Pre-lecture brain teaser

We saw a linear time selection algorithm in the previous lecture. (Quick Select + MoM)

Why did we choose lists of size 5? Will lists of size 3 work? (Hint) Write a recurrence to analyze the algorithm’s running time if we choose a list of size $k$. 
ECE-374-B: Lecture 11 - Backtracking and memoization

Instructor: Abhishek Kumar Umrawal
February 27, 2024
University of Illinois at Urbana-Champaign
We saw a linear time selection algorithm in the previous lecture.

Why did we choose lists of size 5? Will lists of size 3 work?

(Hint) Write a recurrence to analyze the algorithm’s running time if we choose a list of size $k$.

$$k = 3: \quad T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

$$\Rightarrow \quad T(n) = O(n \log n)$$
$k = 7 : \quad T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{5}{7}n\right) + O(n)$
Learning Objectives
At the end of the lecture, you should be able to understand

- the details of the **quickselect** and **medians of median** algorithms,
- the idea of **backtracking** through the **8-queens puzzle**,
- the **longest increasing subsequence** problem and **recursive** algorithms to solve it,
- the intuition behind **memoization**.
Review linear time selection

Given an array $A = [0, \ldots, n - 1]$ of $n$ numbers and an index $i$, where $0 \leq i \leq n - 1$, find the $i^{th}$ smallest element of $A$.

For instance, assume $n = 20$ and $i = 10$.

The smallest element of rank 10 would be 11. But how do we figure that out

Do median of medians.....

Call $\text{Median-of-Medians}(A, 10)$
Given an array \( A = [0, ..., n - 1] \) of \( n \) numbers and an index \( i \), where \( 0 \leq i \leq n - 1 \), find the \( i^{th} \) smallest element of \( A \).

For instance, assume \( n = 20 \) and \( i = 10 \).

The smallest element of rank 10 would be 11. But how do we figure that out?

Do median of medians.....

Call **Median-of-Medians**\((A, 10)\)

First thing we need to do is find the pivot!
Given an array $A = [0, ..., n - 1]$ of $n$ numbers and an index $i$, where $0 \leq i \leq n - 1$, find the $i^{th}$ smallest element of $A$.

For instance, assume $n = 20$ and $i = 10$.

The smallest element of rank 10 would be 11. But how do we figure that out?

Do median of medians.....

Call Median-of-Medians$(A, 10)$

First thing we need to do is find the pivot!
First we reorganize:

<table>
<thead>
<tr>
<th>4</th>
<th>17</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>
Review linear time selection

First we reorganize:

<table>
<thead>
<tr>
<th>4</th>
<th>17</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

\[ k = \frac{n}{5} \]

Then we sort each column:

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>
Review linear time selection

First we reorganize:

<table>
<thead>
<tr>
<th>4</th>
<th>17</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

Then we sort each column:

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>

Still need the pivot. Find median of medians
Review linear time selection

- Call Median-of-Medians ([4,13,11,16], floor(len/2) = 2)
- Can sort this in linear time.
- Get back 13.
- 13 is our new pivot!

![Number Grid]

1  9  2  5
3 10  8  6
4 13 11 16
7 14 12 19
15 17 18 20
Review linear time selection

- Call Median-of-Medians([4, 13, 11, 16], floor(len/2) = 2)
- Can sort this in linear time.
- Get back 13.
- 13 is our new pivot!

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>
Review linear time selection

Back to our original array! Use the pivot (=13) to break it up into two.

We know the following:

- \( \text{len}(A_{\text{Lower}}) = 12 \)
- \( \text{len}(A_{\text{Upper}}) = 7 \)
- Want \( k = 10 \)
Review linear time selection

Back to our original array! Use the pivot (=13) to break it up into two.

\[
\begin{array}{cccccccccccccccccc}
4 & 3 & 15 & 7 & 1 & 17 & 9 & 10 & 14 & \color{green}{13} & 8 & 18 & 11 & 2 & 12 & 16 & 6 & 19 & 5 & 20 \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccccc}
4 & 3 & 7 & 1 & 9 & 10 & 8 & 11 & 2 & 12 & 6 & 5 & \color{green}{13} & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccccc}
\end{array}
\]

We know the following:

• \(\text{len}(A_{\text{Lower}}) = \underline{12}\)
• \(\text{len}(A_{\text{Upper}}) = \underline{7}\)
• Want \(k = 10\)

Call \textbf{Median-of-Medians}(A_{\text{Lower}}, 10)
Review linear time selection

Then we do this again:

```
| 4 | 3 | 7 | 1 | 9 | 10 | 8 | 11 | 2 | 12 | 6 | 5 |
```

First we reorganize:

```
| 4 | 10 |
| 3 |
| 7 |
| 1 |
| 9 |
| 10 |
| 8 |
| 11 |
| 2 |
| 12 |
| 6 |
| 5 |
```

Then we sort each column:

```
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
```
Review linear time selection

Then we do this again:

\[\text{\textbf{A}}_{\text{Low]} = \begin{bmatrix} 4 & 3 & 7 & 1 & 9 & 10 & 8 & 11 & 2 & 12 & 6 & 5 \end{bmatrix}\]

First we reorganize:
Review linear time selection

Then we do this again:

```
4  3  7  1  9  10  8  11  2  12  6  5
```

First we reorganize:

```
4  10
3  8  6
7  11  5
1  2
9  12
```

Then we sort each column:

```
1  2
3  8  5
4  10  6
7  11
9  12
```
Review linear time selection

- Call Median-of-Medians on the array [4, 10, 6], floor(len/2) = 1
- Can sort this in linear time.
- Get back 6.
- 6 is our new pivot!
### Review linear time selection

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Call **Median-of-Medians**([4,10,6], \(\text{floor}(\text{len}/2) = 1\))
- Can sort this in linear time.
- Get back 6.
- **6** is our new pivot!

\[ \frac{n}{2} = \text{e.g. } \frac{12}{5} \]

length: \(f(n)\)
Back to our original array! Use the pivot (=12) to break it up into two (well three).

We know the following:

\[
\begin{align*}
\text{len}(A_{\text{Lower}}) &= 5 \\
\text{len}(A_{\text{Upper}}) &= 6 \\
\text{Want } k &= 10 \text{ (pivot is of rank 6)}
\end{align*}
\]
Review linear time selection

Back to our original array! Use the pivot (=12) to break it up into two (well three).

We know the following:

- \( \text{len}(A_{\text{Lower}}) = 5 \)
- \( \text{len}(A_{\text{Upper}}) = 6 \)
- Want \( k = 10 \) (pivot is of rank 6)

Call \text{Median-of-Medians}(A_{\text{Upper}}, 10 - 6 = 4)
Review linear time selection

Then we do this again:

<table>
<thead>
<tr>
<th>7</th>
<th>9</th>
<th>10</th>
<th>8</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

First we reorganize:

<table>
<thead>
<tr>
<th>7</th>
<th>9</th>
<th>10</th>
<th>8</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

Then we sort each column:

<table>
<thead>
<tr>
<th>7</th>
<th>9</th>
<th>8</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>
Review linear time selection

Then we do this again:

| 7 | 9 | 10 | 8 | 11 | 12 |

First we reorganize:

<table>
<thead>
<tr>
<th>7</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Review linear time selection

Then we do this again:

```
7  9  10  8  11  12
```

First we reorganize:

```
7
9
10  12
8
11
```

Then we sort each column:

```
7
8
9  12
10
11
```
Review linear time selection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

- Call Median-of-Medians ([9, 12], floor(len/2) = 1)
- Can sort this in linear time.
- Get back 12.
- 12 is our new pivot!
• Call **Median-of-Medians**([9,12], floor(len/2) = 1)
• Can sort this in linear time.
• Get back 12.
• **12** is our new pivot!
Review linear time selection

Back to our original array! Use the pivot (=6) to break it up into two (well three).

\[
\begin{array}{cccccc}
7 & 9 & 10 & 8 & 11 & 12 \\
\end{array}
\]

\[
\begin{array}{cccc}
7 & 9 & 10 & 8 \\
11 & 12 \\
\end{array}
\]

We know the following:

- \(\text{len}(A_{\text{Lower}}) = 5\)
- \(\text{len}(A_{\text{Upper}}) = 0\)
- Want \(k = 4\) (pivot is of rank 5)
Review linear time selection

Back to our original array! Use the pivot (=6) to break it up into two (well three).

\[
\begin{array}{cccccc}
7 & 9 & 10 & 8 & 11 & 12 \\
\end{array}
\]

\[
\begin{array}{cccc}
7 & 9 & 10 & 8 & 11 \\
\end{array}
\]

12

We know the following:

- \( \text{len}(A_{\text{Lower}}) = 5 \)
- \( \text{len}(A_{\text{Upper}}) = 0 \)
- Want \( k = 4 \) (pivot is of rank 5)

Call \text{Median-of-Medians}(A_{\text{Lower}}, 4)
Review linear time selection

Final Step!

Can sort in linear time!

Return $\text{Sorted}(A[4]) = 11$
Median of medians time analysis

```python
Median-of-medians(A, i):
    sublists = [A[j:j+5] for j in range(0, len(A), 5)]
    medians = [sorted(sublist)[len(sublist)/2] for sublist in sublists]

    // Base Case
    if len(A) <= 5 return sorted(a)[i]

    // Find median of medians
    if len(medians) <= 5
        pivot = sorted(medians)[len(medians)/2]
    else
        pivot = Median-of-medians(medians, len/2)

    // Partitioning Step
    low = [j for j in A if j < pivot]
    high = [j for j in A if j > pivot]

    k = len(low)
    if i < k
        return Median-of-medians(low, i)
    elseif i > k
        return Median-of-medians(low, i-k-1)
    else
        return pivot
```

$T(n) = T(\frac{15}{2} n) + T(\frac{7}{10} n) + cn$
Median of medians time analysis

```
Median-of-medians(A, i):
    sublists = [A[j:j+5] for j ∈ range(0, len(A), 5)]
    medians = [sorted (sublist)[len (sublist)/2] for sublist ∈ sublists]

    // Base Case
    if len (A) ≤ 5 return sorted (a)[i]

    // Find median of medians
    if len (medians) ≤ 5
        pivot = sorted (medians)[len (medians)/2]
    else
        pivot = Median-of-medians (medians, len/2)

    // Partitioning Step
    low = [j for j ∈ A if j < pivot]
    high = [j for j ∈ A if j > pivot]

    k = len (low)
    if i < k
        return Median-of-medians (low, i)
    elseif i > k
        return Median-of-medians (low, i-k-1)
    else
        return pivot
```

\[ T(n) = T(\frac{1}{5}n) + T(\frac{7}{10}n) + cn \]
We saw a linear time selection algorithm in the previous lecture.

Why did we choose lists of size $5$? Will lists of size $3$ work?
We saw a linear time selection algorithm in the previous lecture.

Why did we choose lists of size 5? Will lists of size 3 work?

\[ T(n) = T\left(\frac{1}{3}n\right) + T\left(\frac{4}{6}n\right) + cn \]
Pre-lecture brain teaser

We saw a linear time selection algorithm in the previous lecture.

Why did we choose lists of size 5? Will lists of size 3 work?

\[ T(n) = T\left(\frac{1}{3}n\right) + T\left(\frac{4}{6}n\right) + cn \]

What about \( k = 7 \)?
We saw a linear time selection algorithm in the previous lecture.

Why did we choose lists of size 5? Will lists of size 3 work?

\[ T(n) = T\left(\frac{1}{3}n\right) + T\left(\frac{4}{6}n\right) + cn \]

What about \( k = 7 \)?

\[ T(n) = T\left(\frac{1}{7}n\right) + T\left(\frac{10}{14}n\right) + cn \]
On different techniques for recursive algorithms
**Reduction**: Reduce one problem to another

**Recursion**
A special case of reduction

- reduce problem to a **smaller** instance of itself
- **self-reduction**

- Problem instance of size \( n \) is reduced to one or more instances of size \( n - 1 \) or less.
- For termination, problem instances of small size are solved by some other method as **base cases**.
Recursion in Algorithm Design

• **Tail Recursion**: problem reduced to a single recursive call after some work. Easy to convert algorithm into iterative or greedy algorithms.

  **Examples**: Interval scheduling, MST algorithms....

• **Divide and Conquer**: Problem reduced to multiple independent sub-problems that are solved separately. Conquer step puts together solution for bigger problem.

  **Examples**: Closest pair, median selection, quick sort.

• **Backtracking**: Refinement of brute force search. Build solution incrementally by invoking recursion to try all possibilities for the decision in each step.

• **Dynamic Programming**: problem reduced to multiple (typically) dependent or overlapping sub-problems. Use memoization to avoid recomputation of common solutions leading to iterative bottom-up algorithm.
Search trees and backtracking
Q: How many queens can one place on the board?
Q: Can one place 8 queens on the board? YES!
The queens problem
The queens problem
The queens problem
The queens problem
The queens problem
The queens problem
The queens problem

Q: How many queens can one place on the board?

Q: Can one place 8 queens on the board? How many permutations?

YES

Q: How many queens can one place on the board?

Q: Can one place 8 queens on the board? How many permutations?
The eight queens puzzle

Problem published in 1848, solved in 1850.

8x8 board
8 queens
92 distinct solution
The eight queens puzzle

Problem published in 1848, solved in 1850.

$n \times n$ board

$n$ queens

Q: How to solve problem for general $n$?
Introducing concept of state tree

What if we attempt to find all the possible permutations and then check?

\[ n = 8 \quad \binom{8 \times 8 = 64}{8} = 4,426,165,368 \]

\[ \Rightarrow 140 \text{ year @ 1 second/perm.} \]
Search tree for 5 queens

Let’s be a bit smarter and recognize that:

• Queens can’t be on the same row, column or diagonal
• Can have $n$ queens max.
Search tree for 5 queens

(a)  

(b)  

"Implicit" state tree

Row1:  
Row2:  
Row3:  

"Backtracking"  

Queen 1

distinct

# solutions

= 10

24
Recursive search over an implicit tree, where we “backtrack” if certain possibilities do not work.
```cpp
void generate_permutations(int * permut, int row, int n) {
    if (row == n) {
        print_board(permut, n);
        return;
    }

    for (int val = 1; val <= n; val++)
        if (isValid(permut, row, val)) {
            permut[row] = val;
            generate_permutations(permut, row + 1, n);
        }
}
generate_permutations(permut, 0, 8);
```
Quick note: **n queens** - number of solutions

<table>
<thead>
<tr>
<th>$n$</th>
<th>Number of Solutions</th>
<th>Number of Unique Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>46</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>92</td>
</tr>
<tr>
<td>11</td>
<td>2,680</td>
<td>341</td>
</tr>
<tr>
<td>12</td>
<td>14,200</td>
<td>1,787</td>
</tr>
<tr>
<td>13</td>
<td>73,712</td>
<td>9,233</td>
</tr>
<tr>
<td>14</td>
<td>365,596</td>
<td>45,752</td>
</tr>
<tr>
<td>15</td>
<td>2,279,184</td>
<td>285,053</td>
</tr>
</tbody>
</table>
Longest Increasing Sub-sequence
**Definition**

**Sequence:** an ordered list $a_1, a_2, \ldots, a_n$. **Length** of a sequence is number of elements in the list.

**Definition**

$a_{i_1}, \ldots, a_{i_k}$ is a **subsequence** of $a_1, \ldots, a_n$ if $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

**Definition**

A sequence is **increasing** if $a_1 < a_2 < \ldots < a_n$. It is **non-decreasing** if $a_1 \leq a_2 \leq \ldots \leq a_n$. Similarly **decreasing** and **non-increasing**.
Example

- Sequence: $6, 3, 5, 2, 7, 8, 1, 9$
- Subsequence of above sequence: $5, 2, 1$
- Increasing sequence: $3, 5, 9, 17, 54$
- Decreasing sequence: $34, 21, 7, 5, 1$
- Increasing subsequence of the first sequence: $2, 7, 9$.

$3 \underline{5} \underline{5} 9 17 \underline{5} 4$ : non-decreasing
Longest Increasing Subsequence Problem

**Input**  A sequence of numbers $a_1, a_2, \ldots, a_n$

**Goal**  Find an increasing subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of maximum length

\[\begin{align*}
1 & \quad 2 & \quad 9 & \quad 12 & \quad 18 & \quad n = 5 \\
\text{LIS:} & \quad 1 & \quad 2 & \quad 9 & \quad 12 & \quad 18 & \quad \#(\text{LIS}) = 5
\end{align*}\]
Longest Increasing Subsequence Problem

**Input**  A sequence of numbers $a_1, a_2, \ldots, a_n$

**Goal**  Find an increasing subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of maximum length

**Example**

- Sequence: 6, 3, 5, 2, 7, 8, 1
- Increasing subsequences: 6, 7, 8 and 3, 5, 7, 8 and 2, 7 etc
- Longest increasing subsequence: 3, 5, 7, 8
Naive Enumeration

Assume \( a_1, a_2, ..., a_n \) is contained in an array \( A \)

```
lglisNaive(A[1..n]):
    max = 0
    for each subsequence \( B \) of \( A \) do
        if \( B \) is increasing and \( |B| > max \) then
            max = |B|
    Output max
```

Length of the given seq. \( |A| = n \)

Runtime: \( O(n2^n) \)
Naive Enumeration

Assume $a_1, a_2, \ldots, a_n$ is contained in an array $A$

```python
def algLISNaive(A[1..n]):
    max = 0
    for each subsequence $B$ of $A$ do
        if $B$ is increasing and $|B| > max$ then
            max = $|B|
    Output max
```

Running time:

$O(n^2 n)$ subsequences of a sequence of length $n$ and $O(n)$ time to check if a given sequence is increasing.
Naive Enumeration

Assume $a_1, a_2, \ldots, a_n$ is contained in an array $A$

```python
algLISNaive(A[1..n]):
    max = 0
    for each subsequence $B$ of $A$ do
        if $B$ is increasing and $|B| > max$ then
            max = $|B|$
    Output max
```

Running time: $O(n2^n)$.

$2^n$ subsequences of a sequence of length $n$ and $O(n)$ time to check if a given sequence is increasing.
Can we find a recursive algorithm for LIS?

\text{LIS}(A[1..n]):
Recursive Approach: LIS: Longest increasing subsequence

Can we find a recursive algorithm for LIS?

\[
\text{LIS}(A[1..n]):
\]

- **Case 1:** Does not contain \( A[n] \) in which case \( \text{LIS}(A[1..n]) = \text{LIS}(A[1..(n - 1)]) \)
- **Case 2:** contains \( A[n] \) in which case \( \text{LIS}(A[1..n]) \) is not clear.

\[
\begin{align*}
6 & 3 & 5 & 2 & 7 & 8 & 1 \\
\text{1 is not there} & \text{1 is there} : \text{find LIS from 6 3 5 2 7 8} \\
[6 3 5 2 7 8] & \text{"not clear"} & \text{such that all entries } \leq A[n]
\end{align*}
\]
Recursive Approach: LIS: Longest increasing subsequence

Can we find a recursive algorithm for LIS?

\textbf{LIS}(A[1..n]):

- \textbf{Case 1:} Does not contain \(A[n]\) in which case \(\text{LIS}(A[1..n]) = \text{LIS}(A[1..(n - 1)])\)
- \textbf{Case 2:} contains \(A[n]\) in which case \(\text{LIS}(A[1..n])\) is not so clear.
Can we find a recursive algorithm for LIS?

LIS(A[1..n]):

- **Case 1:** Does not contain $A[n]$ in which case $\text{LIS}(A[1..n]) = \text{LIS}(A[1..(n-1)])$
- **Case 2:** contains $A[n]$ in which case $\text{LIS}(A[1..n])$ is not so clear.

**Observation**
For second case we want to find a subsequence in $A[1..(n-1)]$ that is restricted to numbers less than $A[n]$. This suggests that a more general problem is $\text{LIS\_smaller}(A[1..n], x)$ which gives the longest increasing subsequence in $A$ where each number in the sequence is less than $x$. 

Example

Sequence: \( A[1..5] = 5, 9, 7, 8, 1 \)

\[
\begin{align*}
ss &= [] \\
x &= \text{inf}
\end{align*}
\]

\[
\begin{align*}
ss &= [] \\
x &= \text{inf}
\end{align*}
\]

\[
\begin{align*}
ss &= [1] \\
x &= 1
\end{align*}
\]

\[
\begin{align*}
ss &= [] \\
x &= \text{inf}
\end{align*}
\]

\[
\begin{align*}
ss &= [8] \\
x &= 8
\end{align*}
\]

\[
\begin{align*}
ss &= [] \\
x &= \text{inf}
\end{align*}
\]

\[
\begin{align*}
ss &= [7] \\
x &= 7
\end{align*}
\]

\[
\begin{align*}
ss &= [8] \\
x &= 8
\end{align*}
\]

\[
\begin{align*}
ss &= [78] \\
x &= 7
\end{align*}
\]
**Recursive Approach**

**LIS\_smaller**$(A[1..n], x)$: length of longest increasing subsequence in $A[1..n]$ with all numbers in subsequence less than $x$

\[
\text{LIS}\_\text{smaller}(A[1..n], x): \\
\text{if } (n = 0) \text{ then return 0} \\
\text{if } (A[n] < x) \text{ then} \\
\hspace{1cm} m = \max(m, 1 + \text{LIS}\_\text{smaller}(A[1..(n-1)], A[n])) \\
\text{Output } m
\]

**LIS**(A[1..n]):

return **LIS\_smaller**$(A[1..n], \infty)$
Running time analysis
Running time of LIS([1..n])

**LIS_smaller**(*A[1..n], x*):

if \( n = 0 \) then return 0

\[ m = \text{LIS_smaller}(A[1..(n-1)], x) \]

if \( A[n] < x \) then

\[ m = \max(m, 1 + \text{LIS_smaller}(A[1..(n-1)], A[n])) \]

Output \( m \)

**LIS**(*A[1..n]*):

return **LIS_smaller**(*A[1..n], \infty*)

Diagram:
- \( 2^n \) subproblems
- Runtime: \( O(2^n) \)
Running time of LIS([1..n])

Lemma

LIS_smaller runs in $O(2^n)$ time.
Running time of LIS([1..n])

Lemma
LIS_smaller runs in $O(2^n)$ time.

Improvement: From $O(n2^n)$ to $O(2^n)$. 

...one can do much better using memoization!
Lemma **LIS_smaller** runs in $O(2^n)$ time.

Improvement: From $O(n2^n)$ to $O(2^n)$.

....one can do much better using memoization!