Prepared by Julian García and Phillip Abramson

Lecture 25
Dynamic programming I

FIT 1008&2085

Introduction to Computer Science

MONASH University

Information Technology

Dynamic Programming

Maximum subsequence sum (part I)

Knapsack (part II)

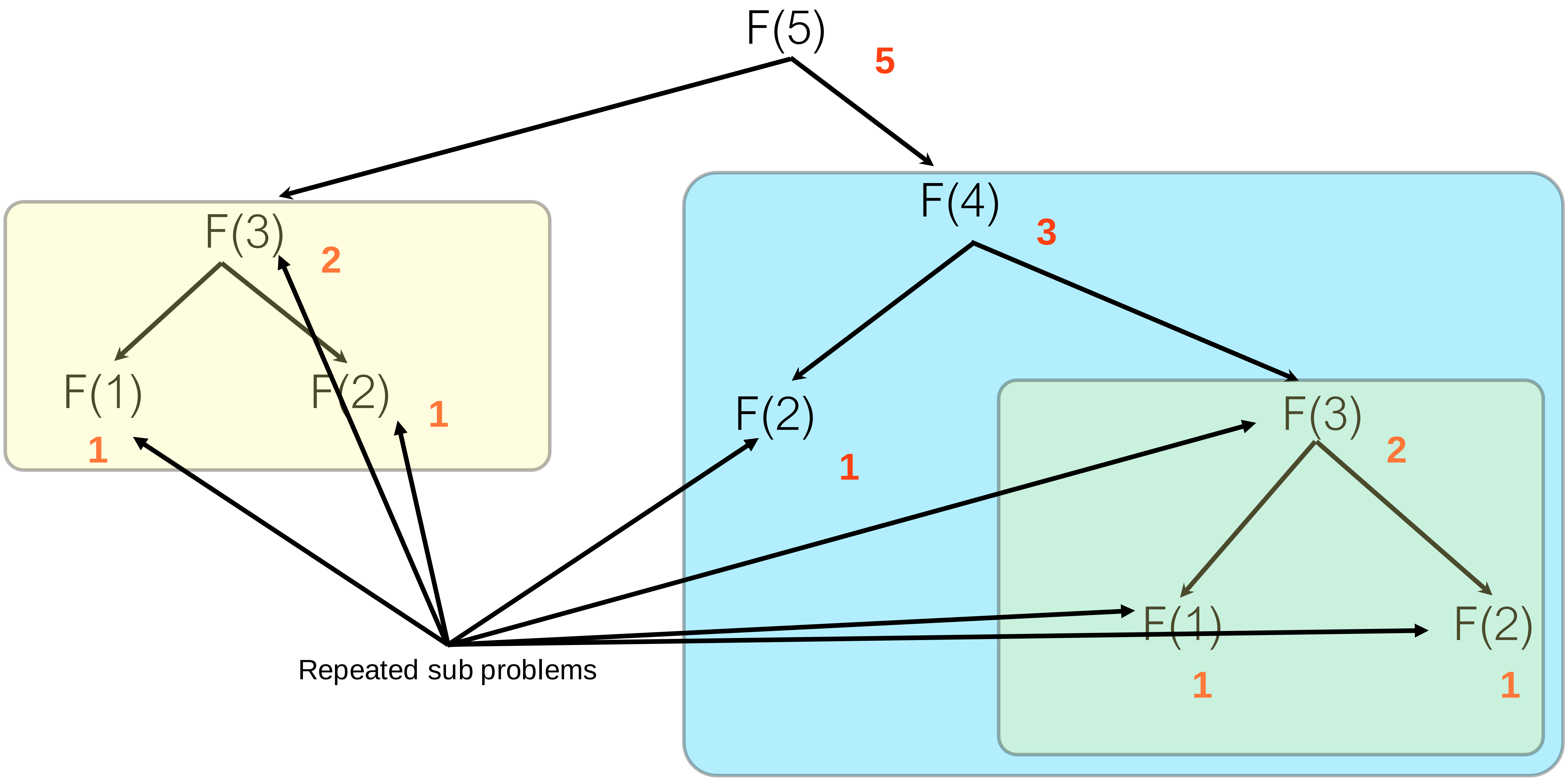
For the Fibonacci sequence, if we wanted to compute fib(5), how many times do we encounter fib(1) and fib(2)?

A) 3

B) 4

C) 5

D) None of the above


F(5)
5
F(3)
2
F(1)
1
F(2)
1
F(4)
3
F(2)
1
F(3)
2
F(1)
1
F(2)
1
Repeated sub problems

For the Fibonacci sequence, if we wanted to compute fib(5), how many times do we encounter fib(1) and fib(2)?

A) 3

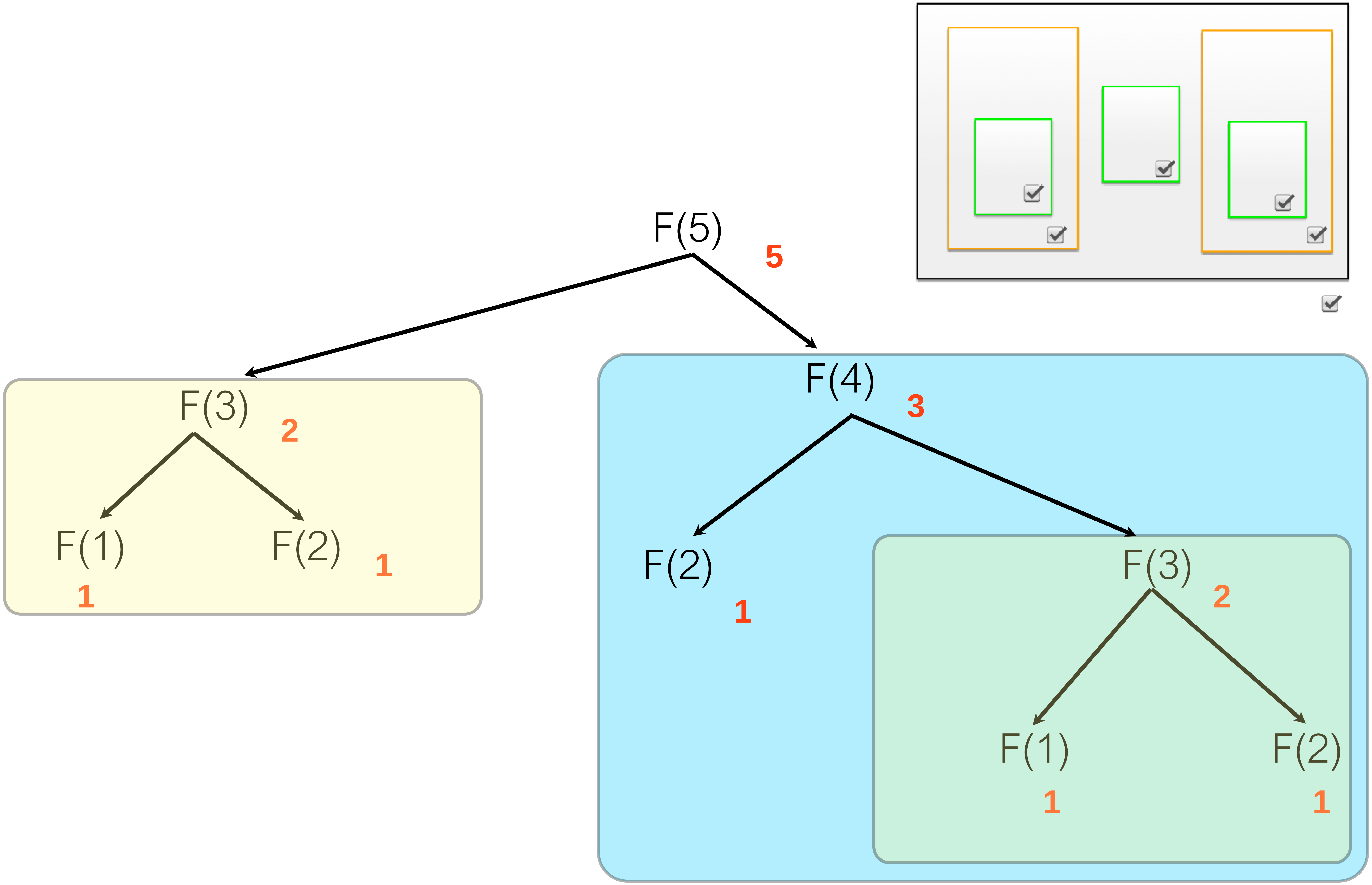
B) 4

C) 5

D) None of the above

Dynamic Programming

- Optimisation problems.
- Solves problems by **solving subproblems**.
- Subproblems may **overlap**.
- Each subproblem is solved only once, storing relevant information.



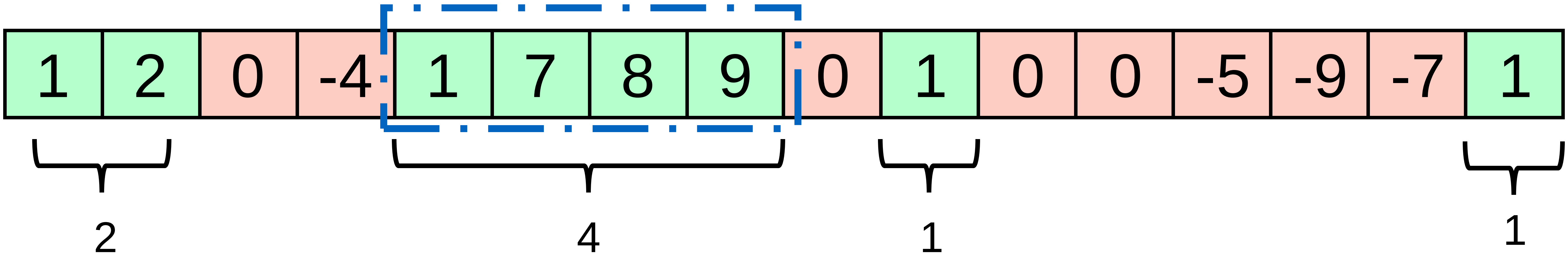
Subproblems may **overlap**.

Dynamic Programming

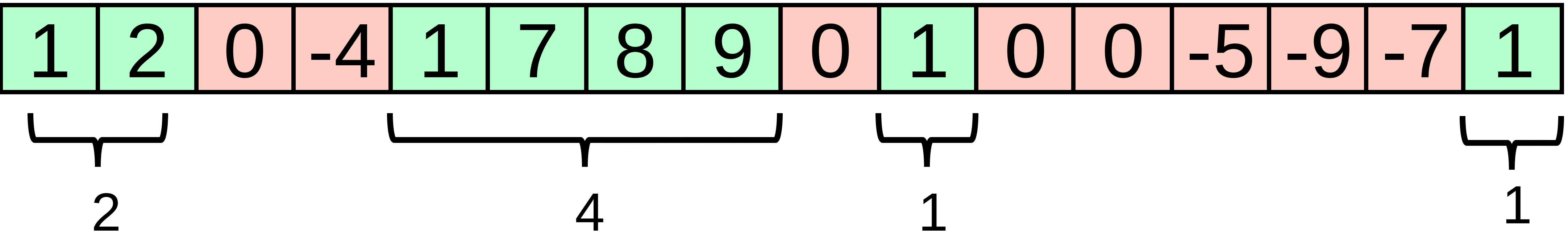
Maximum subsequence sum (part I)

Knapsack (part II)

Longest sequence of positive numbers



Longest sequence of positive numbers



Naïve approach *O(n)*

```python
def longPosSeq(the_list):
    bestStart = None
    bestLength = 0
    pos = 0
    start = None
    currentLen = 0
    while pos < len(the_list):
        if the_list[pos]>0:
            currentLen+=1
            if currentLen==1:
                start = pos
            if currentLen>bestLength:
                bestStart = start
                bestLength = currentLen
        else:
            currentLen = 0
            start = None
        pos+=1
    return [bestStart,bestLength]
```

Dynamic Programming?

Keep track of start and length of current best

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

... pos ...

Key insights:

1) It must form a sequence (if we find a non-positive element, we are starting a new sequence)
2) Any positive element is either the start of a sequence or a continuation of the sequence to the left of it

Crux of the DP solution

*If the list **ended at pos**, how long would the sequence involving the_list[pos] be?*

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

... pos ...

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

... pos ...

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

... pos ...

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

... pos ...

What is the longest positive sequence for each value of pos?

A) 2,2,2,4
B) 2,1,1,4
C) 2,0,1,4
D) 1,0,1,1
E) 4,4,4,4

If the list **ended at pos**, how long would the sequence involving the_list[pos] be?

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

What is the longest positive sequence for each value of pos?

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

A) 2,2,2,4
B) 2,1,1,4
C) 2,0,1,4
D) 1,0,1,1
E) 4,4,4,4

If the list **ended at pos**, how long would the sequence **involving the_list[pos]** be?

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

If the list **ended at pos**, how long would the sequence **involving the_list[pos]** be

Longest Positive Sequence (L)

1	2	0	0	1	?

… pos …

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

If the list **ended at pos**, how long would the sequence **involving the_list[pos]** be

Longest Positive Sequence (L)

1	2	0	0	1		?

… pos …

$$L[pos] = \begin{cases} 1+L[pos-1], & \text{where the\_list[pos]}>0 \\ 0, & \text{where the\_list[pos]}<=0 \end{cases}$$

Dynamic programming relation

This relation works for all of L?

A) Yes

B) No

Longest sequence of positive numbers

1	2	0	-4	1	7	8	9	0	1	0	0	-5	-9	-7	1

… pos …

If the list **ended at pos**, how long would the sequence **involving the_list[pos]** be

Longest Positive Sequence (L)

?	2	0	0	1		?

pos …

$$L[pos] = \begin{cases} 1+L[pos-1], & \text{where the\_list[pos]} > 0 \\ 0, & \text{where the\_list[pos]} <= 0 \end{cases}$$

Dynamic programming relation

This relation works for all of L?

A) Yes

B) No

Dynamic Programming algorithms need a relation AND initial conditions

$$L[pos] = \begin{cases} 1+L[pos-1], & \text{where the\_list[pos]>0, for all pos>0} \\ 0, & \text{where the\_list[pos]<=0, for all pos>0} \end{cases}$$

Dynamic programming relation

$$L[0] = \begin{cases} 1, & \text{where the\_list[0]>0} \\ 0, & \text{where the\_list[0]<=0} \end{cases}$$

Initial condition

Dynamic Programming algorithms need a relation AND initial conditions

$$L[pos] = \begin{cases} 1+L[pos-1], & \text{where the\_list[pos]>0, for all pos>0} \\ 0, & \text{where the\_list[pos]<=0, for all pos>0} \end{cases}$$

Dynamic programming relation

$$L[0] = \begin{cases} 1, & \text{where the\_list[0]>0} \\ 0, & \text{where the\_list[0]<=0} \end{cases}$$

Initial condition

Once L is complete (a value in each cell)
what would be the complexity to find the start
and length of the longest positive sequence?

A) O(1)
B) O(logN)
C) O(N)
D) O(N^2)

Dynamic Programming algorithms need a relation AND initial conditions

$$L[pos] = \begin{cases} 1+L[pos-1], & \text{where the\_list[pos]>0, for all pos>0} \\ 0, & \text{where the\_list[pos]<=0, for all pos>0} \end{cases}$$

Dynamic programming relation

$$L[0] = \begin{cases} 1, & \text{where the\_list[0]>0} \\ 0, & \text{where the\_list[0]<=0} \end{cases}$$

Initial condition

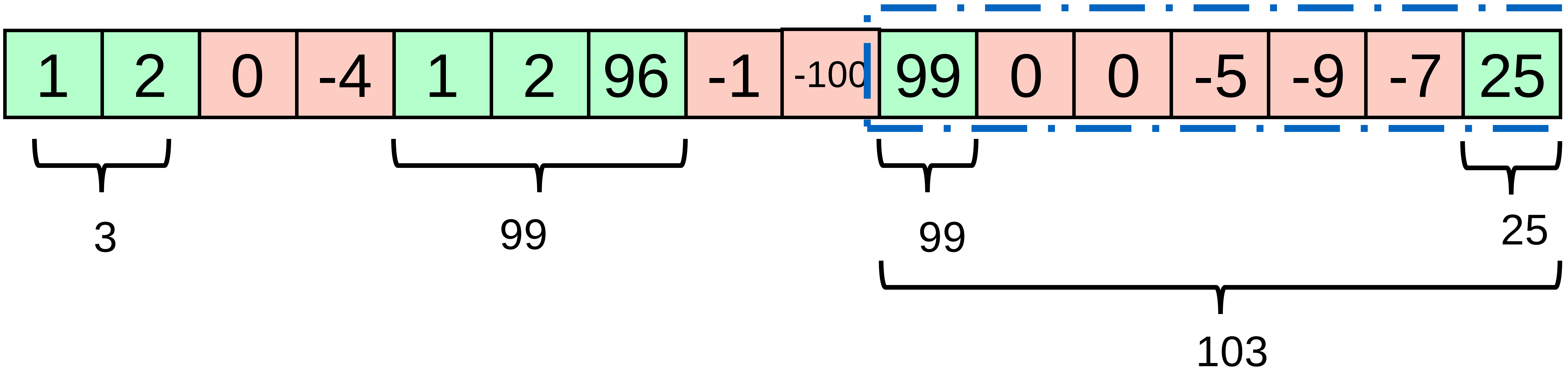
Once L is complete (a value in each cell)
what would be the complexity to find the start
and length of the longest positive sequence?

A) O(1)
B) O(logN)
C) O(N)
D) O(N^2)

O(n) to create L and O(n) to find the best sequence afterwards

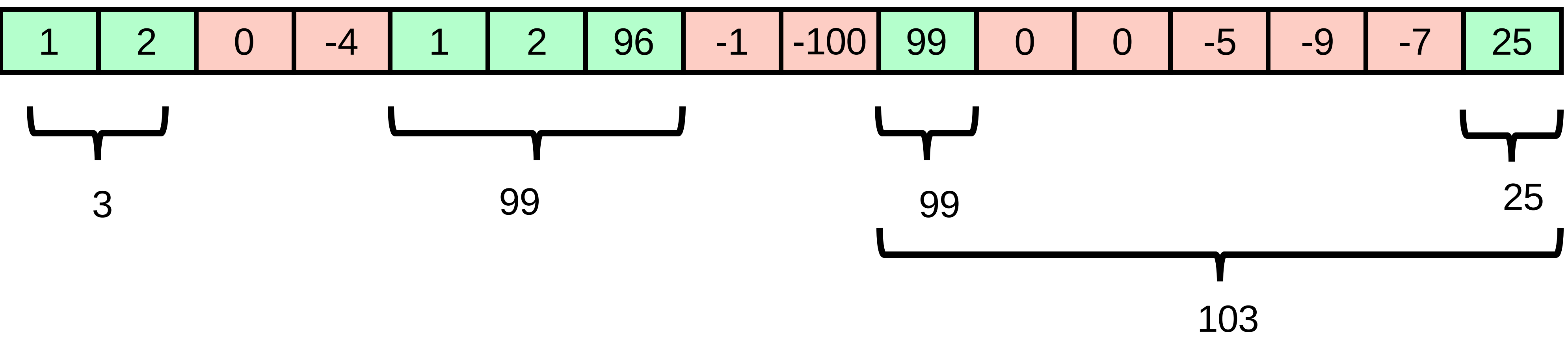
What if the values themselves had some meaning?

Maximum subsequence sum



Find a sequence the sequence of numbers within a list which is of maximal sum
(typically we ignore trailing zeroes for this)

Maximum subsequence sum



Naïve approach

$O(n^3)$

```
def maxSum(the_list):
   bestStart = None
   bestEnd = None
   bestSum = None
   for start in range(len(the_list)):
      for end in range(start,len(the_list)):
         sum = 0
         for item in range(start,end+1):
            sum += the_list[item]
         if bestSum is None or sum > bestSum:
            bestSum = sum
            bestStart = start
            bestEnd = end
   return [bestSum,bestStart,bestEnd]
```

Can be optimised to $O(n^2)$

Dynamic Programming?

Maximum subsequence sum

1	2	0	-4	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

3 99 99 25

103

Key insights?

Which of the following are true about this problem?

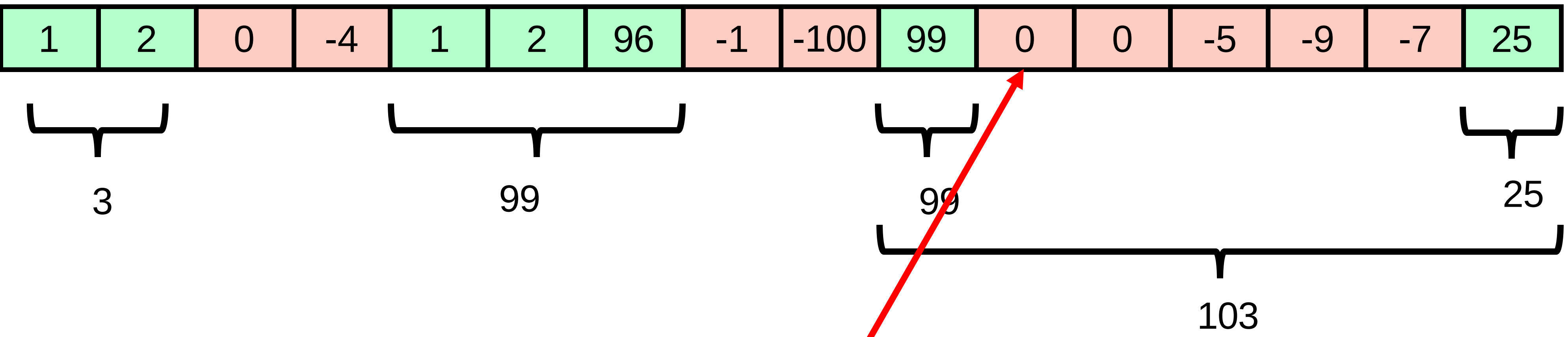
A) It must form a sequence

B) Any positive element is either the start of a sequence or a continuation of the sequence to the left of it

C) Both A and B

D) None of the above

Maximum subsequence sum



Key insights?

Which of the following are true about this problem?

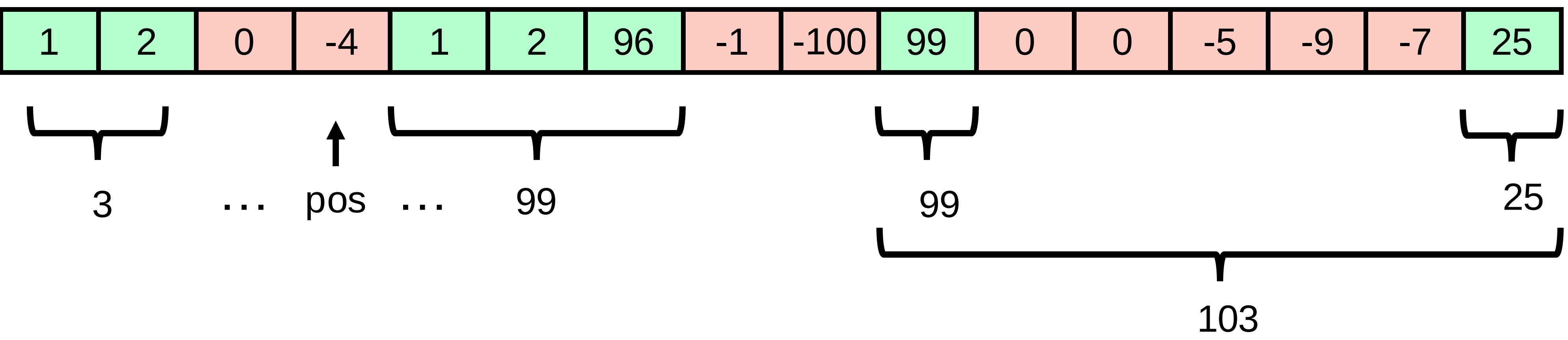
A) It must form a sequence

B) Any element at all is either the start of a sequence or a continuation of the sequence to the left of it

C) Both A and B

D) None of the above

Maximum subsequence sum



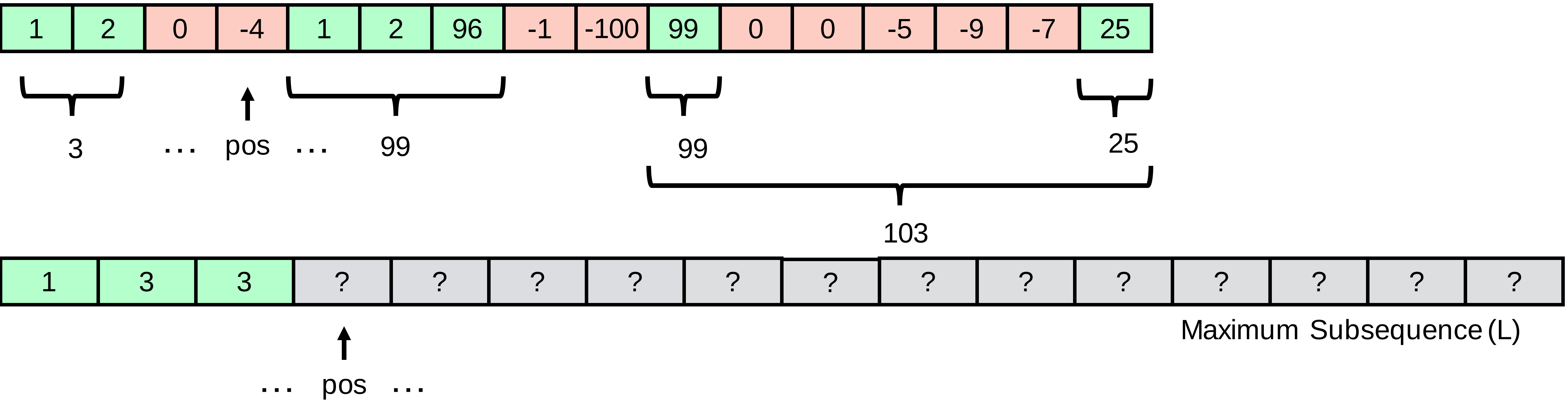
Key insights

- It must form a sequence
- Any element is either the start of a sequence or a continuation of the sequence to the left of it
- If a sequence’s sum is non-negative, it is always better to try to extend it

Crux of DP approach

- If we ended the list as pos, what would be the sum of the best sequence involving the_list[pos]?

Maximum subsequence sum



What is the maximum sum of a sequence ending at pos?

A. 0
B. 3
C. 4
D. -1
E. Cannot be determined

Maximum subsequence sum

1	2	0	-4	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

3 ... pos ... 99 99 25

103

1	3	3	-1	?	?	?	?	?	?	?	?	?	?	?	?

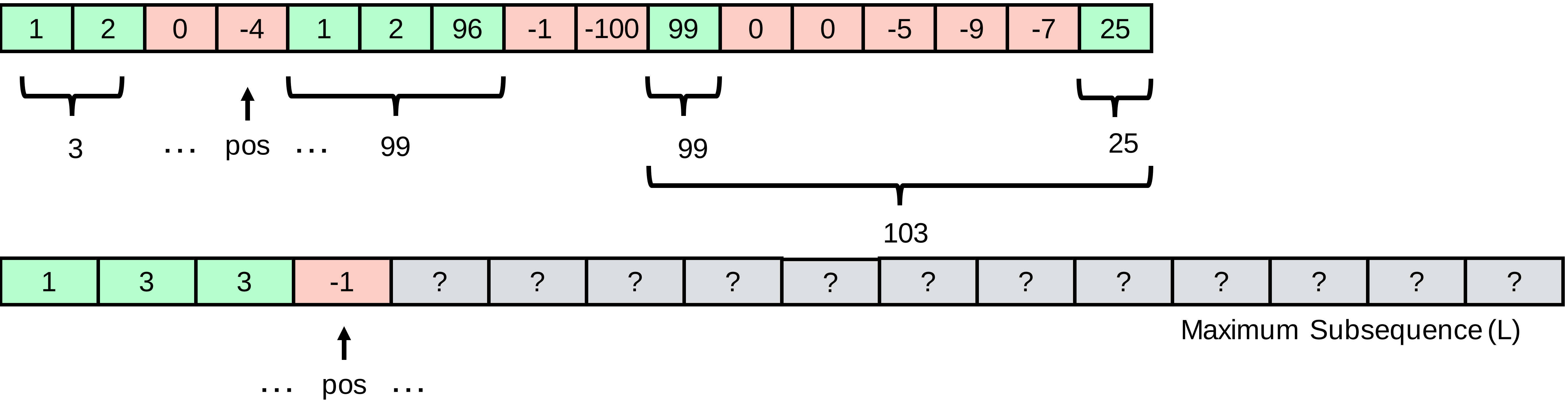
Maximum Subsequence (L)

... pos ...

What is the **maximum** sum a sequence **ending at pos**?

A. 0
B. 3
C. 4
D. -1 1+2+0-4 = -1
E. Cannot be determined

Maximum subsequence sum



Extend existing sequence

$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}$$

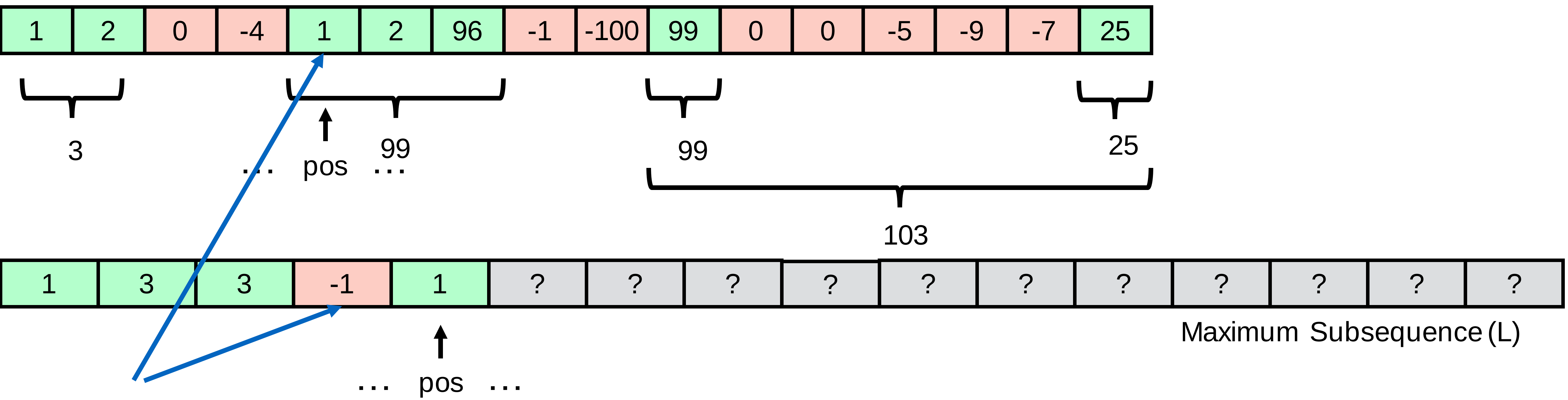
Start a new sequence

Dynamic programming relation

Key insights

- It must form a sequence
- Any element is either the **start of a sequence** or a **continuation of the sequence to the left** of it
- If the left sequence is non-negative, it is always better to extend it

Maximum subsequence sum

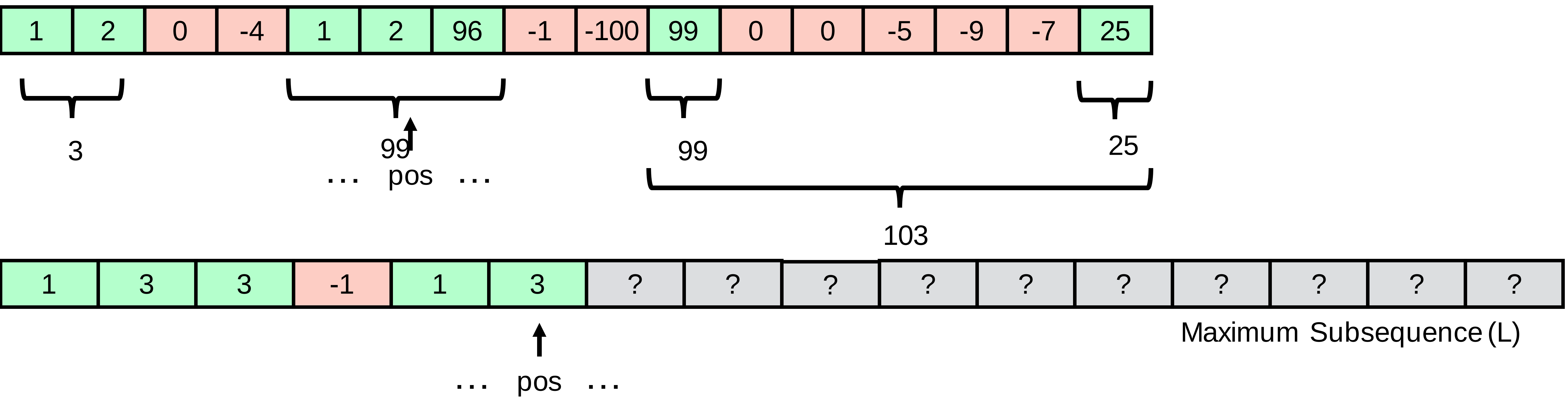


Do we extend (-1+1)?
Or start again (1)?

1 > (1+-1) so we start again

$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds the\_list[pos]} \\ the\_list[pos], & \text{otherwise} \end{cases}$$

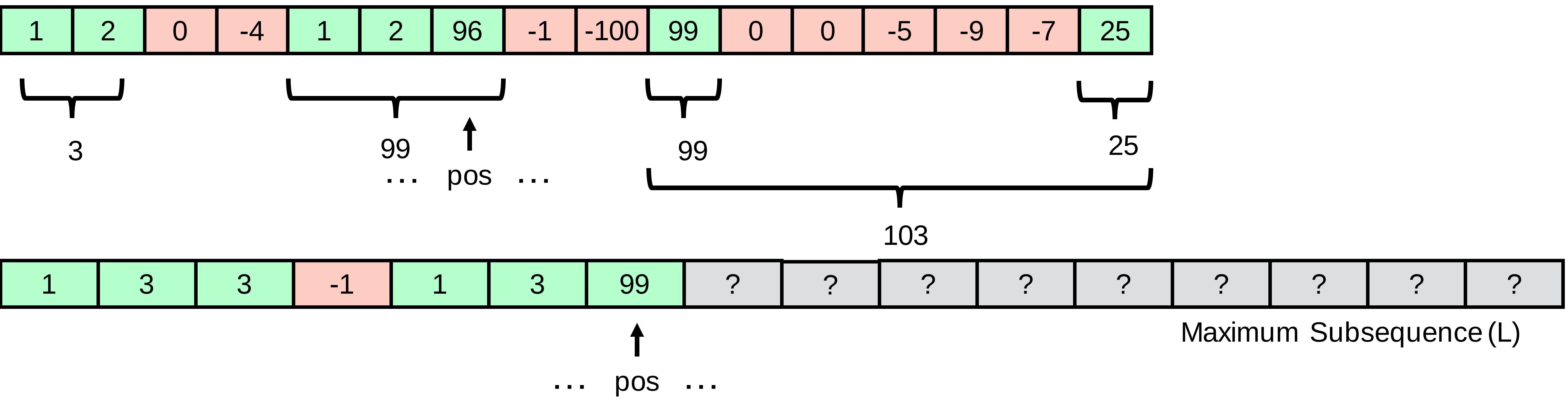
Maximum subsequence sum



2 < (2+1) therefore better to extend

$$L[pos] = \begin{cases} L[pos-1] + the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}$$

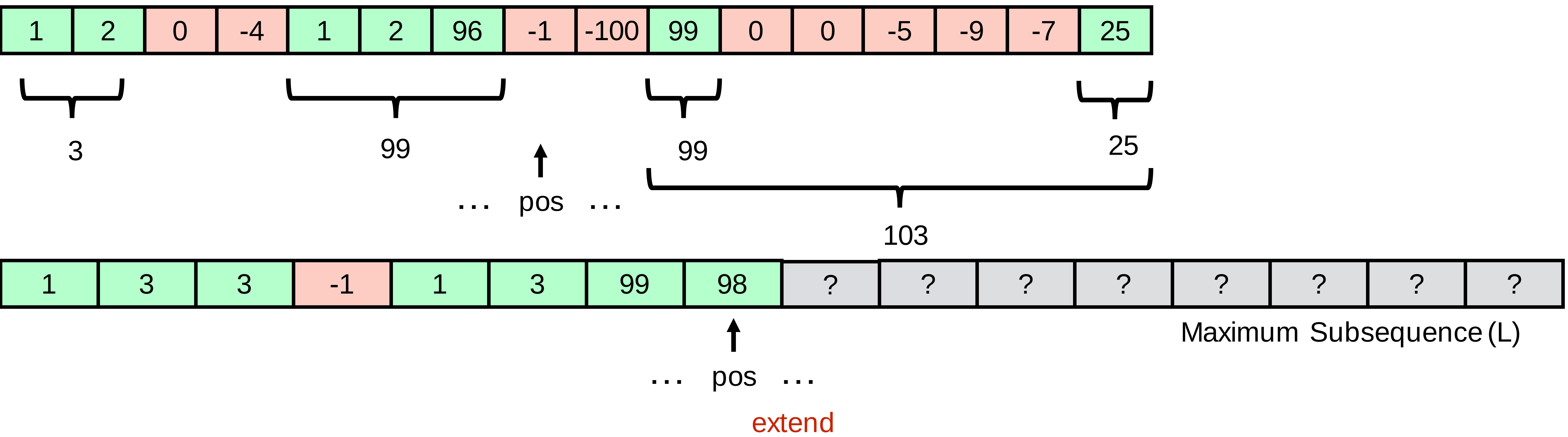
Maximum subsequence sum



96 < (96+3) so again we extend

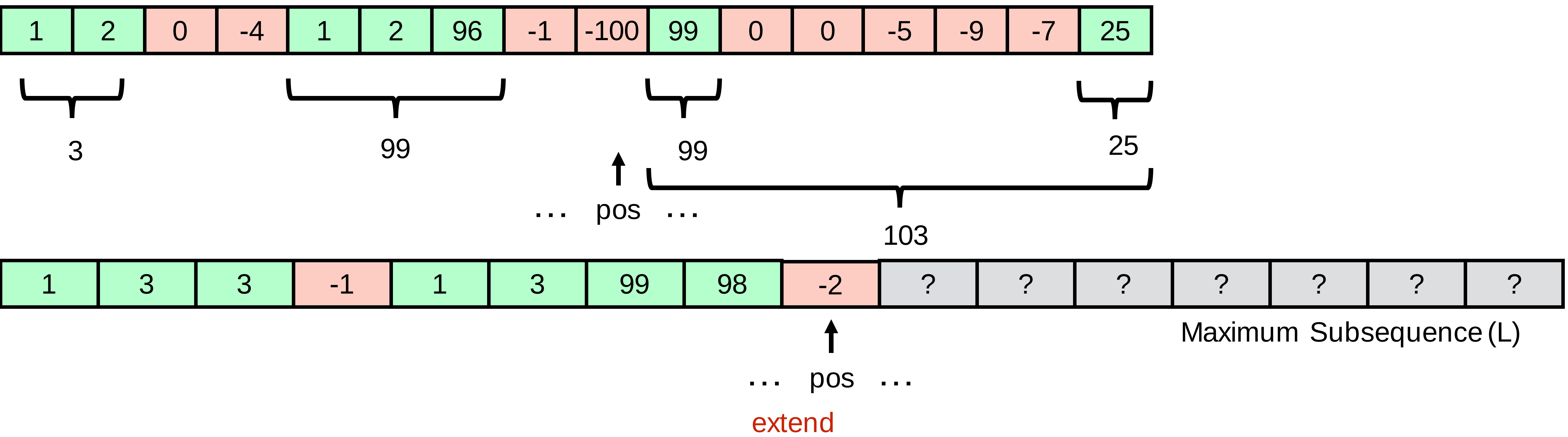
$$
L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds the\_list[pos]} \\ the\_list[pos], & \text{otherwise} \end{cases}
$$

Maximum subsequence sum



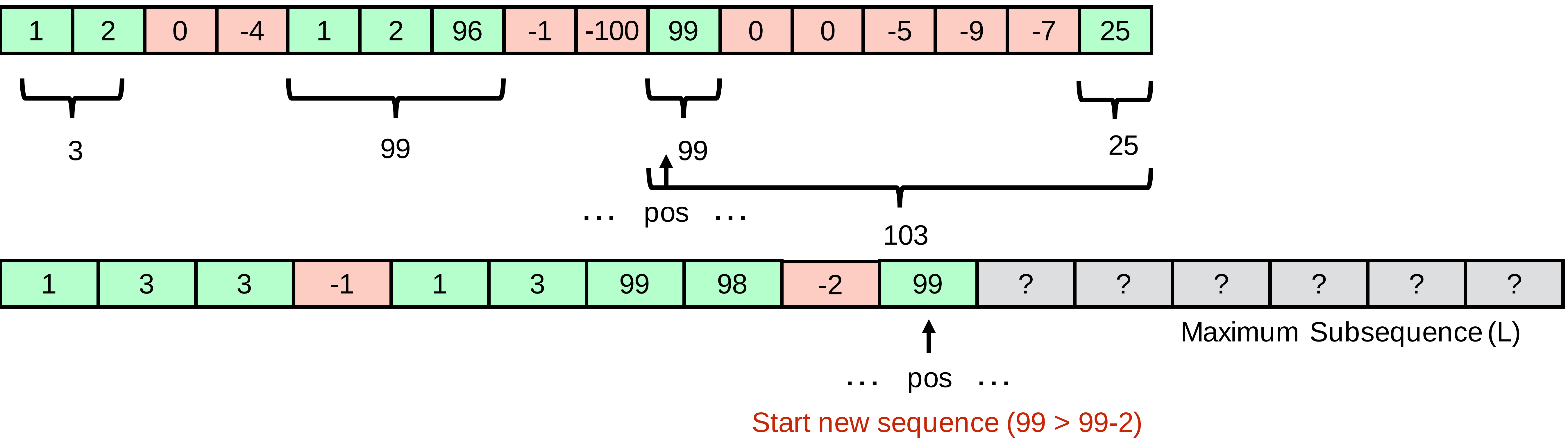
$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds the\_list[pos]} \\ the\_list[pos], & \text{otherwise} \end{cases}$$

Maximum subsequence sum



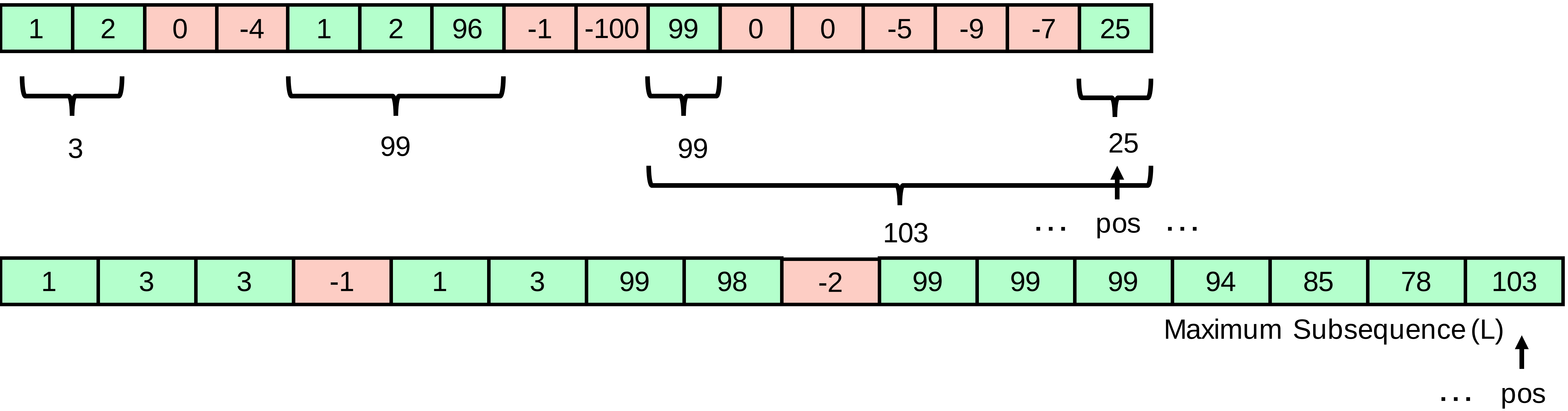
$$
L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}
$$

Maximum subsequence sum



$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds the\_list[pos]} \\ the\_list[pos], & \text{otherwise} \end{cases}$$

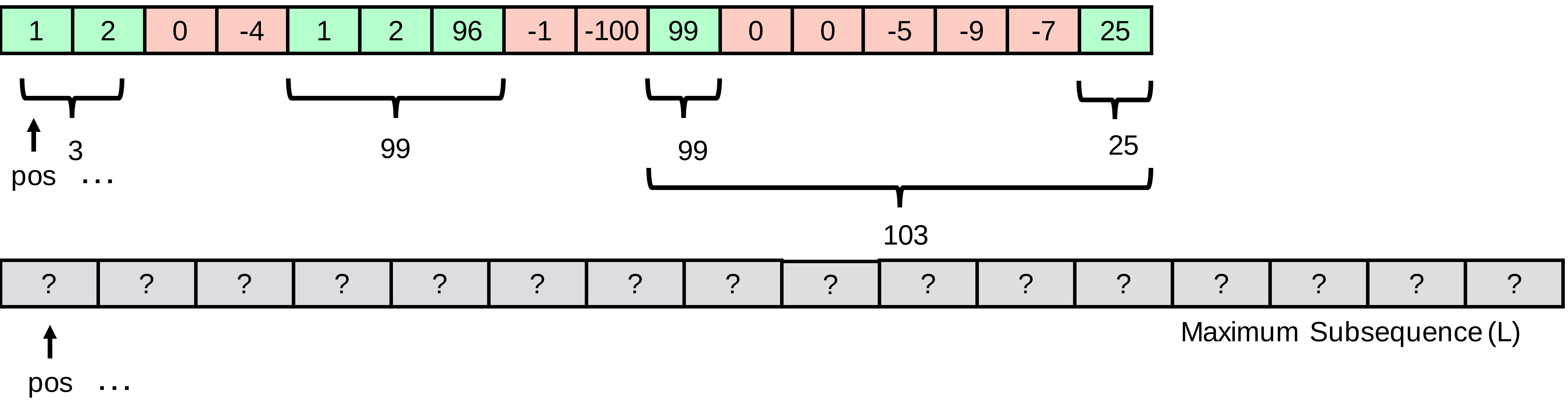
Maximum subsequence sum



$$L[pos] = \begin{cases} \text{L[pos-1]+the\_list[pos],} & \text{where exceeds the\_list[pos]} \\ \text{the\_list[pos],} & \text{otherwise} \end{cases}$$

What about the initial condition?

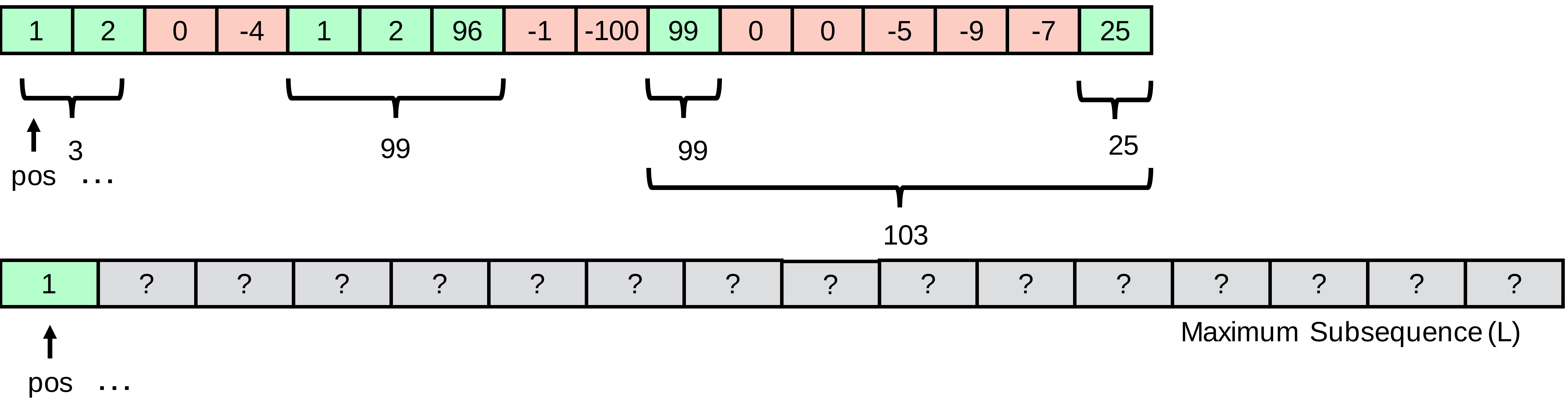
Maximum subsequence sum



What is the maximum sum for a sequence ending at pos?

A. 0
B. 1
C. 3
D. 103
E. Cannot be determined

Maximum subsequence sum



What is the maximum sum for a sequence ending at pos?

A. 0

B. 1

C. 3

D. 103

E. Cannot be determined

Begins and ends at position 0

Maximum subsequence sum

1	2	0	-4	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

3

99

99

25

103

... pos ...

1	3	3	-1	1	3	99	98	-2	99	99	99	94	85	78	103

Maximum Subsequence (L)

... pos

L[pos] = { L[pos-1]+the_list[pos], where exceeds the_list[pos]
the_list[pos], otherwise

For pos > 0 relation

L[0] = the_list[0]

Initial condition

For pos = 0

Maximum subsequence sum

1	2	0	-4	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

3 99 99 25

103 … pos …

1	3	3	-1	1	3	99	98	-2	99	99	99	94	85	78	103

Maximum Subsequence (L)

… pos

$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}$$

For pos > 0 relation

L[0] = the_list[0] Initial condition

For pos = 0

Will the maximum subsequence sum always be found at the end?

A)Yes

B)No

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

... pos ...

90+90 (+0 if you wish) = 180 which is the maximum sum possible

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

... pos ...

*Maximum subsequence sum is known once we complete L but it will be the **maximum** in the array*

$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}$$

For pos > 0 relation

L[0] = the_list[0]

Initial condition

For pos = 0

Will the maximum subsequence sum always be found at the end?

A)Yes

B)No

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

Solid blue box = position considered

Last element in L is positive so the end of a decent sequence sum

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

103 - X

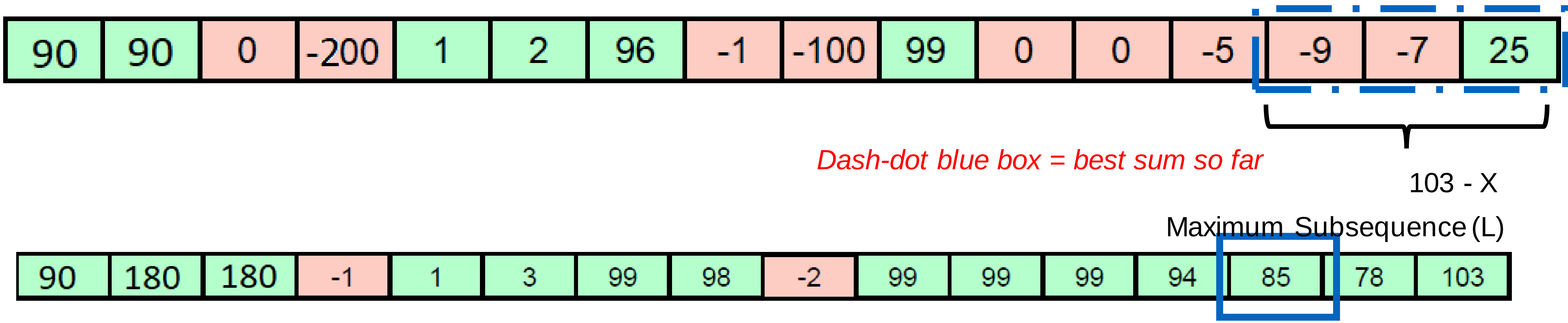
Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] > 0 and under 103
Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum



Dash-dot blue box = best sum so far

L[pos] > 0 and under 103
Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

103 - X

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] > 0 and under 103

Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

103 - X

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] > 0 and under 103
Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

103 - X

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] > 0 and under 103
Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

Dash-dot blue box = best sum so far

103 - X

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] > 0 and under 103
Shift the start of the sequence back

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

dashed red box = sequence of lower sum than best so far

103

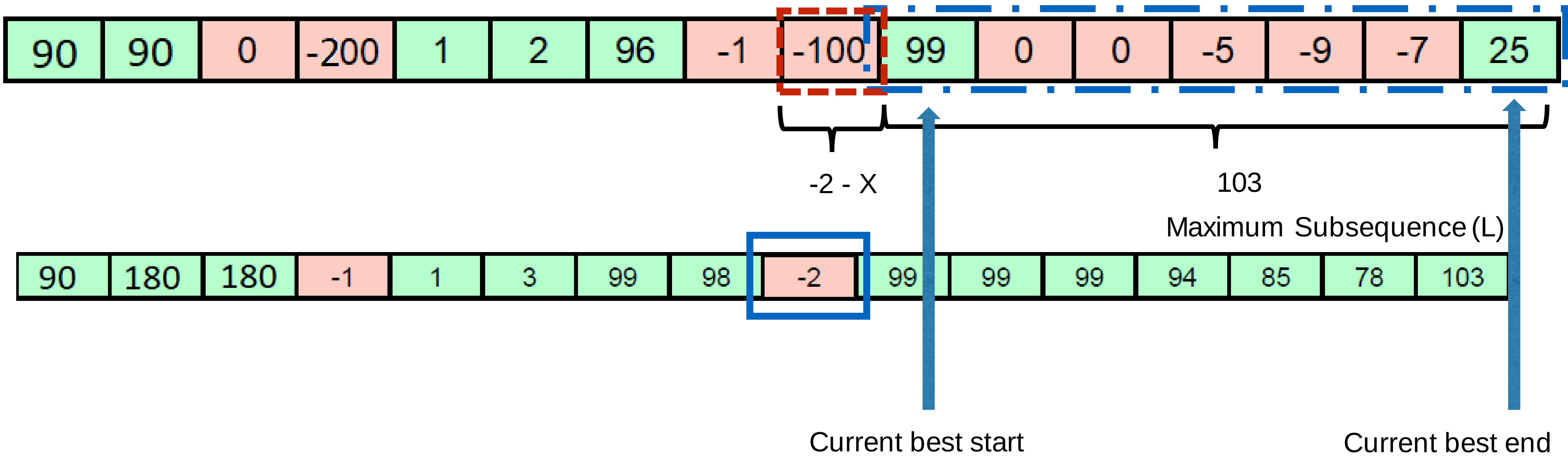
Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] = -2
As L[pos] <= 0 sequence ended

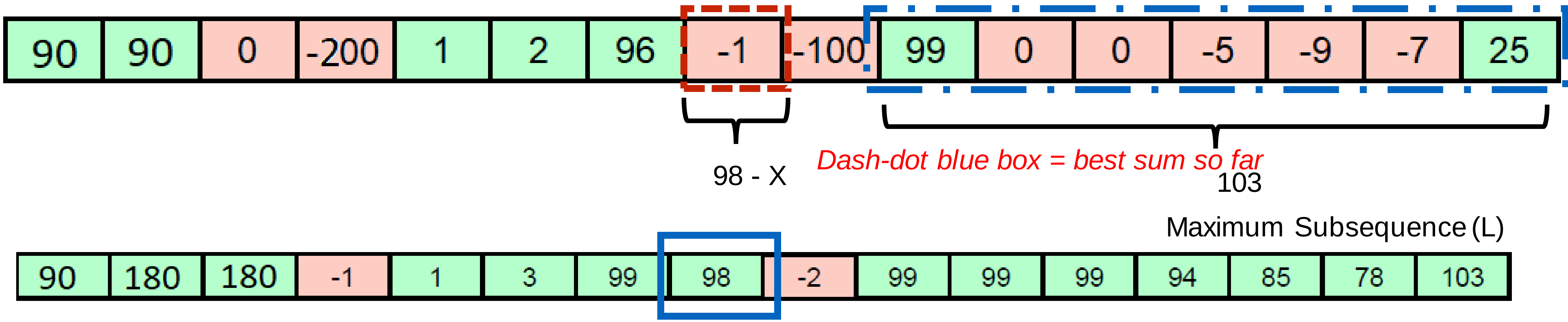
How do we find the maximum subsequence sum from L?

Maximum subsequence sum



How do we find the maximum subsequence sum from L?

Maximum subsequence sum



L[pos] = 98

As L[pos] >0 and L[pos] > -2

better sequence sum than -2 ends here

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

99 - X

Dash-dot blue box = best sum so far

103

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] = 99
As L[pos] >0 and L[pos] > 98
better sequence sum than 98 ends here

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

99 - X

Dash-dot blue box = best sum so far

103

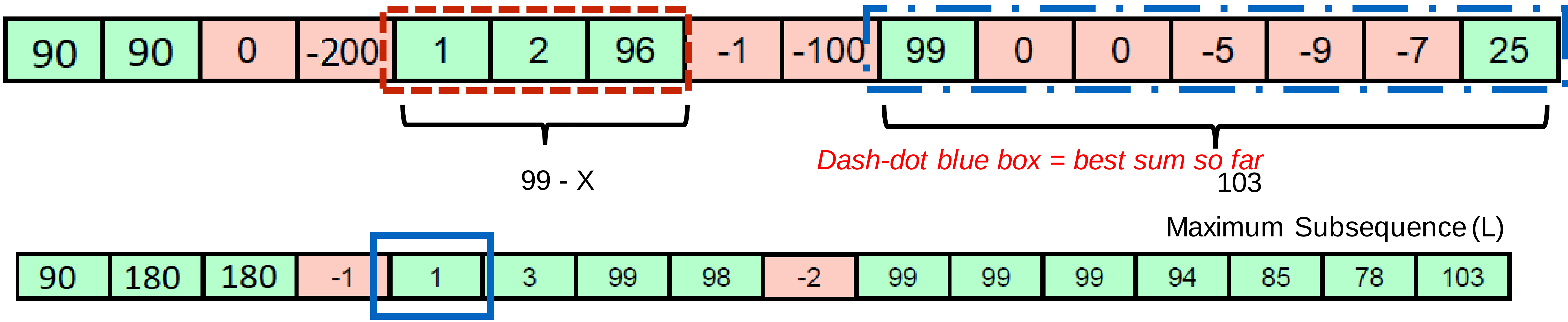
Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] = 3
L[pos] >0 but L[pos] < 99
Extends the 99 sequence sum

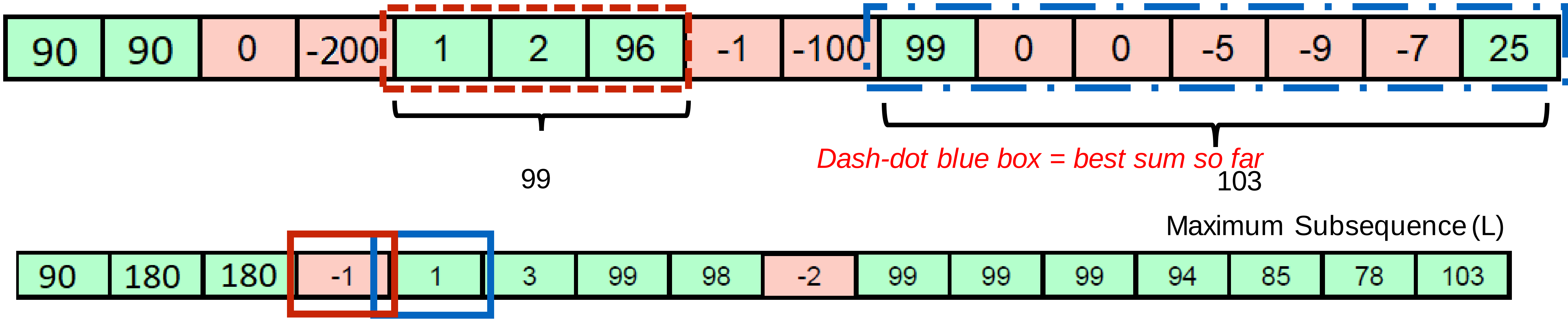
How do we find the maximum subsequence sum from L?

Maximum subsequence sum



How do we find the maximum subsequence sum from L?

Maximum subsequence sum



L[pos] < 0
Start of sequence found but no better than 103 so discarded

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

-1 - X

Dash-dot blue box = best sum so far

103

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

180 - X

Dash-dot blue box = best sum so far

103

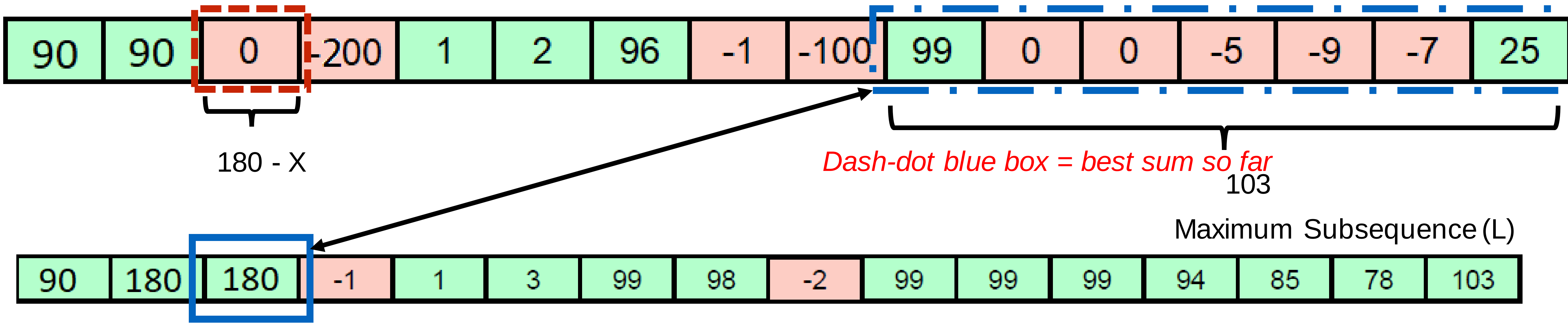
Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

L[pos] = 180
As L[pos] > 0
New sequence end found

How do we find the maximum subsequence sum from L?

Maximum subsequence sum



L[pos] = 180
As L[pos] > 0 **and L[pos] > 103**
New sequence end found **and it is the best so far**

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

$\underbrace{0}_{180 - X}$

Dash-dot blue box = best sum so far

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

180 is now the best subsequence sum so far

Current best end

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

180 - X

Dash-dot blue box = best sum so far

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

No change in moving to the left, hence the item (0)
does not contribute to the sum and we can shift the endpoint to the left

Current best end

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

180 - X

Dash-dot blue box = best sum so far

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

180

Dash-dot blue box = best sum so far

Maximum Subsequence (L)

90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

Have reached end of list without a negative or zero so this sequence ends at the left end of the list

the maximum subsequence sum is 180 and begins at position 0 and ends at position 2

How do we find the maximum subsequence sum from L?

Maximum subsequence sum

90	90	0	-200	1	2	96	-1	-100	99	0	0	-5	-9	-7	25

$\underbrace{90 \quad 90}_{180}$

Maximum Subsequence (L)

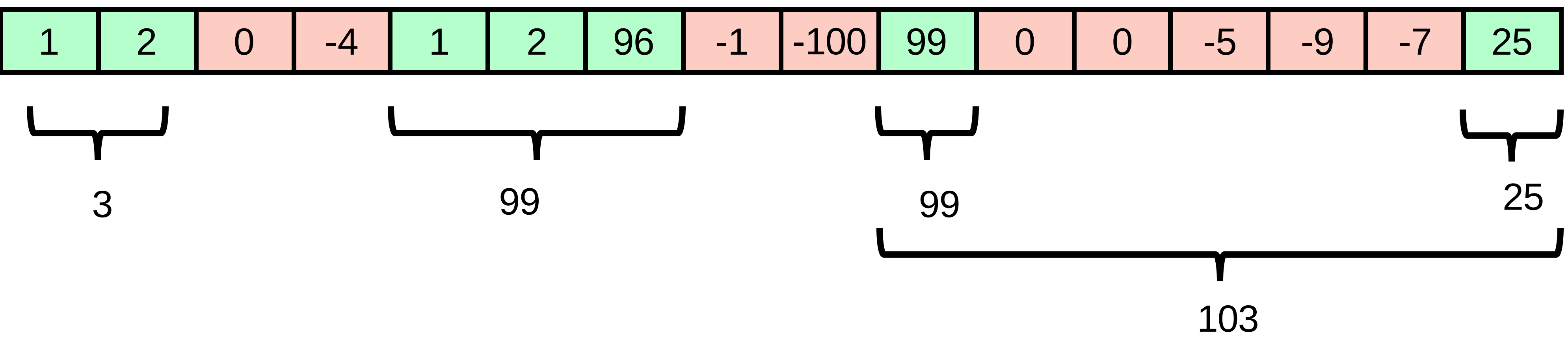
90	180	180	-1	1	3	99	98	-2	99	99	99	94	85	78	103

Go from right to left…
Find the maximum element in L --- this is the end of a sequence
The start of this sequence is to the right of the next zero in L (or is position 0 if no zeroes to the left in L)

How do we find the maximum subsequence sum from L?

Brief

Maximum subsequence sum



Naïve approach O(n^3)

```python
def maxSum(the_list):
   bestStart = None
   bestEnd = None
   bestSum = None
   for start in range(len(the_list)):
      for end in range(start,len(the_list)):
         sum = 0
         for item in range(start,end+1):
            sum += item
         if bestSum is None or sum > bestSum:
            bestSum = sum
            bestStart = start
            bestEnd = end
   return [bestSum,bestStart,bestEnd]
```

Can be optimised to O(n^2)

Dynamic Programming approach O(n)

$$L[pos] = \begin{cases} L[pos-1]+the\_list[pos], & \text{where exceeds } the\_list[pos] \\ the\_list[pos], & \text{otherwise} \end{cases}$$

For pos > 0 relation

L[0] = the_list[0]

Initial condition

For pos = 0

Dynamic Programming

Maximum subsequence sum (part I)

Knapsack (part II)