Greedy Algorithms: Main Ideas

Michael Levin

Higher School of Economics

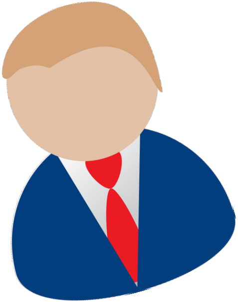
Algorithmic Toolbox
Data Structures and Algorithms

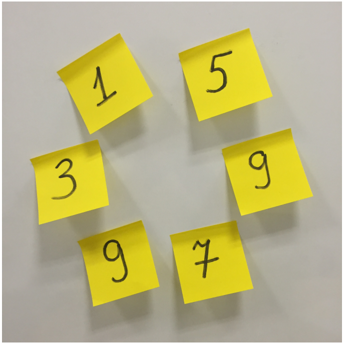
Outline

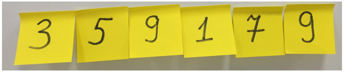
Learning objectives

- Come up with a greedy algorithm yourself

Job Interview







Largest Number

Toy problem

What is the largest number that consists of digits 3, 9, 5, 9, 7, 1? Use all the digits.

Examples

$$359179, 537991, 913579, \ldots$$

Correct answer

997531

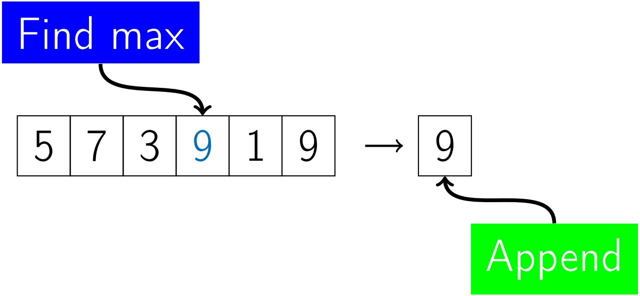
Greedy Strategy

Find max

5	7	3	9	1	9	$\longrightarrow$

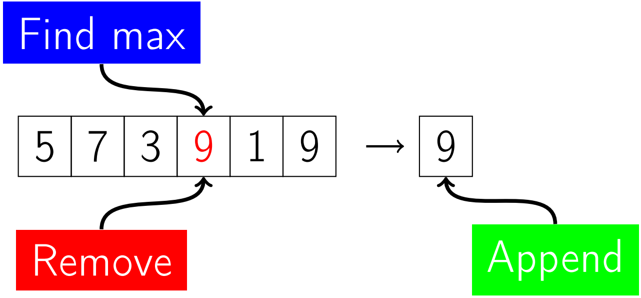
- Find max digit

Greedy Strategy



- Find max digit
- Append it to the number

Greedy Strategy



- Find max digit
- Append it to the number
- Remove it from the list of digits

Greedy Strategy

Find max

5	7	3		1	9	$\longrightarrow$	9

Remove

Append

- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy

Find max

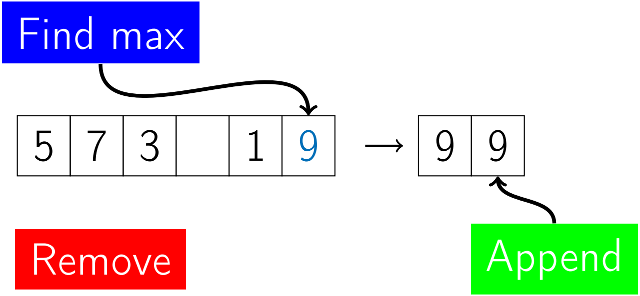
5	7	3		1	9	→	9

Remove

Append

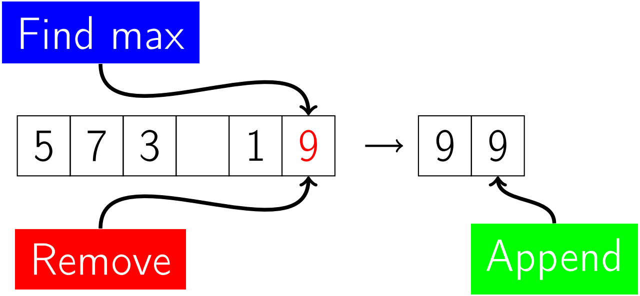
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



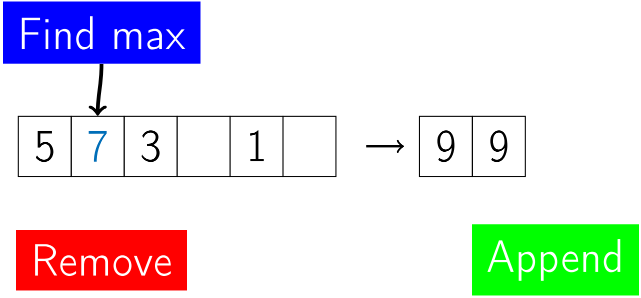
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



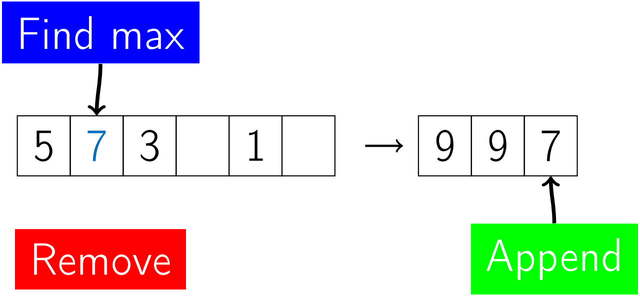
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



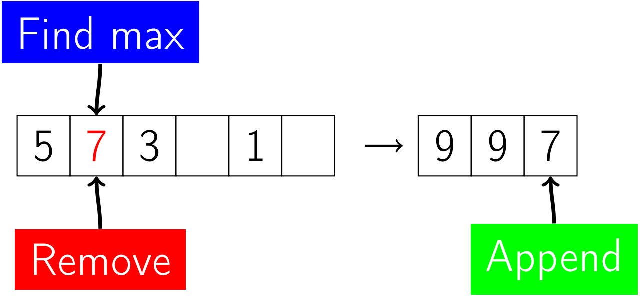
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



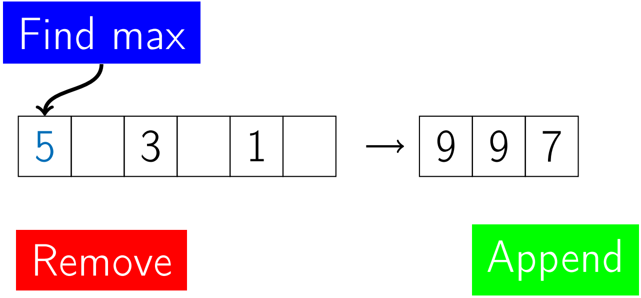
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



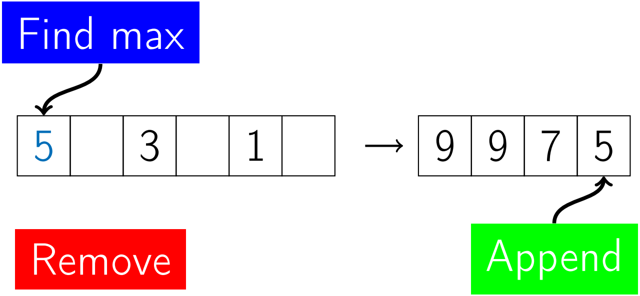
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



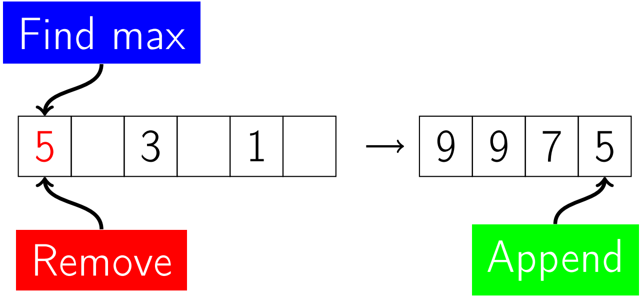
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



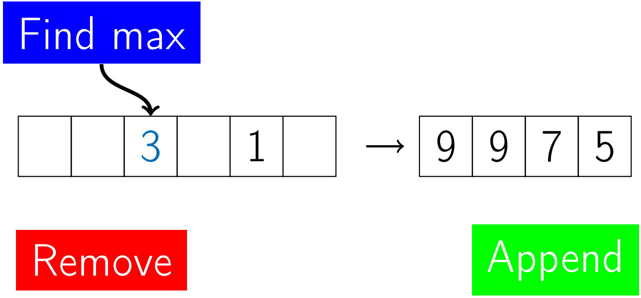
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



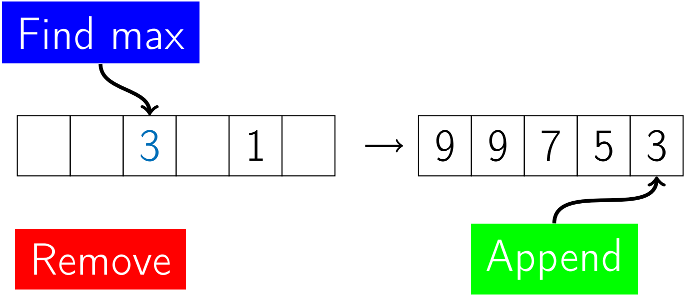
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



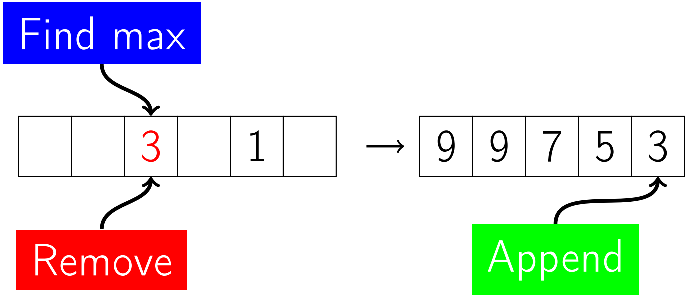
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



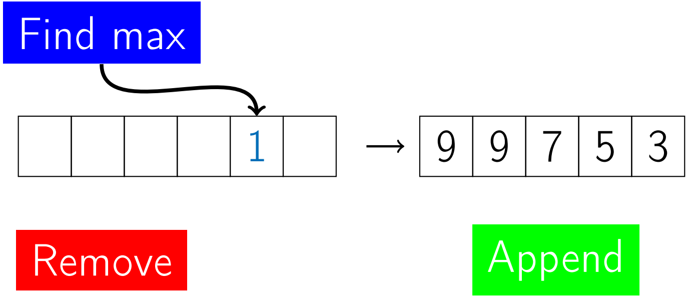
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



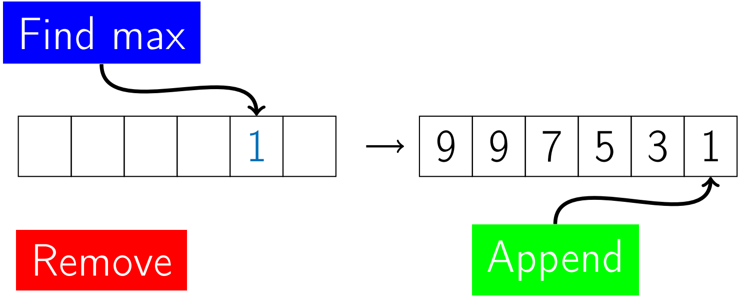
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



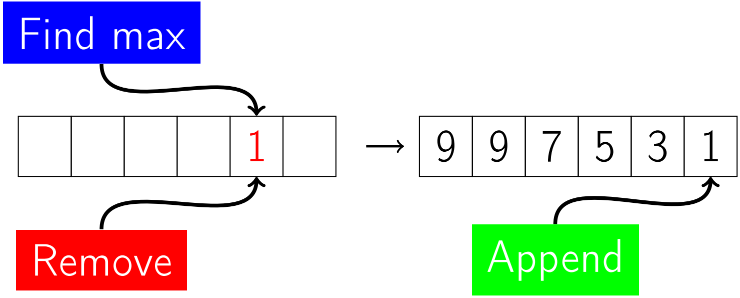
- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy



- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Greedy Strategy

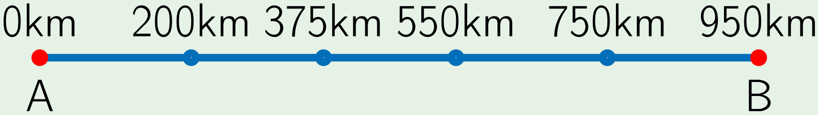
5	7	3	9	1	9	$\longrightarrow$	9	9	7	5	3	1

- Find max digit
- Append it to the number
- Remove it from the list of digits
- Repeat while there are digits in the list

Outline

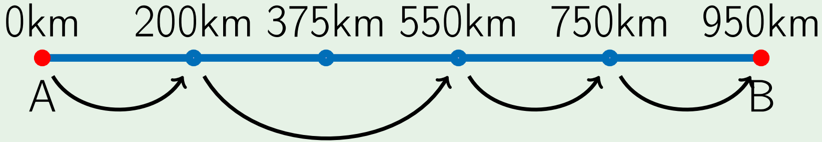
Car Fueling

Distance with full tank = 400km



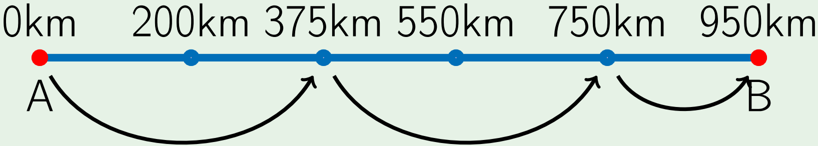
Car Fueling

Distance with full tank = 400km



Car Fueling

Distance with full tank = 400km



Minimum number of refills = 2

Car Fueling

Input: A car which can travel at most L kilometers with full tank, a source point A, a destination point B and n gas stations at distances $x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_n$ in kilometers from A along the path from A to B.

Output: The minimum number of refills to get from A to B, besides refill at A.

Greedy Strategy

- Make some greedy choice
- Reduce to a smaller problem
- Iterate

Greedy Choice

- Refill at the the closest gas station
- Refill at the farthest reachable gas station
- Go until there is no fuel

Greedy Algorithm

- Start at A
- Refill at the farthest reachable gas station G
- Make G the new A
- Get from new A to B with minimum number of refills

Definition

Subproblem is a similar problem of smaller size.

Subproblem

Examples

- `LargestNumber`$(3, 9, 5, 9, 7, 1) =$ "9" $+$ `LargestNumber`$(3, 5, 9, 7, 1)$
- Min number of refills from A to B $=$ first refill at G $+$ min number of refills from G to B

Safe Move

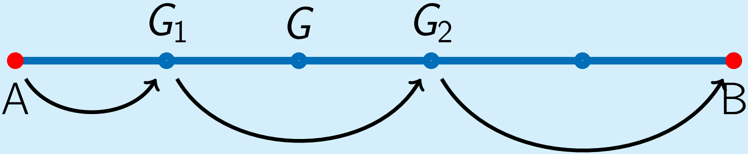
Definition

A greedy choice is called safe move if there is an optimal solution consistent with this first move.

Lemma

To refill at the farthest reachable gas station is a safe move.

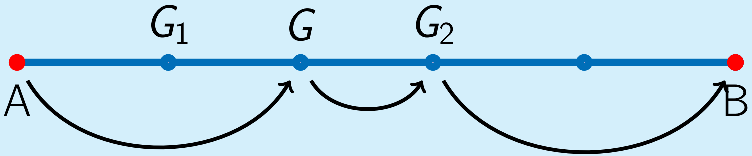
Proof



First case: G is closer than G_2

Refill at G instead of G_1

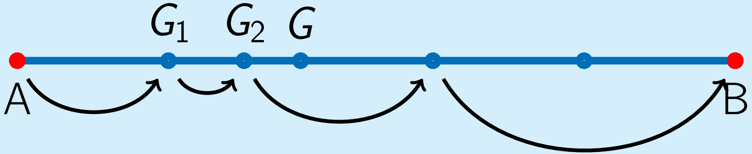
Proof



First case: G is closer than G_2

Refill at G instead of G_1

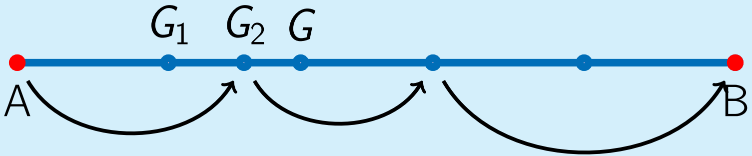
Proof



Second case: G_2 is closer than G

Avoid refill at G_1

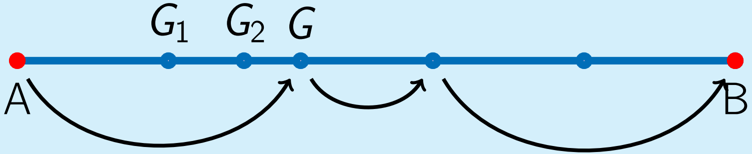
Proof



Second case: G_2 is closer than G

Avoid refill at G_1

Proof



Second case: G_2 is closer than G

Avoid refill at G_1

Proof

- Route R with the minimum number of refills
- G_1 — position of first refill in R
- G_2 — next stop in R (refill or B)
- G — farthest refill reachable from A
- If G is closer than G_2, refill at G instead of G_1
- Otherwise, avoid refill at G_1 $\square$

Outline

$$A = x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_n \leq x_{n+1} = B$$

MinRefills(x, n, L)

```
numRefills ← 0, currentRefill ← 0
while currentRefill ≤ n:
  lastRefill ← currentRefill
  while (currentRefill ≤ n and
         x[currentRefill + 1] − x[lastRefill] ≤ L):
    currentRefill ← currentRefill + 1
  if currentRefill == lastRefill:
    return IMPOSSIBLE
  if currentRefill ≤ n:
    numRefills ← numRefills + 1
return numRefills
```

Lemma

The running time of `MinRefills`(x, n, L) is $O(n)$.

Proof

- *currentRefill* changes from 0 to $n + 1$, one-by-one
- *numRefills* changes from 0 to at most n, one-by-one
- Thus, $O(n)$ iterations $\square$

Outline

Reduction to Subproblem

- Make a first move
- Then solve a problem of the same kind
- Smaller: fewer digits, fewer fuel stations
- This is called a "subproblem"

Safe move

- A move is called safe if there is an optimal solution consistent with this first move
- Not all first moves are safe
- Often greedy moves are not safe

General Strategy

$$\begin{tikzcd}[column sep=small] \text{Problem} \arrow[rr, "\text{greedy choice}"] & & \text{Safe move} \arrow[dl, bend left] \\ & \text{Subproblem} \arrow[ul, bend left] & \end{tikzcd}$$

- Make a greedy choice
- Prove that it is a safe move
- Reduce to a subproblem
- Solve the subproblem

Greedy Algorithms: Grouping Children

Michael Levin

Higher School of Economics

Algorithmic Toolbox
Data Structures and Algorithms

Outline

1. The Problem
2. Naive Algorithm
3. Efficient Algorithm

Many children came to a celebration. Organize them into the minimum possible number of groups such that the age of any two children in the same group differ by at most one year.

Outline

1. The Problem
2. Naive Algorithm
3. Efficient Algorithm

MinGroups(C)

```
m ← len(C)
for each partition into groups
C = G_1 ∪ G_2 ∪ ··· ∪ G_k:
  good ← true
  for i from 1 to k:
    if max(G_i) − min(G_i) > 1:
      good ← false
  if good:
    m ← min(m, k)
return m
```

Running time

Lemma

The number of operations in `MinGroups`(C) is at least 2^n, where n is the number of children in C.

Proof

- Consider just partitions in two groups
- $C = G_1 \cup G_2$
- For each $G_1 \subset C$, $G_2 = C \setminus G_1$
- Size of C is n
- Each item can be included or excluded from G_1
- There are 2^n different G_1
- Thus, at least 2^n operations $\square$

Asymptotics

- Naive algorithm works in time $\Omega(2^n)$
- For $n = 50$ it is at least

$$2^{50} = 1125899906842624$$

  operations!
- We will improve this significantly

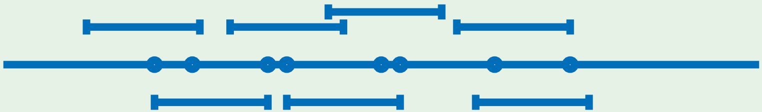
Outline

Covering points by segments

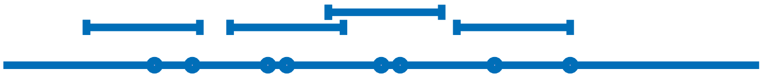
Input: A set of n points $x_1, \ldots, x_n \in \mathbb{R}$.

Output: The minimum number of segments of unit length needed to cover all the points.

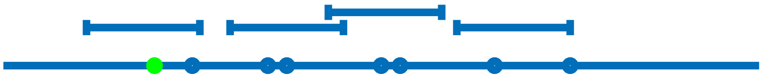
Example



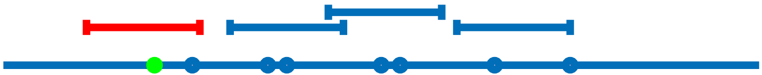
Safe move: cover the leftmost point with a unit segment which starts in this point.



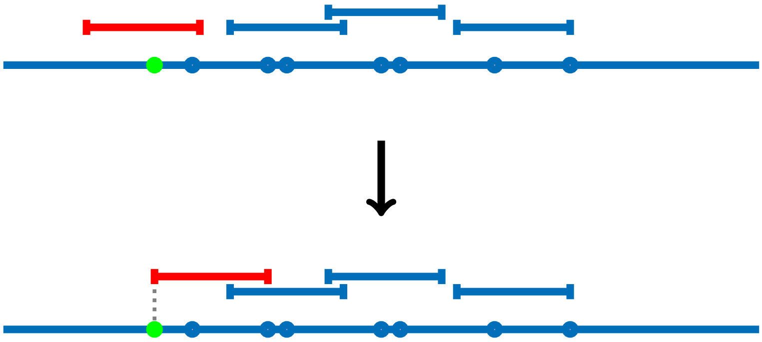
Safe move: cover the leftmost point with a unit segment which starts in this point.



Safe move: cover the leftmost point with a unit segment which starts in this point.



Safe move: cover the leftmost point with a unit segment which starts in this point.



Assume $x_1 \le x_2 \le \ldots \le x_n$

PointsCoverSorted($x_1, \ldots, x_n$)

$R \leftarrow \{\},\ i \leftarrow 1$

while $i \le n$:

  $[\ell, r] \leftarrow [x_i, x_i + 1]$

  $R \leftarrow R \bigcup \{[\ell, r]\}$

  $i \leftarrow i + 1$

  while $i \le n$ and $x_i \le r$:

    $i \leftarrow i + 1$

return R

Lemma

The running time of `PointsCoverSorted` is $O(n)$.

Proof

- i changes from 1 to n
- For each i, at most 1 new segment
- Overall, running time is $O(n)$ $\square$

Total Running Time

- `PointsCoverSorted` works in $O(n)$ time
- Sort $\{x_1, x_2, \ldots, x_n\}$, then call `PointsCoverSorted`
- Soon you'll learn to sort in $O(n \log n)$
- Sort + `PointsCoverSorted` is $O(n \log n)$

Asymptotics

- Straightforward solution is $\Omega(2^n)$
- Very long for $n = 50$
- Sort + greedy is $O(n \log n)$
- Fast for $n = 10\,000\,000$
- Huge improvement!

Conclusion

- Straightforward solution is exponential
- Important to reformulate the problem in mathematical terms
- Safe move is to cover leftmost point
- Sort in $O(n \log n)$ + greedy in $O(n)$

Greedy Algorithms: Fractional Knapsack

Michael Levin

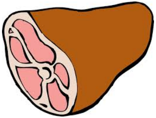
Higher School of Economics

Algorithmic Toolbox
Data Structures and Algorithms

Outline

1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Long Hike



Outline

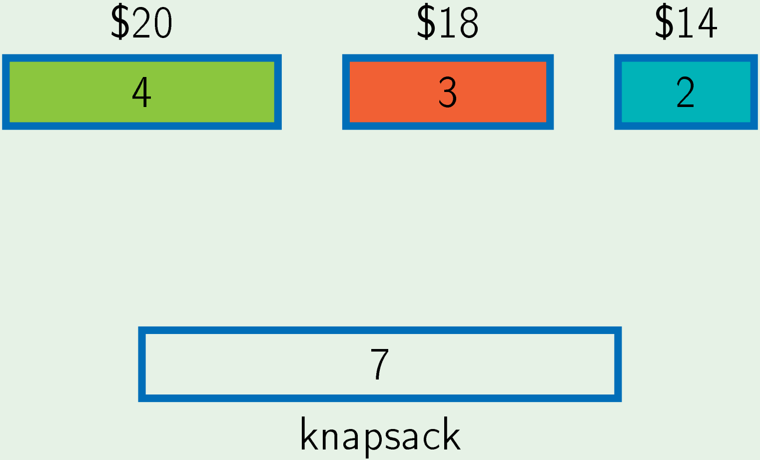
1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Fractional knapsack

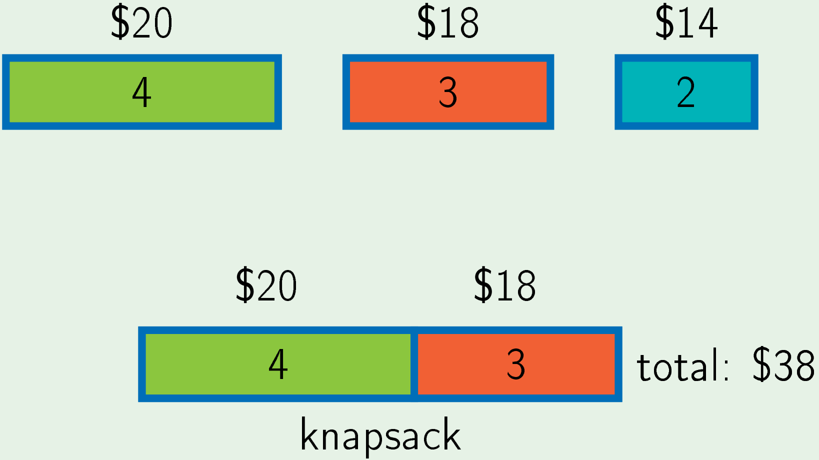
Input: Weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ of n items; capacity W.

Output: The maximum total value of fractions of items that fit into a bag of capacity W.

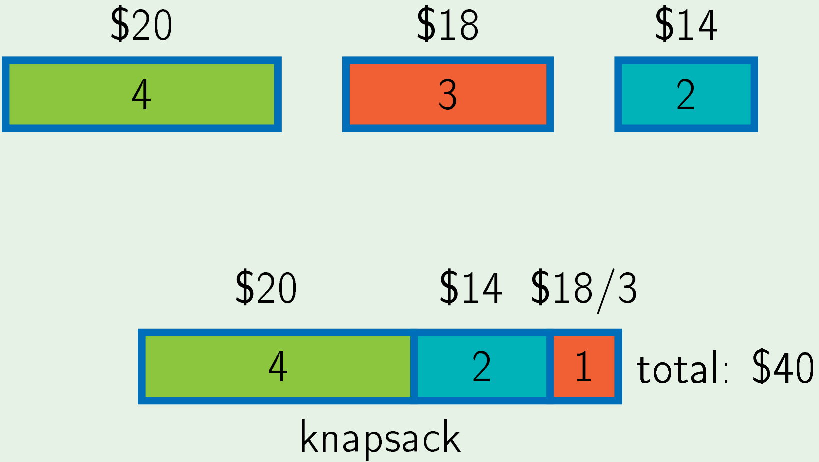
Example



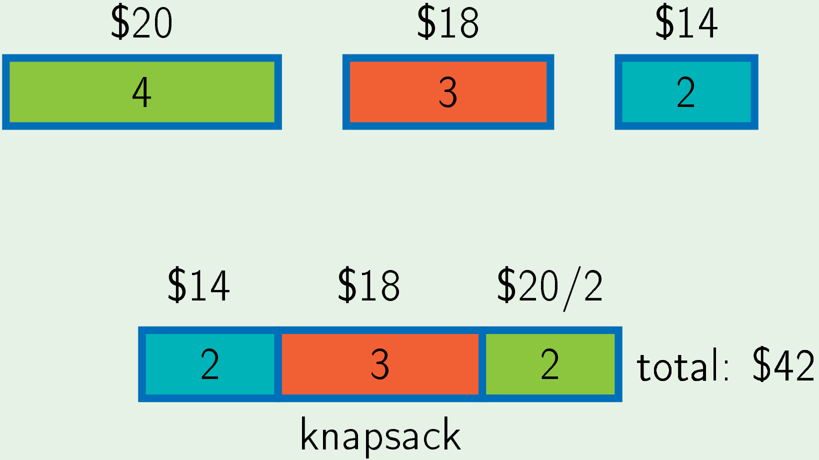
Example



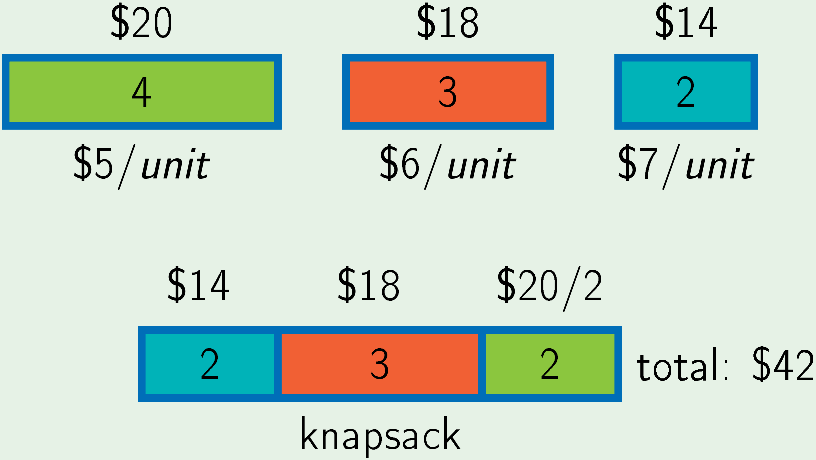
Example



Example



Example

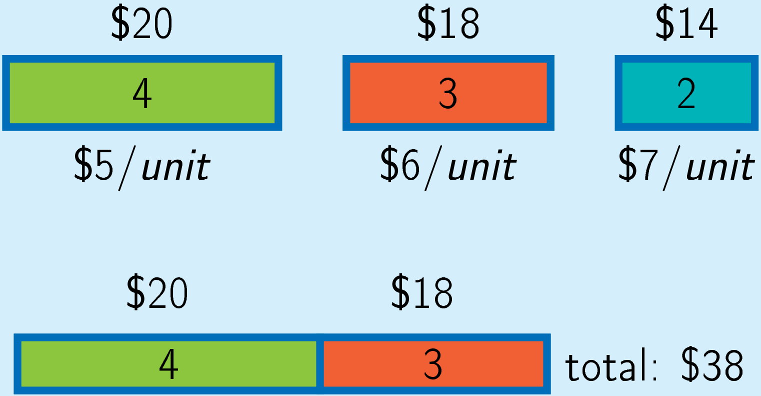


Safe move

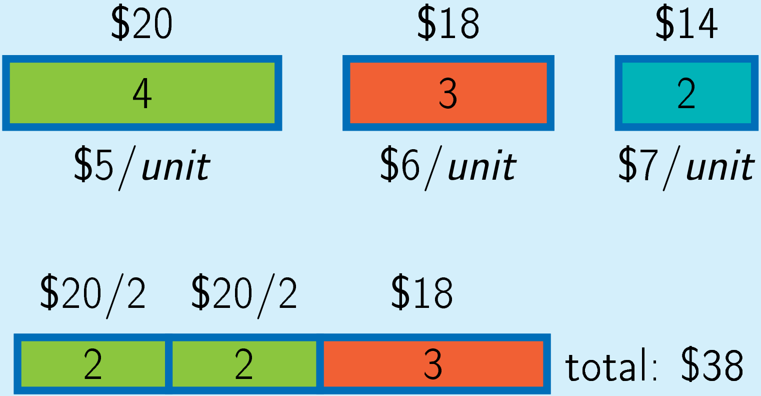
Lemma

There exists an optimal solution that uses as much as possible of an item with the maximal value per unit of weight.

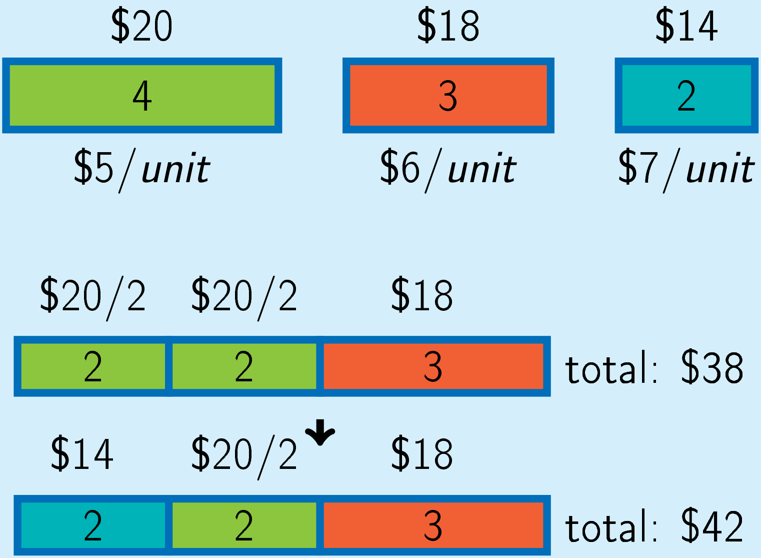
Proof



Proof



Proof


$20
4
$5/unit
$18
3
$6/unit
$14
2
$7/unit
$20/2
$20/2
$18
2
2
3
total: $38
$14
$20/2
$18
2
2
3
total: $42

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

Outline

1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

Knapsack($W, w_1, v_1, \ldots, w_n, v_n$)

```
A ← [0, 0, ..., 0], V ← 0
repeat n times:
    if W = 0:
        return (V, A)
    select i with w_i > 0 and max v_i/w_i
    a ← min(w_i, W)
    V ← V + a v_i/w_i
    w_i ← w_i − a, A[i] ← A[i] + a, W ← W − a
return (V, A)
```

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- Select best item on each step is $O(n)$
- Main loop is executed n times
- Overall, $O(n^2)$ $\square$

Optimization

- It is possible to improve asymptotics!
- First, sort items by decreasing $\frac{v}{w}$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

```
Knapsack(W, w_1, v_1, ..., w_n, v_n)
```

$A \leftarrow [0, 0, \ldots, 0]$, $V \leftarrow 0$

for i from 1 to n:

  if $W = 0$:

    return (V, A)

  $a \leftarrow \min(w_i, W)$

  $V \leftarrow V + a\frac{v_i}{w_i}$

  $w_i \leftarrow w_i - a$, $A[i] \leftarrow A[i] + a$, $W \leftarrow W - a$

return (V, A)

Asymptotics

- Now each iteration is $O(1)$
- `Knapsack` after sorting is $O(n)$
- Sort + `Knapsack` is $O(n \log n)$

Greedy Algorithms: Review

Michael Levin

Higher School of Economics

Algorithmic Toolbox
Data Structures and Algorithms

Main Ingredients

- Safe move
- Prove safety
- Solve subproblem
- Estimate running time

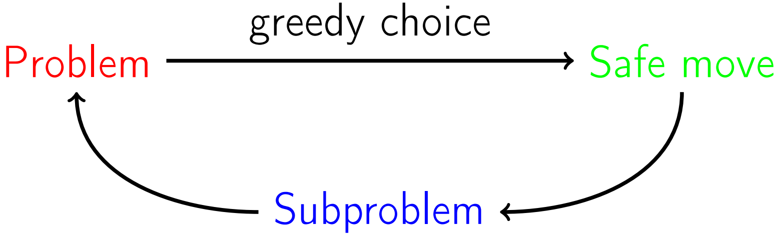
Safe Moves

- Put max digit first
- Refill at the farthest reachable gas station
- Cover leftmost point
- Use item with maximum value per unit of weight

Optimization

- Assume everything is somehow sorted
- Which sort order is convenient?
- Greedy move can be faster after sorting

General Strategy



- Make a greedy choice
- Prove that it is a safe move
- Reduce to a subproblem
- Solve the subproblem