n$ that ends in x_n

$MLIS(x_1,\ldots,x_n)$

$MLIS(x_1,\ldots,x_n)$ =

1. LIS of x_1 that ends in x_1
2. LIS of x_1, x_2 that ends in x_2

⋮

n-1. LIS of $x_1, \ldots , x_{n-1}$ that ends in x_{n-1}
n. LIS of $x_1, \ldots , x_n$ that ends in x_n

How can we produce this?

Construct $MLIS(x_1,\ldots,x_n)$

$MLIS(x_1,\ldots,x_n)$ =

1) $MLIS(x_1,\ldots,x_{n-1})$ plus
2) Choose longest $LIS(x_1,\ldots,x_j)$ ending in x_j $(j<n)$ such that $x_j<x_n$. Append x_n.

Recap: To solve A

- Define B
- Reduce A to B
- Solve B using self-reduction

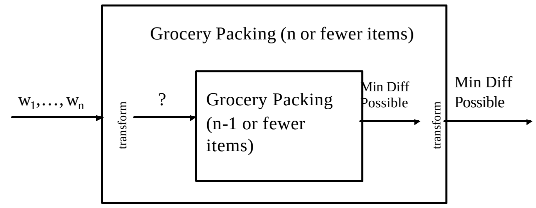
Grocery Bags

How should we pack n items weighing w_1, w_2,…, w_n ($w_i \leq W$) in two bags so as to minimize the difference in the weights of the bags?

Or even simpler: What is the smallest possible weight difference?

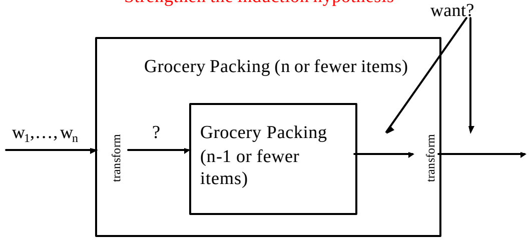
Self-Reduction

I don't know how to make this work!



Self-Reduction

Strengthen the induction hypothesis



Problem B

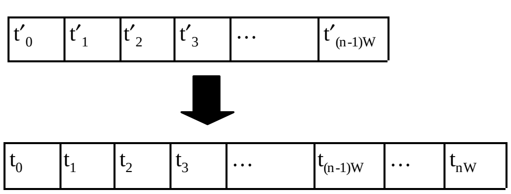
- Input: Weights $w_1, w_2, \ldots, w_n$
- Output: A binary vector T:

  T[i] = 1 if some subset of the weights sum to i

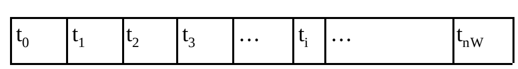
  T[i] = 0 otherwise

  for i=0,…,nW

Transform



Transform

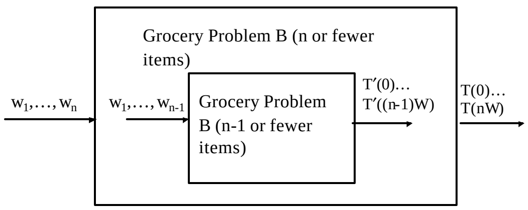


Set t_i= 1 if _______________ or ___________________

Else t_i=0

Self-Reduction: Problem B

What are the base cases?



Algorithm B

We'll use this later

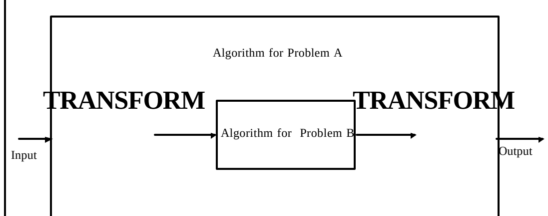
```
// initialize
t[0]=1
for i=1,…,nW
    t[i]=0

// update
for i=1,…,n
    for j=W*i to w_i
        if t[j]=0 and t[j-w_i]=1 then t[j]=1
```


Algorithm B

- Is it correct?
- Is it efficient?

Reduction: A $\propto$ B



Algorithm A

Use Algorithm B to compute t[0]…t[nW]

Let $S=\Sigma w_i$

(Note: t[0..S] is symmetric about S/2)

Let j be the closest index to S/2 such that t[j]=1

Return |j-S/2|

Grocery Bags

What about this problem?

How should we pack n items weighing $w_1, w_2, \ldots, w_n$ ($w_i \le W$) in two bags so as to minimize the difference in the weights of the bags?

Or even simpler: What is the smallest possible weight difference?

Algorithm B: Pack Bags

// initialize
t[0]=1
for i=1,….,nW
    t[i]=0, $\boxed{\text{b[i]=0}}$

// update
for i=1,…,n
    for j=W*i to w_i
        if t[j]=0 and t[j-w_i]=1 then t[j]=1 & $\boxed{\text{b[j]=i}}$

Algorithm A: Pack Bags

Cont. … Let j be the closest index to S/2 such that t[j]=1
Bag 1 = Bag 2 = empty
While j>0
    Put item b[j] in Bag 1
    j -= b[j]
Put unpacked items in Bag 2
Return

Algorithm A

- Is it correct?
- Is it efficient?