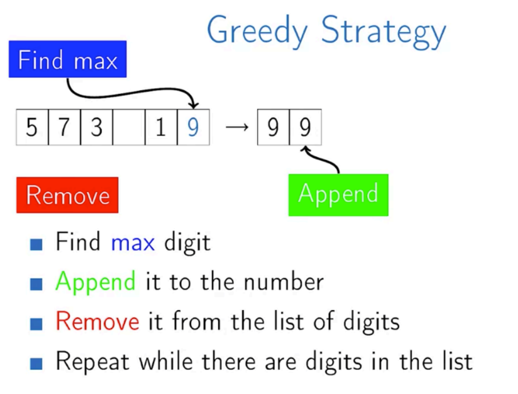
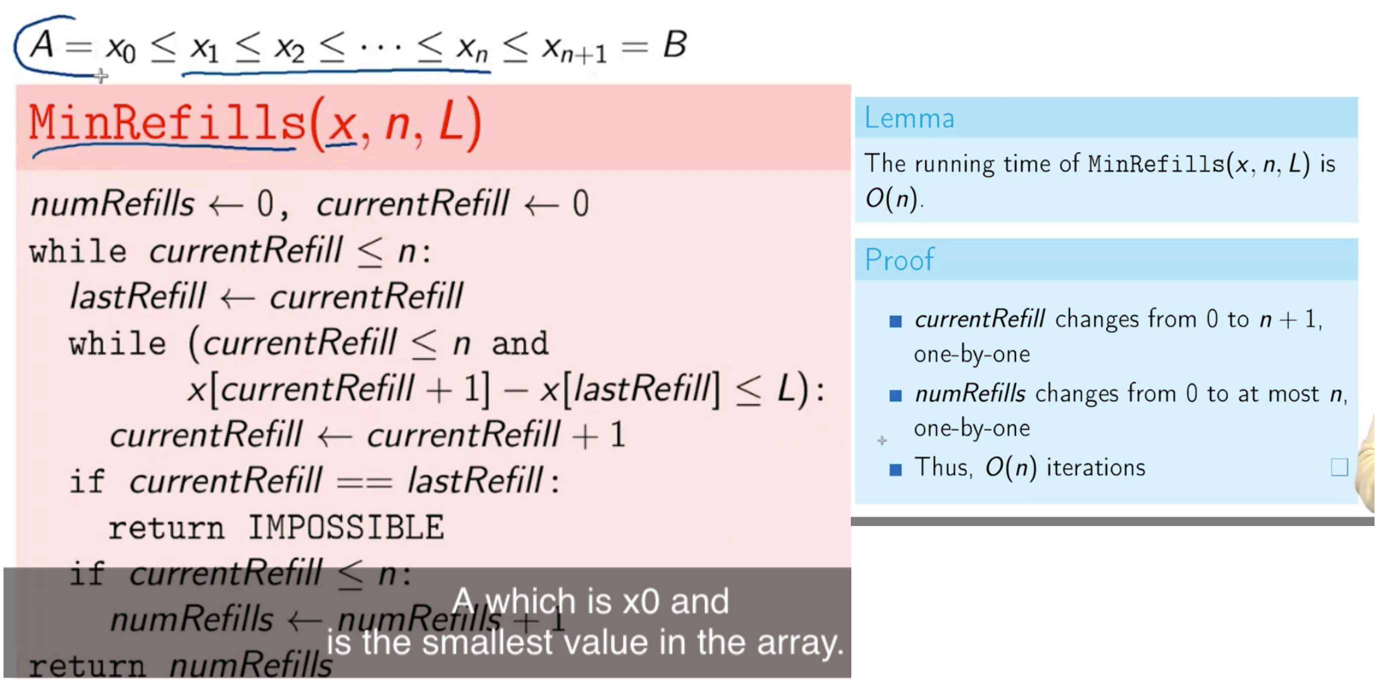
# Sandiego Univ “Algorithm Toolbox”
## Week 3. Greedy Algorithm

### 1. Largest number

---




### Carfueling. MinRefill




최대 리필 횟수는 기껏해야 n 번// 외부 loop, 내부 loop 구분

## 2. Subproblems in Greedy Algorithms

### 2.1 Safe Move

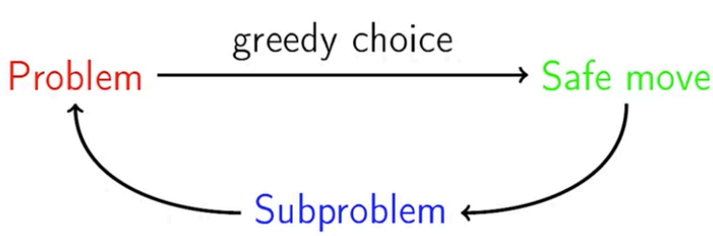
Safe move

- 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

-

### - 2.2 General Strategy

General Strategy




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

재생

## 3. Grouping Children

### 3.1 Naive

**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 $G_1$

So, there can be 2 to

It's too slow and inefficient.

### 3.2 Efficient method

Treat points instead of the above children.

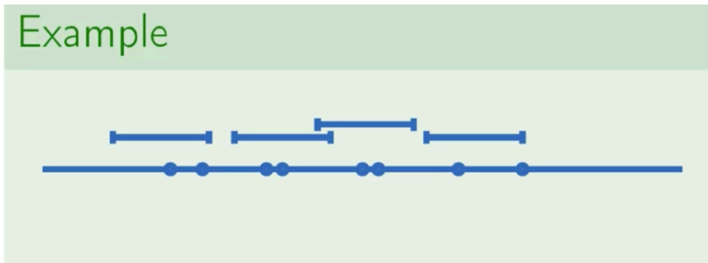
**Covering points by segments**

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

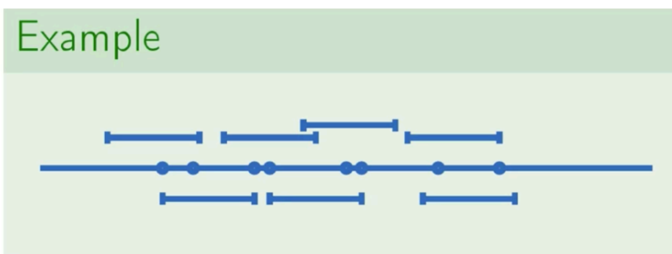
Output: The minimum number of segments of unit length needed to cover all the points.

There is points on a line.

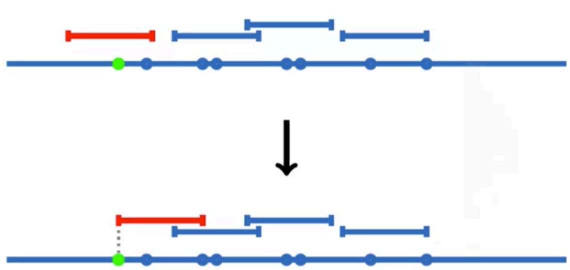
Many lines with same length over the line means covering.
But this is not optimal solution.




See 3 lines below the centered line. These lines mean covering, and this arrangement is optimal solution for there are the least number of the covering required.




Safe move: cover the leftmost point with a unit segment with left end in this point.



### 3.2.1 Algorithm

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

**PointsCoverSorted**$(x_1, \ldots, x_n)$

$R \leftarrow \{\}, \; i \leftarrow 1$
while $i \leq n$:
  $[\ell, r] \leftarrow [x_i, x_i + 1]$
  $R \leftarrow R \bigcup \{[\ell, r]\}$
  $i \leftarrow i + 1$
  while $i \leq n$ and $x_i \leq r$:
    $i \leftarrow i + 1$
return $R$

We'll start with an empty set
of segments denoted by R and

R: empty set
i =1
While (i < = n): # Currently, 'i' refers to leftmost point
l, r = xi, xi+1 # xi point is in left end. (2) [l,r]: [left, right] in a specific segment
R = R U l,r
i = i+1 # Not remove the point, but just move the pointer

**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 $O(n)$ $\square$

# 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 $O(n)$

## 4. Fractional Knapsack

### 4.1 Long Hike

>> It requires enough foods.
What is the maximized amount of calories that I can get from these foods?

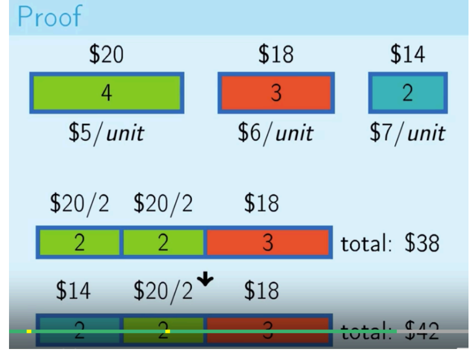
### 4.2 Fractional Knapsack

**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 $W$.

**Proof**




---

# 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.

# 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

## 4.3 Implementation, Analysis, and Optimization

### 4.3.1 Naive

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

$A \leftarrow [0, 0, \ldots, 0]$, $V \leftarrow 0$
repeat $n$ times:
  if $W = 0$:
    return $(V, A)$
  select $i$ with $w_i > 0$ and max $\frac{v_i}{w_i}$
  $a \leftarrow \min(w_i, W)$
  $V \leftarrow V + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$, $A[i] \leftarrow A[i] + a$, $W \leftarrow W - a$
return $(V, A)$

**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, $O(n^2)$ □

For _ in range(n):

If W == 0:
Return (V, A)

### 4.3.2 Efficient ver

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

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

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

## 5. Review

### Main Ingredients

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

### Safe Moves

- Put max digit first
- Find first occurrence of first character
- Cover leftmost point
- Use item with maximum value per unit of weight