Given a sequence S of numbers, find the longest subsequence containing increasing numbers. The numbers in the subsequence must occur in that order in S, but need not be consecutive in S.

Establish the problem.

- specifications – task, input, output, legal solution, optimal solution

Given a sequence S of numbers, find the longest subsequence containing increasing numbers. The numbers in the subsequence must occur in that order in S, but need not be consecutive in S.

task: find the longest subsequence containing increasing numbers

input: sequence S

output: a subsequence

legal solution: elements in subsequence are increasing and in same order as in S

optimization goal: longest subsequence

- examples

```
5 10 2 7 10 1 18 3
```

- `5 10 18` – an increasing subsequence
- `2 7 10 18` – a longer increasing subsequence

Identify avenues of attack.

- known targets
- approach

Series of choices.

- paradigms and patterns

Paradigm: dynamic programming

Flavor: find a subset

Pattern: process input – for the each element in S, include in the subsequence or not?

produce output – repeatedly determine the next element of S in the subsequence

Solution structure variation: find the best solution

- the series of choices

process input has a lower branching factor (2)

for the next element, include in the subsequence or not?

Define the algorithm.

- size

number of choices left: number of elements of S left to consider

- generalize / define subproblems
  - partial solution

the subsequence chosen from the first k elements

  - alternatives – for the next choice

include the next item in the subsequence or not

  - subproblem – solve the rest of the problem, returning the solution for the rest of the problem (only)

task: find the longest increasing subsequence in the remaining elements from k..n given the subsequence formed from elements 0..k-1

input: ~~the original sequence S~~, the current element's index k, index in S of the last thing in the partial solution

output: subsequence from elements k...n

legal solution: increasing values and in the same order as S

optimal solution: longest subsequence

  - memoization

subseq(k,last) – find the longest increasing subseq in S[k..n-1] where S[last] is the last thing in the partial solution

k, last – both 0..n-1 (can be array indexes!)

L[k][last] = length of the longest increasing subseq in S[k..n-1] where S[last] is the last thing in the partial solution

- base case(s)

// base case is a complete (legal) solution

nothing left to consider – k=n

L[n][last] = 0

- main case

if S[k] > S[last], include and not include are both options

L[k][last] = max { L[k+1][k]+1 , L[k+1][last] }

otherwise, only not include is an option

L[k][last] = L[k+1][last]

- top level
  - initial subproblem

L[0][-1]

max { L[i+1][i] } for 0 <= i < n

  - setup
  - wrapup

// determine the actual choices (the actual subseq)

// work backwards from the initial subproblem, looking for the case that led to the max value for the current subproblem

- special cases

|S| = 0 → empty subsequence (length 0)

- algorithm – determine the order of computation and write the loops

```
for ( last = 0 to n-1 ) {  L[n][last] = 0; }
for ( k = n-1 down to 0 ) {
  for ( last = 0 to n-1 ) {
    if ( S[k] > S[last] ) {   L[k][last] = max { L[k+1][k]+1 , L[k+1][last] } }
    else {  L[k][last] = L[k+1][last] }
  }
}
```

Show termination and correctness.

- termination
  - making progress
  - the end is reached
- correctness
  - establish the base case(s)
  - show the main case

    - final answer

Determine efficiency.

- implementation
- time and space
- room for improvement