1. specifications

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: subsequence

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

optimization goal: longest subsequence

2. size

3. examples

`5 10 2 7 10 1 18 3`

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

4. targets

5. tactics

6. approaches

subset

process input – for each element, include in the subsequence or not

produce output – what's the next element in the subsequence?

7. generalize / define subproblems
    a) partial solution

the subsequence built so far

    b) alternatives

process input – include or not include the current element in the subsequence

    c) subproblem

task: find the longest subsequence containing increasing numbers, given a partial subsequence already started

input: sequence S, current position, partial solution (last thing included in subsequence)

output: subsequence and its length

8. base case(s)

have a complete solution – current position is at the end

9. main case

subseq(S,k,last)

if S[k] > last

    make both choices – include and not : subseq(S,k+1,S[k]) and subseq(S,k+1,last)

    update best so far

else

    make that choice – don't include

    subseq(S,k+1,last)

    update best so far

return best so far

10. top level
    a) initial subproblem
    b) setup
    c) wrapup

11. special cases

~~12. algorithm~~

13. termination
    a) making progress

b) reaching the end

14. correctness
    a) establish the base case(s)
    b) show the main case
    c) final answer

15. implementation
    a) memoization

subseq(k,last)
- k is already integer 0..n-1
- last the index – S[last] is the last element picked

    b) order of computation
    c) dynamic programming

16. time and space