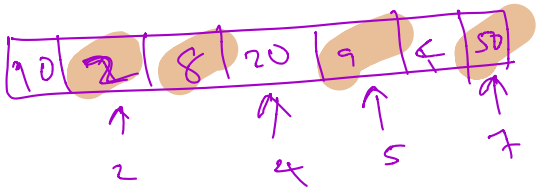
Back Tracking.

- recursive definition
- Solve it using recursive function.
- Analyse running time

→ recursive definition is correct.
† Straight forward induction.

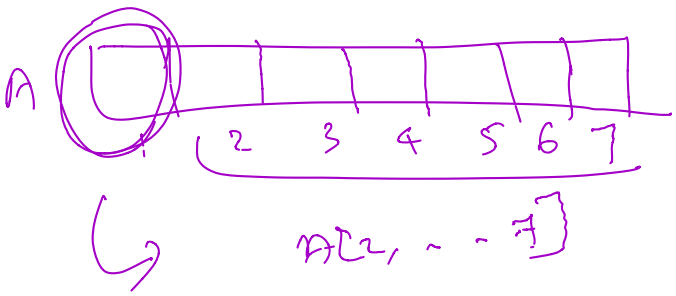
① Text segmentation
② Subset Sum
† $O(2^n)$ - time

Longest Increasing Subsequence

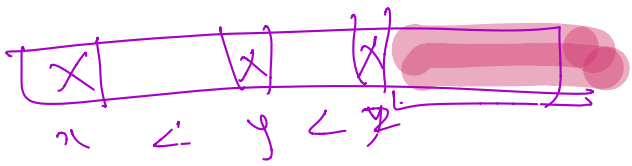


2, 20, 9, 50

Input: Given an array of n numbers
Output: Length of the longest increasing subsequence.



IS A[i] is in the sequence
then find out the
A[2, ..., i-1]
with an additional
condition that all
elements in the
sequence are ≥ A[i]



The length of the longest increasing subsequence s.t. all elements in the subsequence are > prev

```
LISBIGGER(prev, A[1..n]):
  if n = 0
      return 0
  else if A[1] ≤ prev
      return LISBIGGER(prev, A[2..n])
  else
      skip ← LISBIGGER(prev, A[2..n])
      take ← LISBIGGER(A[1], A[2..n]) + 1
      return max{skip, take}
```

A[1...n]

$$LISBigger(prev, i) = \begin{cases} 0 & \text{if } i > n \\ LISBigger(prev, i+1) & \text{if } A[i] \le prev \\ \max\begin{cases} LISBigger(prev, i+1) \\ LISBigger(A[i], i+1) \end{cases} & \text{o.w} \end{cases}$$

Correct using mathematical induction.

LISBigger($-\infty$, A[1...n]) → required output

$x < i$

LIS(x, i) is the length of longest increasing subsequence in A[i...n] where every element is > A[x]

O(1)

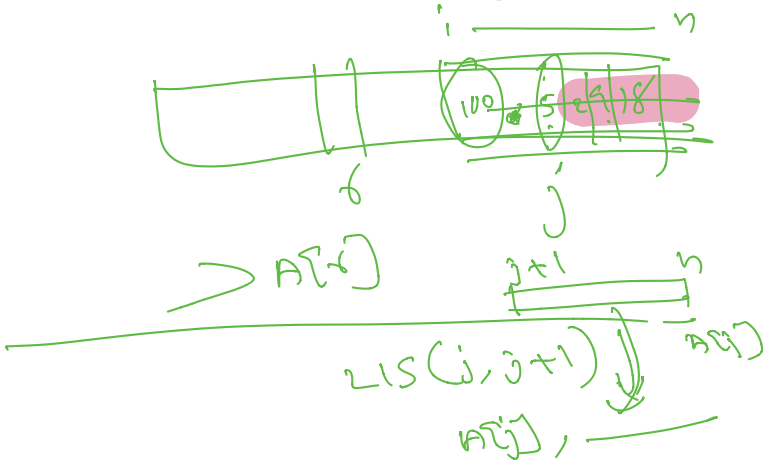
$T(0) = 1$

$T(n) = 2T(n-1) + 3$

$= O(2^n)$.

$LIS(r, i) =$

$\max\{ LIS(j, j+1) + 1 \mid$ where $j \geq i$, $A[j] > A[r] \}$



Dynamic programming.

$A[1 \ldots \ldots n]$

For $r < i$,

$LIS(r, i)$ = the length of LIS in $A[i \ldots n]$ where all elements $> A[r]$.

$$LIS(r, i) = \max \begin{cases} 0 \\ LIS(j, j+1) : \\ j \geq i \ \& \ A[j] > A[r] \end{cases}$$

L.H.S. = R.H.S

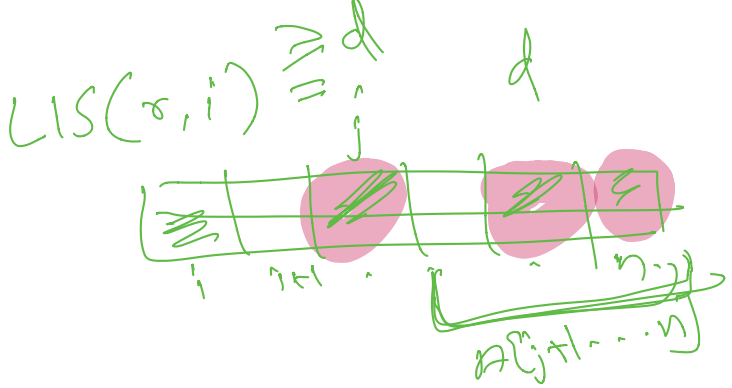
$LIS(r, n+1) = 0$

Induction on $i - (n+1)$

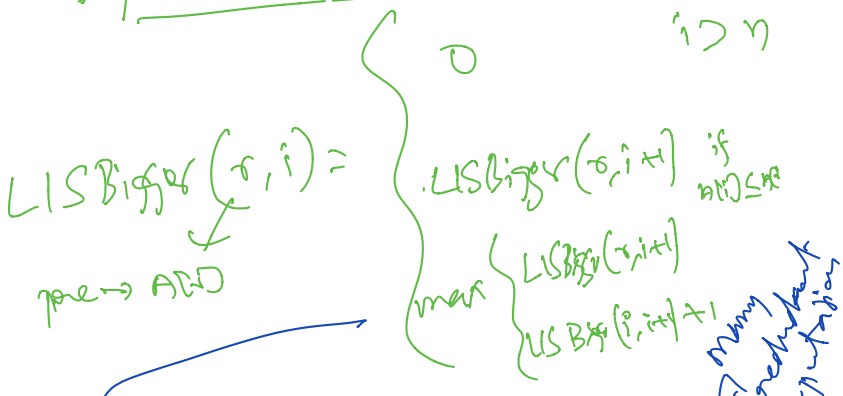
$LIS(r, i)$ - is correct

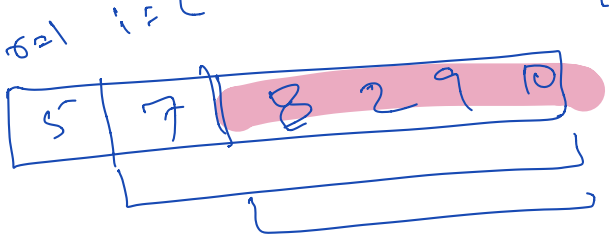
Induction: $i - (n+1) \geq 0$.

by I.H. $LIS(r, i')$ is correct $i' > i$



Dynamic Programing





$O(2^n)$

r, i+1
i, i+1

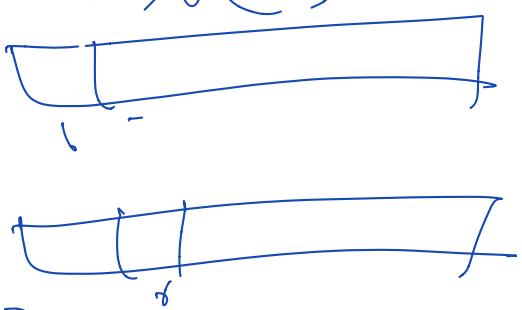
5, i+1 — (6, 4)
7, i+1 — (8, 4)

r < i

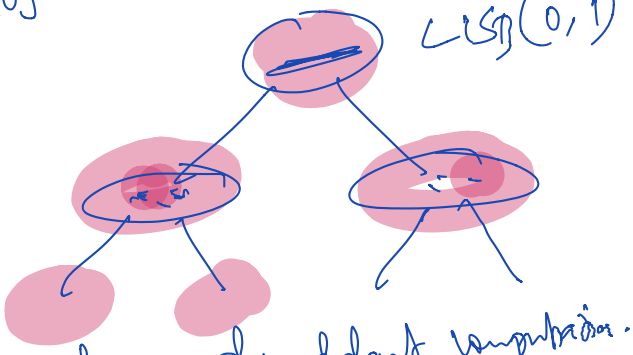
LIS(r, i)
$\hookrightarrow O(n^2)$

2^5

5 (25)

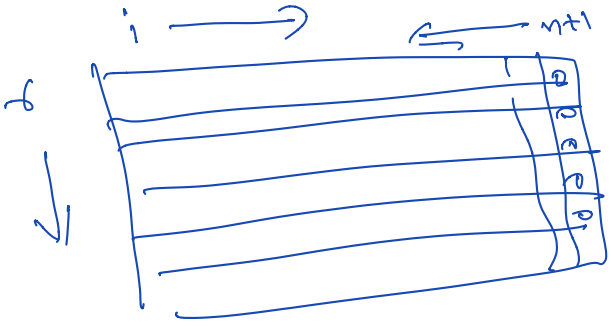


A[0] = -∞



Avoid redundant computation.

LISR(r,i) ↗



LIS(r, n+1)
LIS(r, n) ↑