Recursion

- SUBSET-SUM
- TEXT SEGMENTATION
- LONGEST INCREASING SUBSEQUENCE.

- We got $O(2^n)$ algorithms.

Dynamic Programming.

Text Segmentation.

Input: A sequence of letters $A[i \ldots n]$

Qn: Can we segment A into meaningful words.

IsWord(i, j) - returns True if $A[i \ldots j]$ is a word.

Example: Butterfly - Yes
axbd - No.

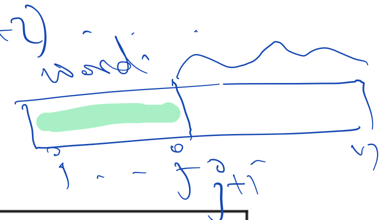
Splitable(i) = True if $A[i \ldots n]$ can be segmented into meaningful words.

$$\boxed{\text{SplitTable}(i)} = \begin{cases} \text{True} & i > n \\ \bigvee_{j=i}^{n} \left(\text{IsWord}(i,j) \wedge \text{SplitTable}(j+1)\right) & \text{o.w.} \end{cases}$$

Runtime $= O(2^n)$.

① # of potential subproblems is only $n+1$.

② SplitTable(i) depends on SplitTable$(i+1)$, ... $(i+2)$ ...



FastSplittable($A[1..n]$):
  $SplitTable[n+1] \leftarrow$ True
  for $i \leftarrow n$ down to 1
    $SplitTable[i] \leftarrow$ False
    for $j \leftarrow i$ to n
      if IsWord(i,j) and $SplitTable[j+1]$ True
        $SplitTable[i] \leftarrow$ True
  return $SplitTable[1]$

Runtime: $O(n^2)$, calls to
Isword(i, j)
$O(1)$ - time.

Dynamic Programming.

1. Write your problem (or a generalization of your problem) as a recursive formula.

"optimal substructure".

2. Count # of potential subproblems.
3. Analyze dependency of subproblems and find out an evaluation order
4. Suitable data structure to store the solutions of subproblem

Longest Increasing Subsequence.

Input: $A[1 \ldots n]$

Output: Length of a longest increasing subsequence.

$A[i_1], A[i_2] \ldots A[i_\ell]$

is a subsequence if

$i_1 < i_2 < i_3 < \ldots < i_\ell$

- increasing subsequence if

$A[i_1] < A[i_2] < \ldots A[i_\ell]$

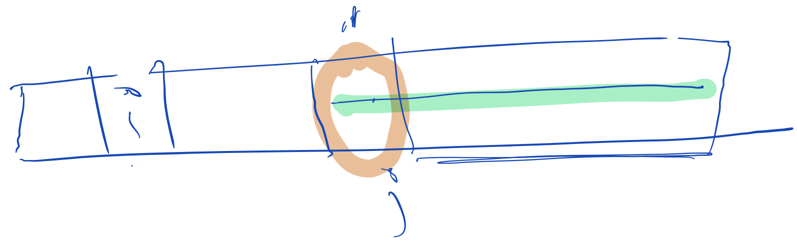
$LIS(i)$ = length of a longest increasing subsequence in $A[i \ldots n]$.

5	8	3	2	10	9	12

For $i < j$,

$\underline{\underline{LISB(i, j)}}$ = length of a longest increasing subsequence in $A[j \ldots n]$ whose

The first element in the subsequence is more than A[i].



$$
LISB(i,j) = \begin{cases} 0 & \text{if } j = n+1 \\ LISB(i,j+1) & \text{if } A[i] \geq A[j] \\ \max\begin{cases} LISB(i,j+1) \\ 1+LISB(j,j+1) \end{cases} & \text{if } A[i] < A[j] \end{cases}
$$

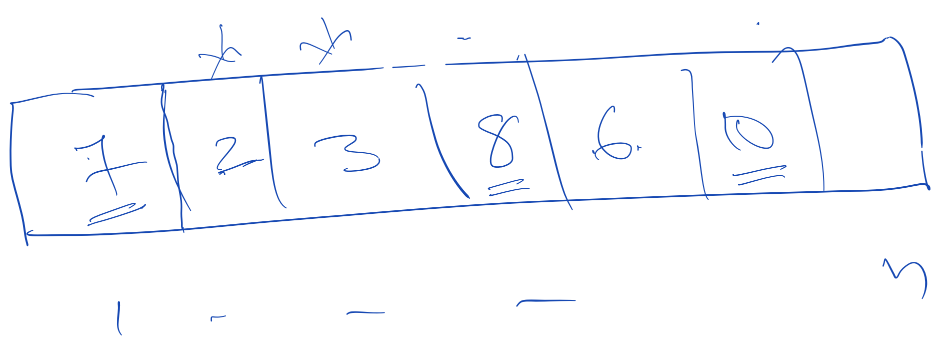
$$
\begin{array}{l} \quad 0 \quad 1 \; - \; - \; - \; - \; - \; n \\ A[-\infty, \qquad\qquad\qquad] \end{array}
$$

$LISB(0,1)$ — we want.

Data structure: Two-dimensional array.

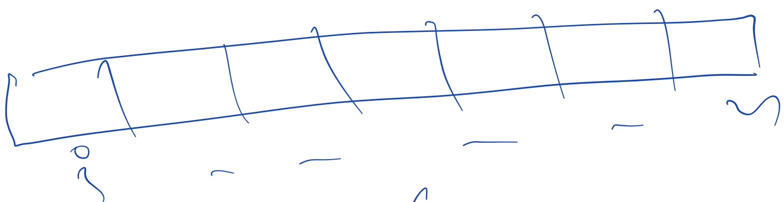
```
FastLIS(A[1..n]):
    A[0] ← −∞                                  ⟨⟨Add a sentinel⟩⟩
    for i ← 0 to n                             ⟨⟨Base cases⟩⟩
        LISbigger[i, n+1] ← 0
    for j ← n down to 1
        for i ← 0 to j − 1                     ⟨⟨... or whatever⟩⟩
            keep ← 1 + LISbigger[j, j+1]
            skip ← LISbigger[i, j+1]
            if A[i] ≥ A[j]
                LISbigger[i, j] ← skip
            else
                LISbigger[i, j] ← max{keep, skip}
    return LISbigger[0, 1]
```

Running time: $O(n^2)$.



Either $A[1]$ is first element or not.

$\mathrm{LISfirst}(i)$ = the length of the longest increasing subsequence in $A[i \ldots n]$ where the first element is $A[i]$.



$$\mathrm{LISfirst}(i) = \max\left\{ 1 + \mathrm{LISfirst}(j) : A[j] > A[i] \right\}$$

$$\mathrm{LISfirst}(n) = 1$$ Base case.

Final answer:

$$\max_i \mathrm{LISfirst}(i).$$