Lecture 6: Backtracking

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

Homework 1 should have been turned in last night before midnight

- If you did not ask for any extension, none will be granted

Homework 2 is released

- Due **Next Tuesday 5/19 at 11:59PM Boston Time**
- First 3 problems can be worked out after this lecture
- Problem 4 will be solvable after tomorrow

I owe a couple of people emails from this morning, will do so tonight

Slides (including reading assignment) are on the course website

- If they are not, I will stop and fix this

General point on homework

Do not wait until the last minute to read the questions

If you are struggling, ask questions early!

- Rule of thumb: If you spend more than 30 minutes on a problem and make little or no progress, ask a question on Piazza
- If you can’t ask a question without giving away part of the solution, ask privately to the instructors on Piazza
- If you don’t know how to start, ask a private question **where you give some thoughts on how you could maybe approach the problem**
  - We can’t help if you just say “I don’t get it”, we need somewhere to start!

There is a LaTeX tag on Piazza. Ask questions if you are having problems with LaTeX.

Today

Backtracking

- N Queens
- SubsetSum
- Text Segmentation

Backtracking

- So far, we have seen cases where the next recursive call is clear
  - In MergeSort, we need both left and right subarrays to be sorted
  - In MOMSelect and BinarySearch, we guarantee the value we are looking for is in a specific subarray
- What if we can't tell from the start which decision to make?
- Enter *backtracking*: When we are not sure what to do, try one small step in both directions and evaluate all outcomes.

N Queens

Problem statement: Given an *n*x*n* dimensional chessboard, place *n* queens on the board such that none can attack each other.

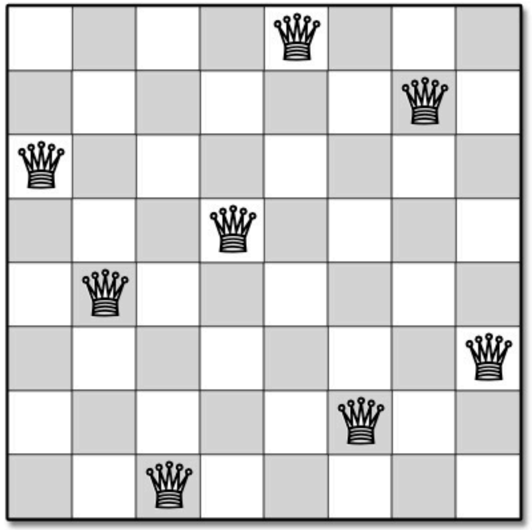


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array $[5, 7, 1, 4, 2, 8, 6, 3]$

N Queens

Problem statement: Given an *n*x*n* dimensional chessboard, place *n* queens on the board such that none can attack each other.

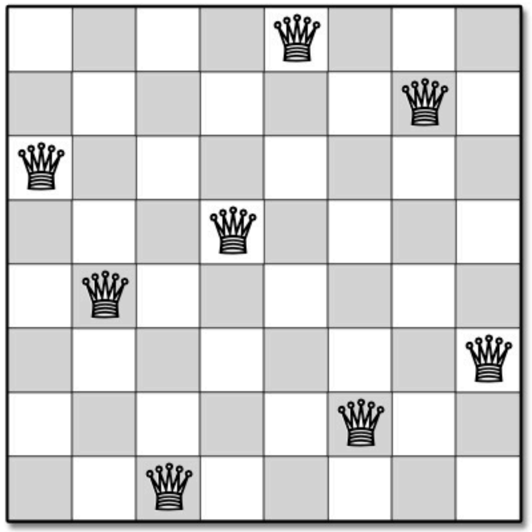


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array $[5, 7, 1, 4, 2, 8, 6, 3]$

Given an arbitrary *n*, how can we decide where to place queens?

N Queens

Problem statement: Given an *n*x*n* dimensional chessboard, place *n* queens on the board such that none can attack each other.

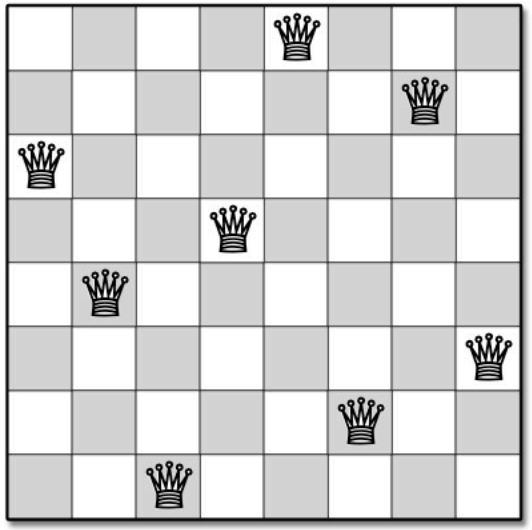


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array $[5, 7, 1, 4, 2, 8, 6, 3]$

Idea: Incrementally build a solution by placing one queen at a time!

Given an arbitrary *n*, how can we decide where to place queens?

N Queens

Idea: Incrementally build a solution by placing one queen at a time!

```
PlaceQueens(Q[1..n], r):
  If r = n+1:
    print Q[1..n]
  Else:
    for j ← 1 to n:
      legal ← True
      for i ← 1 to r − 1:
        if(Q[i]=j) or
          (Q[i]=j+r-i) or
          (Q[i] = j − r):
            legal ← False
      if legal:
        Q[r] ← j
        PlaceQueens(Q[1..n], r+1)
```

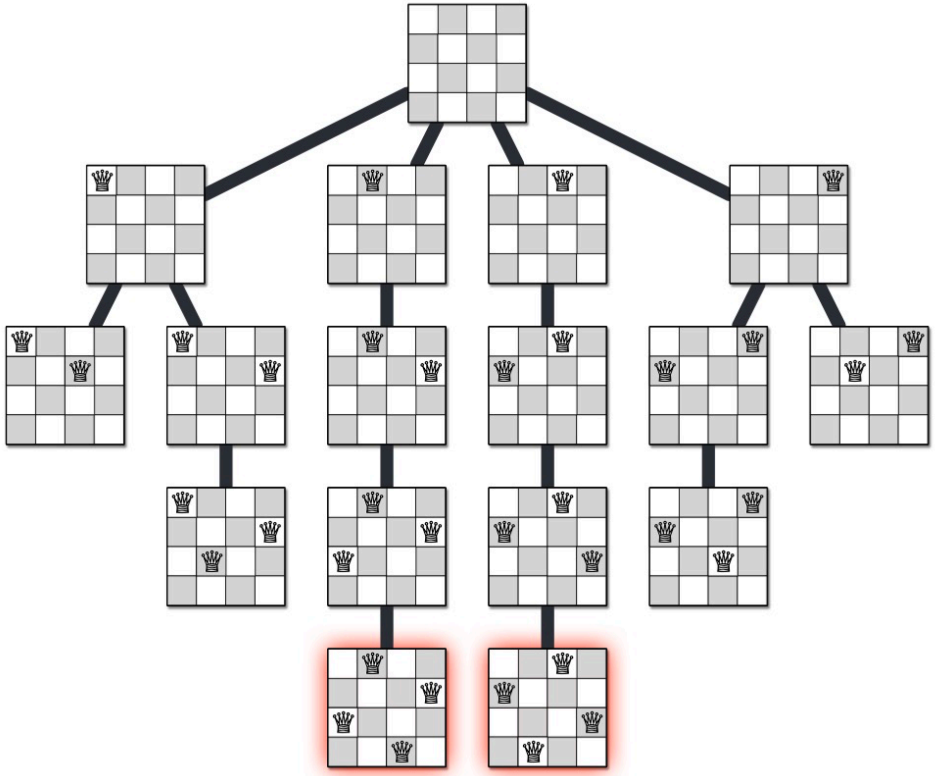


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

N Queens

Idea: Incrementally build a solution by placing one queen at a time!

```
PlaceQueens(Q[1..n], r):
  If r = n+1:
    print Q[1..n]
  Else:
    for j ← 1 to n:
      legal ← True
      for i ← 1 to r − 1:
        if(Q[i]=j) or
          (Q[i]=j+r-i) or
          (Q[i] = j − r):
            legal ← False
      if legal:
        Q[r] ← j
        PlaceQueens(Q[1..n], r+1)
```

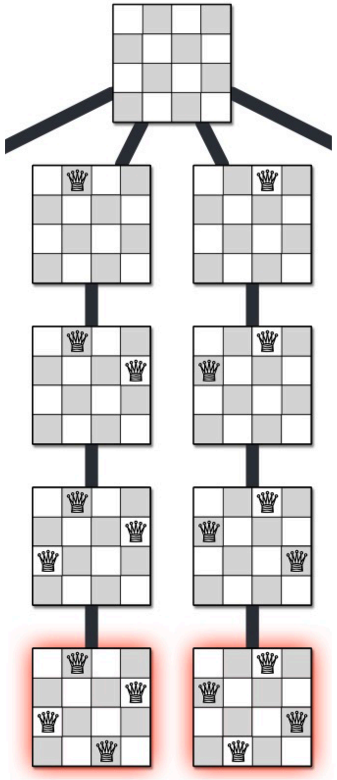


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

N Queens Wrap And Backtracking pattern

Idea: Incrementally build a solution by placing one queen at a time!

- Appropriate when a sequence of incremental decisions can enumerate solutions
  - Solution is often itself a sequence, e.g. Q[1..n] is a sequence of queens placed in rows 1..n
- Exactly 1 decision is made at every step
  - We usually need some information about previous decisions, but this should be as small as possible
- Problem is solved by *recursive brute force*, meaning we do not "prune" decisions that are obviously bad (leaves in the tree)

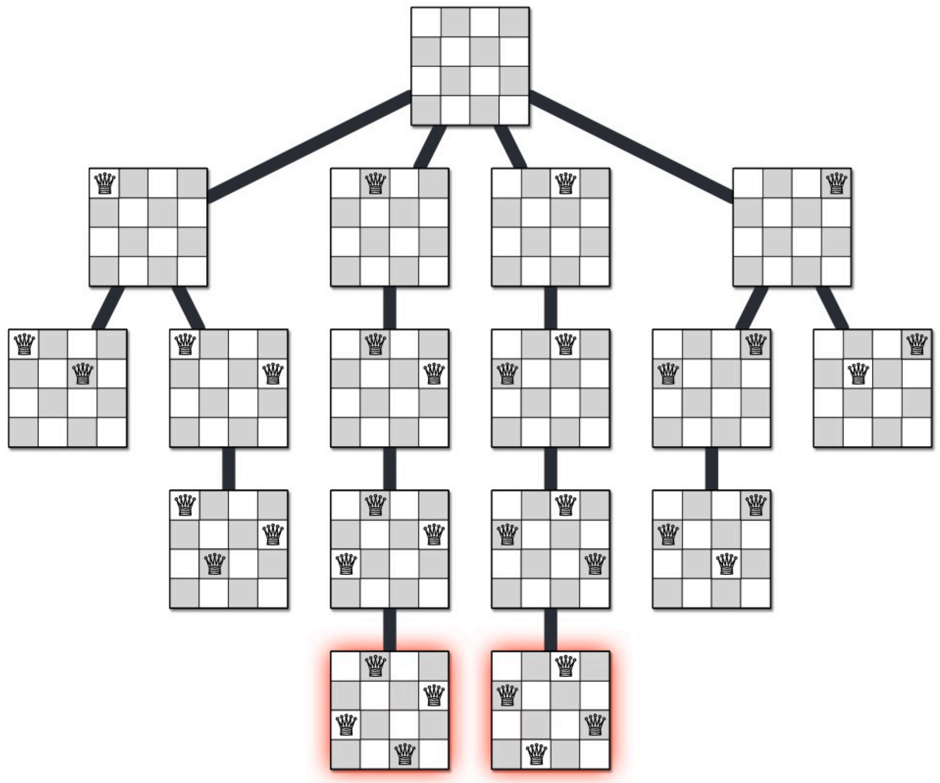


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

Subset Sum

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \ldots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Subset Sum

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \dots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

Subset Sum

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \ldots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

$X = \{8,6,7,5,3,10,9\}, T = 15$

Subset Sum

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \ldots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

$$X = \{11,6,5,1,7,13,12\}, T = 15$$

Subset Sum Solution and Example

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \dots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$$X = [1,2,3], T = 3$$

SubsetSum$(X, i = 3, T = 3)$

Subset Sum Example

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \dots, x_n\}$ and a target integer value T. We want to find a subset $\mathrm{Y} \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$$X = [1,2,3], T = 3$$

$$\begin{tikzcd}[column sep=small] & \texttt{SubsetSum}(X, i = 3, T = 3) \arrow[dl, "\text{with}"'] \arrow[dr, "\text{without}"] & \\ (X, 2, 3 - 3 = 0) & & (X, 2, 3) \end{tikzcd}$$

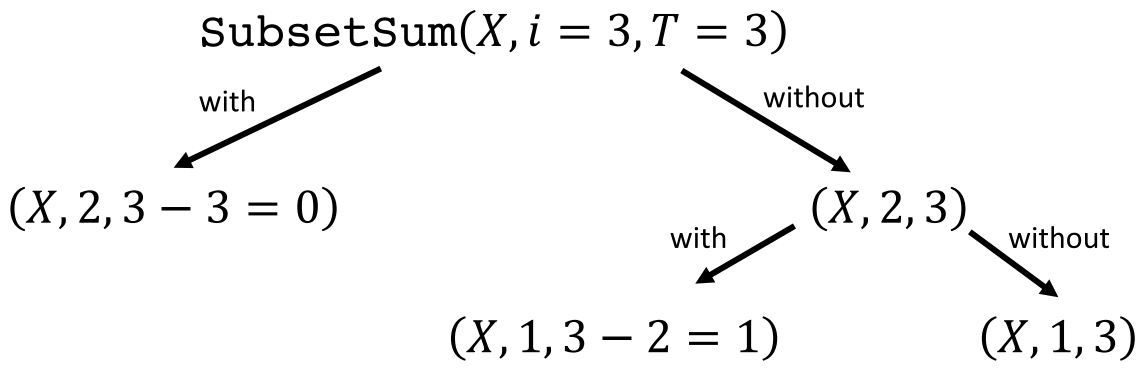
Subset Sum Example

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \ldots, x_n\}$ and a target integer value T. We want to find a subset $Y \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$X = [1,2,3], T = 3$



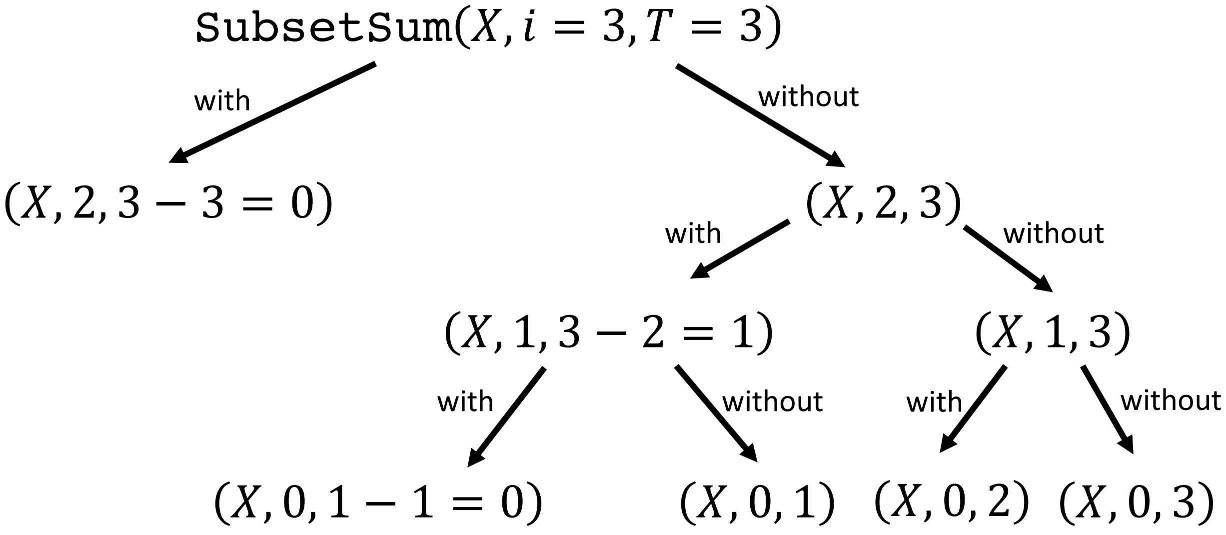
Subset Sum Example

We are given a set of n positive integers $X = \{x_{1,} x_{2,} \ldots, x_n\}$ and a target integer value T. We want to find a subset $Y \subseteq X$ such that the sum of the elements $\sum_{x_i \in Y} x_i = T$.

Our problem: For a given T and X, does such a Y exist?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$$X = [1,2,3], T = 3$$



Subset Sum Correctness

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

Trivially works for base cases:

- T = 0 → Always true (empty subset)
- T < 0 → Always false (our integers are > 0)
- n = 0 (X is empty)→ Always false (no subset can add to any T)

Otherwise, if there is a subset that sums to T, it either contains X[i] or it doesn’t. Both of these possibilities are evaluated by the recursion fairy.

Subset Sum Running Time

Recurrence Relation?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

Subset Sum Running Time

Recurrence Relation?

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$$T(n) = 2T(n-1) + O(1) \leq O(2^n)$$

Subset Sum Running Time

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

Recurrence Relation?

$$T(n) = 2T(n-1) + O(1) \leq O(2^n)$$

(You can show with a recursion tree)

Subset Sum Wrap

```
SubsetSum(X[1..n], i, T):
  If T = 0:
    return True
  ElseIf T < 0 or i = 0:
    return False
  Else:
    with ← SubsetSum(X, i-1, T – X[i])
    wout ← SubsetSum(X, i-1, T)
    return with OR wout
```

$$T(n) = 2T(n-1) + O(1) \leq O(2^n)$$

- Our algorithm tells us whether such a subset exists, but does not return the subset
  - Relatively straightforward modifications to return the subset
- Our algorithm is not scalable
  - We will see later this week how to use *dynamic programming* to speed it up by solving subproblems in a smart order and storing the solutions for reuse

Text Segmentation

Problem: Given an array $A[1..n]$ representing a sequence of n characters without spaces, determine whether the array can be subdivided into a sequence of *words.*

Text Segmentation

Problem: Given an array $A[1..n]$ representing a sequence of n characters without spaces, determine whether the array can be subdivided into a sequence of *words.*

Assume we are given a function $IsWord(i, j)$. This function assumes A is a global variable and returns True if the subarray $A[i..j]$ is a word in the language of the sequence.

- This allows us to avoid passing subarrays as arguments to functions.

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

he

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

brow

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

row

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

own

The sentence: "the brown fox is quick"

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

brown

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

brown

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Recall the pattern from earlier:

- Sequence of decisions made 1 at a time
    - “Does $A[i..\mathrm{j}]$ belong in my sequence of words?”

Text Segmentation Example

A	t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

brown

The sentence: “the brown fox is quick”

Where are there potential problems for $IsWord(i, j)$?

Recall the pattern from earlier:

- Sequence of decisions made 1 at a time
    - “Does $A[i..\mathrm{j}]$ belong in my sequence of words?”
- Recursive brute force
    - Check every possible word, even if there might be a way to prune!

Text Segmentation Solution

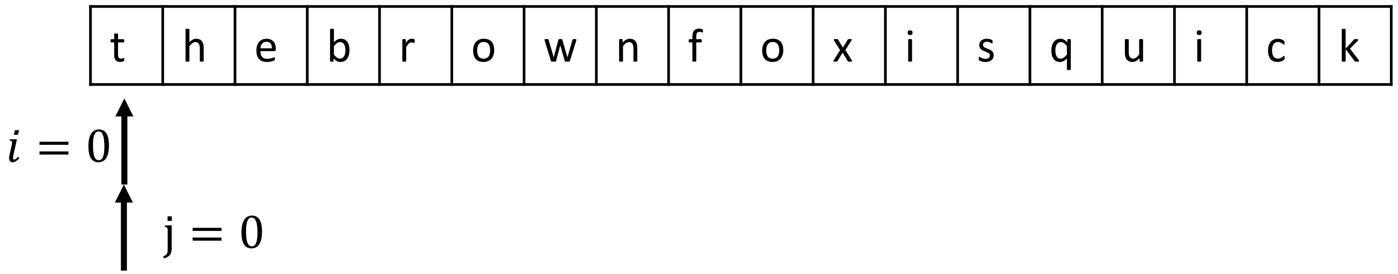
t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$

$Splittable(A[1..n], i)$:
  If $i > n$:
    return True
  Else:
    $j \leftarrow i$
    for i to n:
      If $IsWord(i, j)$:
        If $Splittable(A[1..n], j + 1)$:
          return True

  return False

Text Segmentation Solution




```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Text Segmentation Solution

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ $\quad$ $\text{j} = 1$

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Text Segmentation Solution

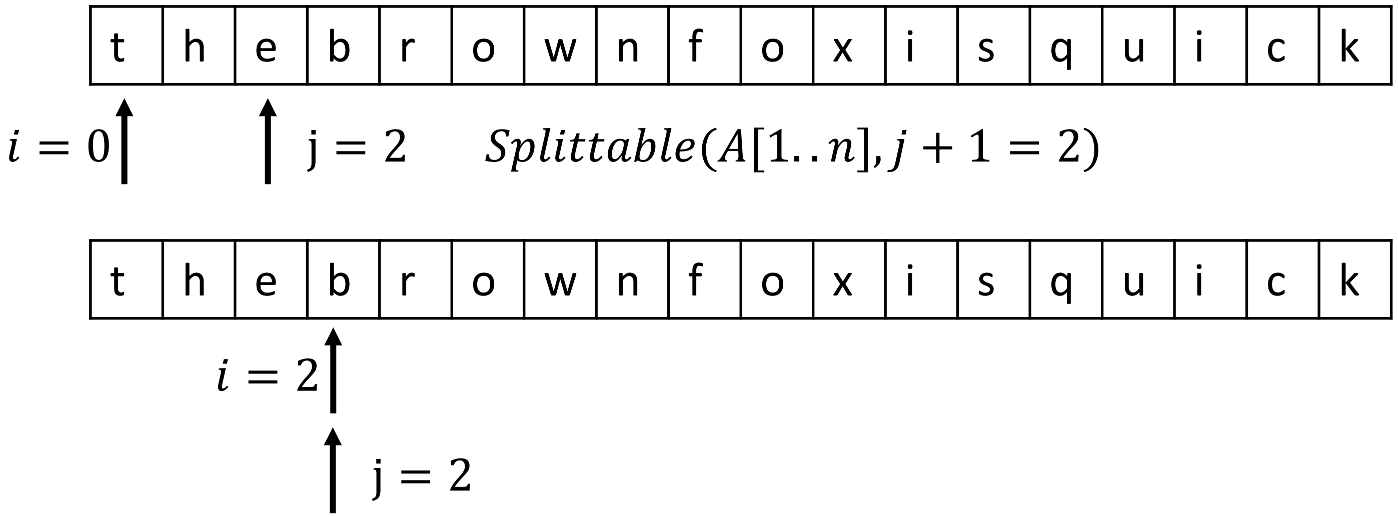
t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j + 1 = 2)$

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```



Text Segmentation Solution

$Splittable(A[1..n], i)$:
  If $i > n$:
    return True
  Else:
    $j \leftarrow i$
    for i to n:
      If $IsWord(i, j)$:
        If $Splittable(A[1..n], j + 1)$:
          return True

  return False

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

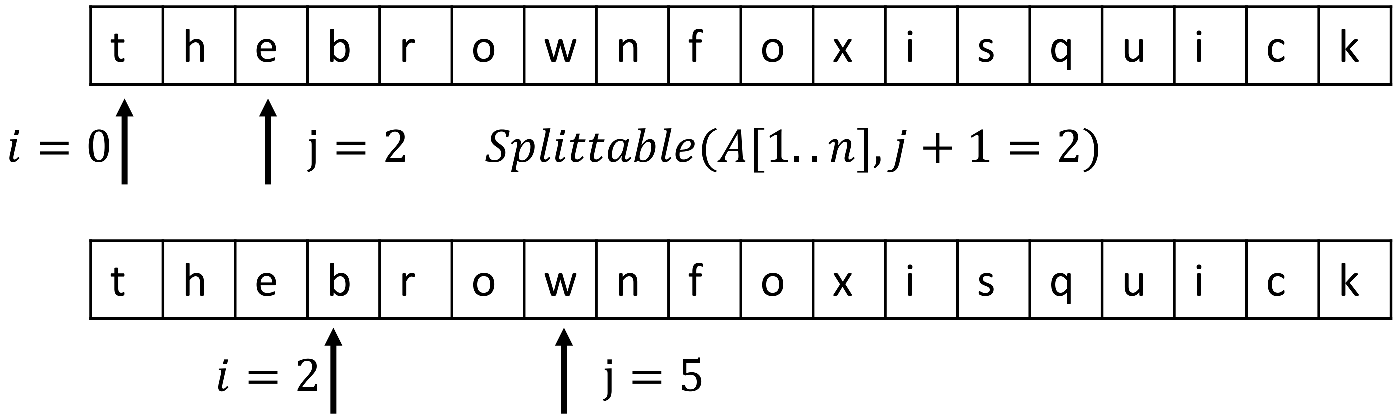
$i = 0$ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 3

Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```



Text Segmentation Solution

$Splittable(A[1..n], i)$:
  If $i > n$:
    return True
  Else:
    $j \leftarrow i$
    for i to n:
      If $IsWord(i, j)$:
        If $Splittable(A[1..n], j + 1)$:
          return True

  return False

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ ↑ ↑ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ ↑ ↑ j = 5 $Splittable(A[1..n], j + 1 = 6)$

What is going to happen?

Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ ↑ ↑ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ ↑ ↑ j = 5 $Splittable(A[1..n], j + 1 = 6)$

What is going to happen?

“nfoxisquick” is not a word, so:

- i never becomes greater than n

Text Segmentation Solution

$Splittable(A[1..n], i)$:
  If $i > n$:
    return True
  Else:
    $j \leftarrow i$
    for i to n:
      If $IsWord(i, j)$:
        If $Splittable(A[1..n], j+1)$:
          return True

  return False

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j+1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 5 $Splittable(A[1..n], j+1 = 6)$

What is going to happen?

“nfoxisquick” is not a word, so:
- i never becomes greater than n
- $IsWord(i, j)$ is never true

Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 5 $Splittable(A[1..n], j + 1 = 6)$

What is going to happen?

“nfoxisquick” is not a word, so:

- i never becomes greater than n
- $IsWord(i, j)$ is never true
- $Splittable(A[1..n], j + 1 = 6)$ returns `False`

Text Segmentation Solution

$Splittable(A[1..n], i)$:
  If $i > n$:
    return True
  Else:
    $j \leftarrow i$
    for i to n:
      If $IsWord(i, j)$:
        If $Splittable(A[1..n], j + 1)$:
          return True

  return False

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ ↑ ↑ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ ↑ ↑ j = 5 $Splittable(A[1..n], j + 1 = 6)$

What is going to happen?

"nfoxisquick" is not a word, so:

- i never becomes greater than n
- $IsWord(i, j)$ is never true
- $Splittable(A[1..n], j + 1 = 6)$ returns `False`
- We go back to the loop!

Text Segmentation Solution

$Splittable(A[1..n], i)$:
  `If` $i > n$:
    `return True`
  `Else:`
    $j \leftarrow i$
    `for` i `to` n:
      `If` $IsWord(i, j)$:
        `If` $Splittable(A[1..n], j+1)$:
          `return True`

  `return False`

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j+1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 6

What is going to happen?

"nfoxisquick" is not a word, so:

- i never becomes greater than n
- $IsWord(i, j)$ is never true
- $Splittable(A[1..n], j+1 = 6)$ returns `False`
- We go back to the loop!

Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 6 $Splittable(A[1..n], j + 1 = 7)$

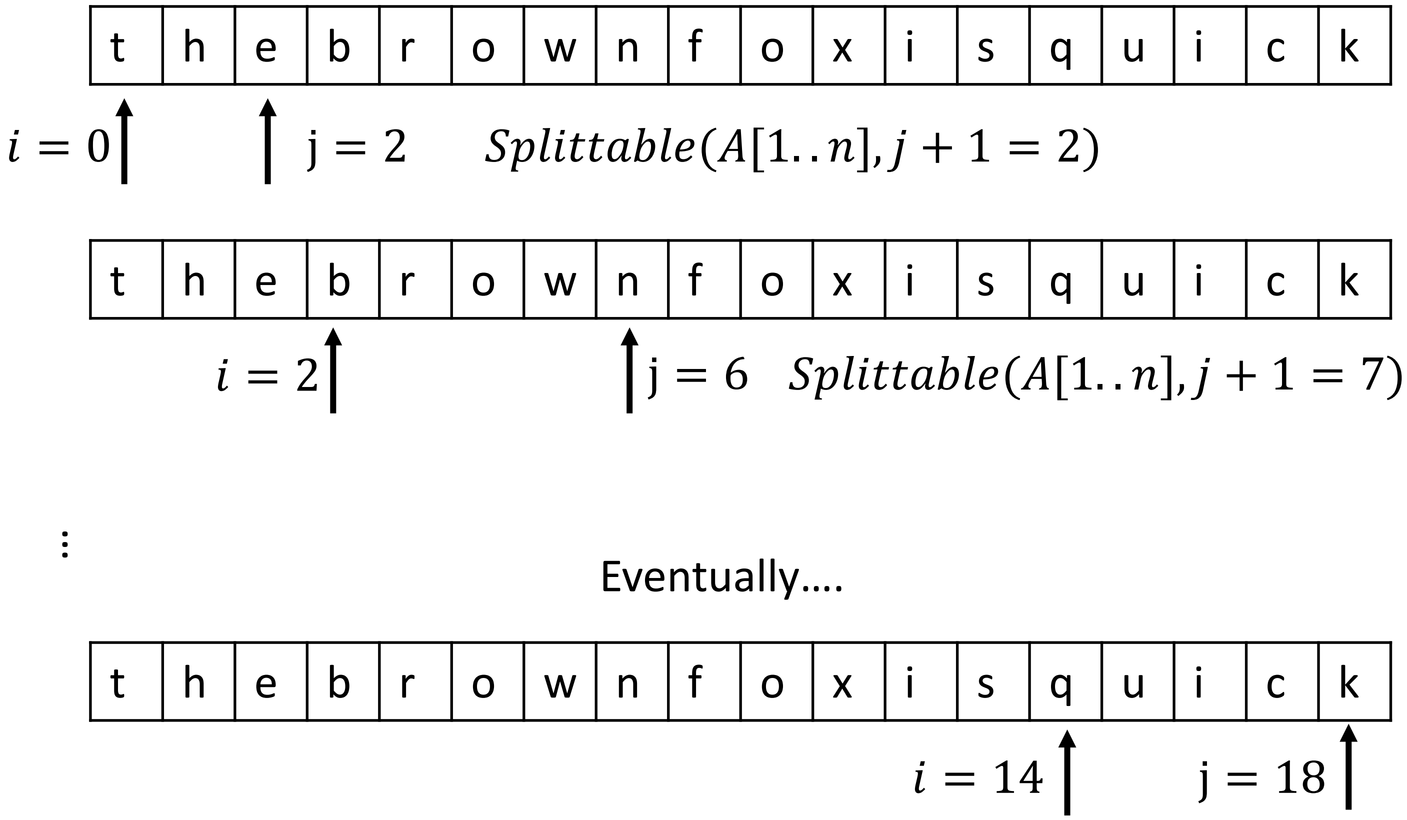
What is going to happen?

"nfoxisquick" is not a word, so:

- i never becomes greater than n
- $IsWord(i, j)$ is never true
- $Splittable(A[1..n], j + 1 = 6)$ returns `False`
- We go back to the loop!

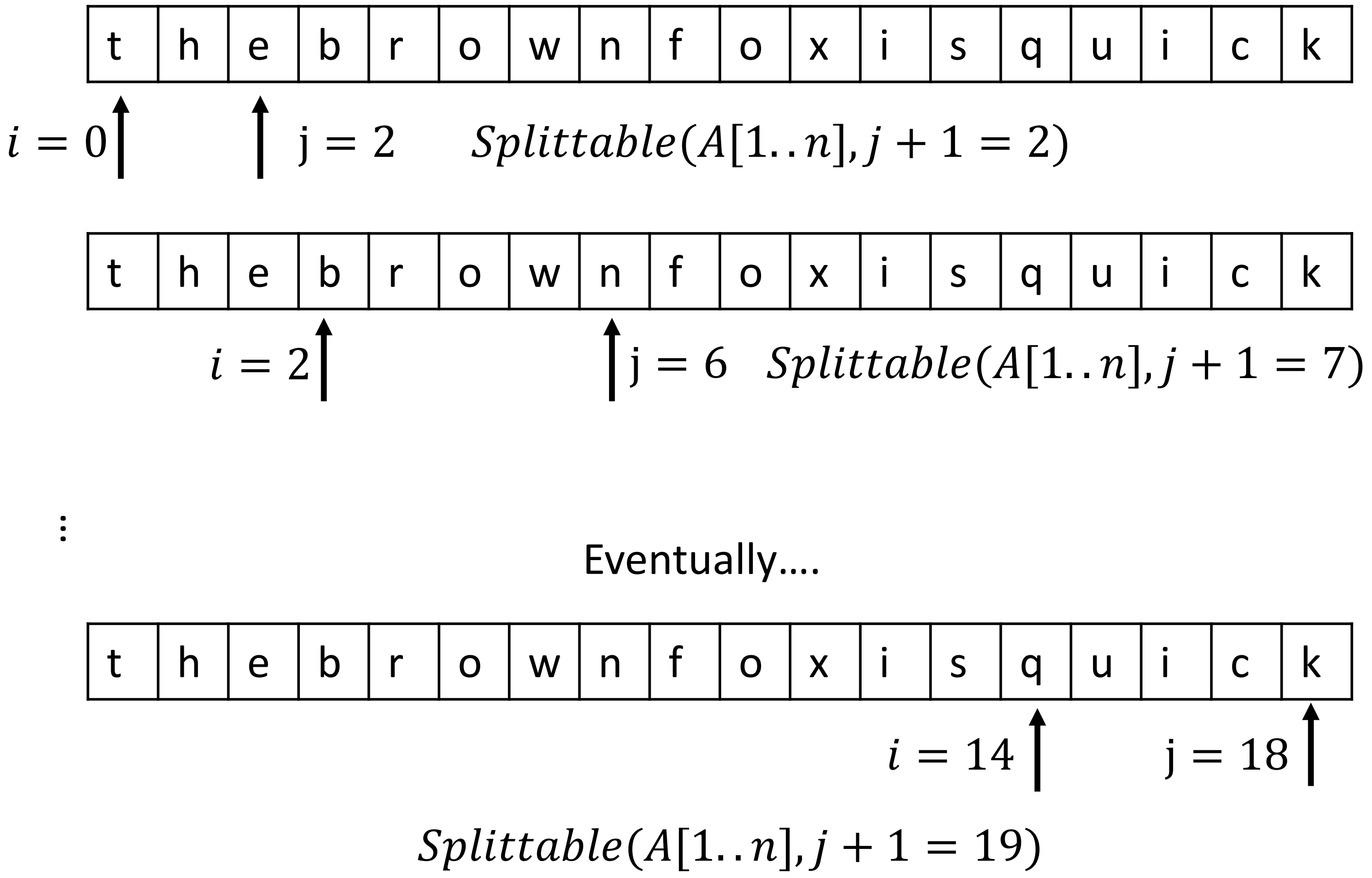
Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```



Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```



Text Segmentation Solution

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 0$ j = 2 $Splittable(A[1..n], j + 1 = 2)$

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 2$ j = 6 $Splittable(A[1..n], j + 1 = 7)$

⋮

Eventually….

t	h	e	b	r	o	w	n	f	o	x	i	s	q	u	i	c	k

$i = 14$ j = 18

$Splittable(A[1..n], j + 1 = 19)$

$19 > n$, returns `True`!

Text Segmentation Correctness

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Base case: $n = 1$. Either:

- $IsWord(1,1)$ returns False, in which case the loop ends and the algorithm returns False, or
- $IsWord(1,1)$ returns True, in which case $Splittable(A[1..n], 2)$ returns True

Text Segmentation Correctness

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Base case: $n = 1$. Either:

- $IsWord(1,1)$ returns False, in which case the loop ends and the algorithm returns False, or
- $IsWord(1,1)$ returns True, in which case $Splittable(A[1..n], 2)$ returns True

Assuming $IsWord(i, j)$ is correct and $Splittable(A[1..k], 1)$ is correct for $1 \le k \le n$, we immediately see that it must be correct for $Splittable(A[1..n+1], 1)$, since this runs the algorithm on inputs of maximum size $n - 1 < n$.

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

Calling $IsWord$ for all n items

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

Worst case: calling $Splittable$ on every subsequence

Calling $IsWord$ for all n items

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i,j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i,j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + cn$$

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + cn$$

$$T(n-1) \leq \sum_{i=0}^{n-1} T(i) + c(n-1)$$

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + cn$$

$$T(n-1) \leq \sum_{i=0}^{n-2} T(i) + c(n-1)$$

$$T(n) - T(n-1) \leq T(n-1) + c$$

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i,j):
        If Splittable(A[1..n], j+1):
          return True

  return False
```

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + cn$$

$$T(n-1) \leq \sum_{i=0}^{n-1} T(i) + c(n-1)$$

$$T(n) - T(n-1) \leq T(n-1) + c$$

$$T(n) = T(n-1) + T(n-1) + c = 2T(n-1) + c$$

Text Segmentation Analysis

```
Splittable(A[1..n], i):
  If i > n:
    return True
  Else:
    j ← i
    for i to n:
      If IsWord(i, j):
        If Splittable(A[1..n], j + 1):
          return True

  return False
```

$$\boldsymbol{T(n) = 2T(n-1) + c \ \leq O(2^n)}$$

(You can show with a recursion tree)

Since we do not know the running time of $IsWord(i, j)$, we will instead express the running time as the number of calls we make to it.

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + O(n)$$

$$T(n) \leq \sum_{i=0}^{n-1} T(i) + cn$$

$$T(n-1) \leq \sum_{i=0}^{n-1} T(i) + c(n-1)$$

$$T(n) - \ T(n-1) \leq T(n-1) + c$$

$$T(n) = \ T(n-1) + T(n-1) + c \ = 2T(n-1) + c$$

Wrap up

Homework 2 is out - read through it ASAP!

- Problem 4 is not solvable yet, the first 3 are after this lecture.

Next time:

- Dynamic Programming

Reading Assignment: Erickson Chapter 3 (for real this time)