# PROGRAMMING

Lecture 19

Dept. of Computer Engineering
Hanbat National University

# OUTLINE



Maximum subsequence sum problem

Brute force enumeration

Incremental computation

Divide and conquer

Dynamic programming

# MAX. SUBSEQUENCE SUM PROBLEM

Given a sequence of n numbers, X = $[x_0, x_1 \ldots\ldots, x_{n-1}]$, find a subsequence $X^*$ in X such that

(1) The numbers in $X^*$ is **contiguous** in X

(2) The sum of the numbers in $X^*$ is the **maximum** over all contiguous subsequences of S.

(3) The sum of numbers in $X^*$ is **positive**. Is this needed?

X = [ 31, -41, $\boxed{59, 26, -53, 58, 97,}$ -93, -23, 84]

187

2 6

$X^*$ = X[2 : 7]

# Observations

$X = [x_0, x_1, \ldots.. , x_{n-1}]$

What if $x_i > 0$ for all $0 \leq i < n$ ?

What if $x_i < 0$ for all $0 \leq i < n$ ?

## Basic idea

For **all possible** subsequences, find their sums and compare the results to choose the subsequence with the maximum sum.

How many subsequences?
How to enumerate them?

# Pseudo code

1. **Enumerate** all subsequences.

2. For each subsequence, compute the **sum** of elements.

3. Compute the **maximum** by comparing the sum of every subsequence

## Step 1: How many subsequences?

X[L : U]

| L | U | | |
|---|---|---|---|
| 0 | 1 | X[0 : 1] | |
| | ...... | ............ | n |
| | n | X[0 : n] | |
| 1 | 2 | X[1 : 2] | |
| | ...... | ........... | n-1 |
| | n | X[1 : n] | |
| .... | ...... | ............ | |
| n-1 | n | X[n-1 : n] | 1 |

$$n + (n - 1) + \ldots\ldots + 1 = \frac{n(n+1)}{2}$$

$\therefore \frac{n(n+1)}{2}$ subsequences

**How to enumerate**?

Nested loop.

for L in range(n):
    for U in range(L+1, **n+1**):

What to do here?

**Performance Analysis**

```
n = len(X)
MaxSoFar = 0
for L in range(n):
    for U in range(L+1, n+1):   Why?
        sum= 0
        for i in range(L, U):
            sum = sum + X[i]
        if MaxSoFar < sum:
            MaxL, MaxU, MaxSoFar = L, U, sum
```

Computing the sum of X[L, U]

Finding the maximum

## Computing the sum of X[L:U]

```
sum = 0
for i in range(L, U):
    sum = sum + X[i]
```

How many elements in X[L:U]?

U - L elements.

How many additions?

U - L - 1 additions

∴ At most n - 1 additions. Why?

n – 1 additions when L = 0 and U = n..

∴ At most n – 1 additions to compute the sum of a subsequence.

# How many additions to find the solution?

```
n = len(X)
MaxSoFar = 0
for L in range(n):
    for U in range(L+1, n+1):
        sum = 0
        for i in range(L, U):
            sum = sum + X[i]
        if MaxSoFar < sum:
            MaxL, MaxU, MaxSoFar = L, U, sum
```

$n(n + 1) / 2$ subsequences

At most $n - 1$ additions for a subsequence

$\therefore$ At most $\frac{n(n+1)}{2}$ X (n -1) = $\frac{1}{2}$ ($n^3$ - n) additions

Time complexity: $O(n^3)$

Space complexity O(n) Why?

X = $[x_0, x_1, \ldots.. , x_{n-1}]$

**Observation**

```
n = len(X)
MaxSoFar = 0
for L in range(n):
    for U in range(L+1, n+1):
        sum = 0
        for i in range(L, U):
            sum = sum + X[i]
        if MaxSoFar < sum:
            MaxL, MaxU, MaxSoFar = L, U, sum
```

This many additions needed?

X[L : L+1] : 0 addition

........................

X[L : k] : k - L - 1 additions

X[L : k + 1] : k - L additions

.........................

X[L : n] : n - L - 1 additions

# INCREMENTAL COMPUTATION



X[L : k] : k - L - 1 additions
X[L : k + 1] : k – L additions

sum of X[L : k] = X[L] + X[L+1] + ...... + X[k-1]
sum of X[L : k+1] = $\underbrace{\boxed{X[L] + X[L+1] + \ldots\ldots + X[k-1]}}_{\text{sum of } X[L\,:\,k]}$ + X[k]

```
n = len(X)
MaxSoFar = 0
for L in range(n):
    for U in range(L+1, n+1):
        sum = 0
        for i in range(L, U):        sum = sum + X[U-1]
            sum = sum + X[i]
        if MaxSoFar < sum:
            MaxL, MaxU, MaxSoFar = L, U, sum
```

```
n = len(X)
MaxSoFar = 0
for L in range(n):
    sum = 0
    for U in range(L+1, n):
        sum = sum + X[U-1]
        if MaxSoFar < sum:
            MaxL, MaxU, MaxSoFar = L, U, sum
```

$O(n^2)$ additions !

## Basic Idea

1. If the sequence X has only **one element,** then return max(0, X[0]).

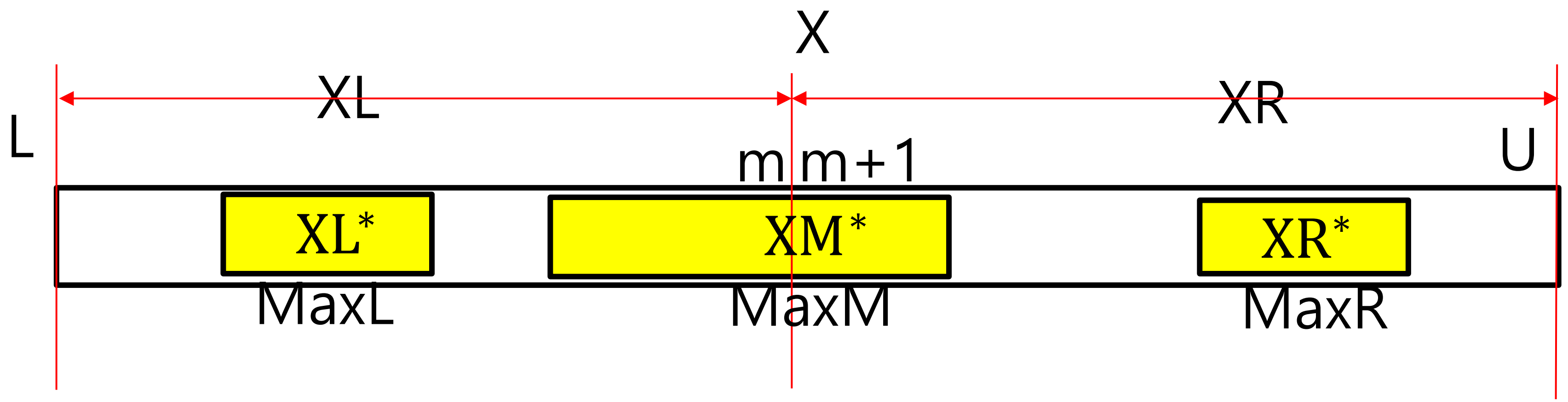
base case

2. Otherwise, **divide** the sequence into **two sub-sequences** of almost **equal size**, and **compute** the **maximum sum** for each of sub-sequences **recursively**.

recursive case

3. **Combine** the **solutions** in step 2 to solve the original problem.

## Step 3: How to combine the solutions of sub-problems

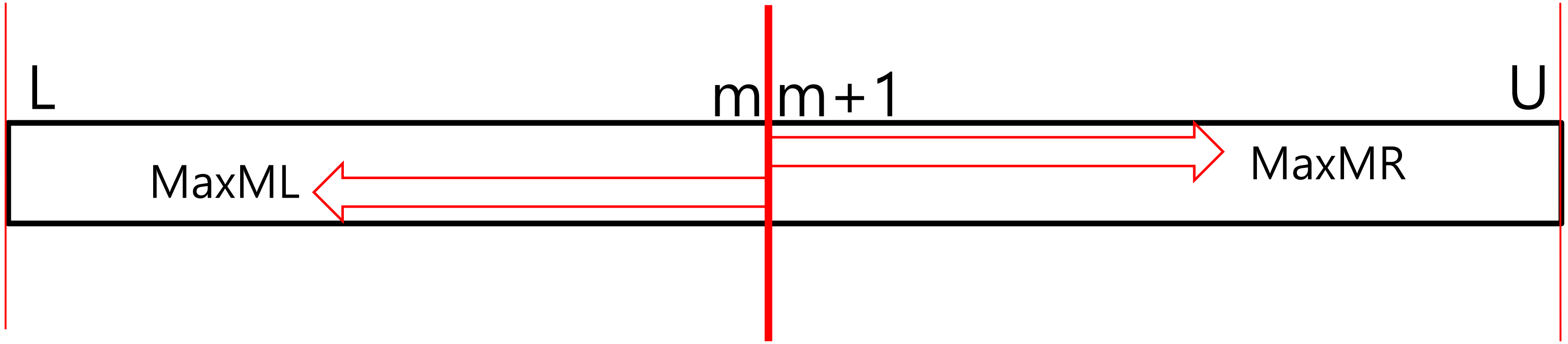



m = (L + U) / 2

MaxX = max(MaxL, MaxR, MaxM)

$X^*$ $XL^*$ $XR^*$ $XM^*$

## How to compute MaxM




MaxM = MaxML + MaxMR

MaxML = max(sum(L,m), sum(L+1,m), ….., sum(m-1,m), sum(m,m))

MaxMR = max(sum(m+1,m+1), sum(m+1,m+2), ….. , sum(m+1, U-1), sum(m+1,U))

How to compute MaxML and MaxMR ?

## How to compute MaxML

**MaxML = max(sum(L,m), sum(L+1,m), ..... Sum(m-1,m), sum(m,m))**

sum(i , m) = sum(i + 1, m) + X[i]

O(n) time

already computed

## How to compute MAXMR

**MaxMR = max(sum(m+1,m+1), sum(m+1,m+2), ..... sum(m+1,U))**

sum(m + 1, i) = sum(m + 1, i - 1) + X[i ]

O(n) time

already computed

# How to compute MaxML

**sum**(i , U) = **sum**(i+1, U) **+ X[i],** $L \leq i \leq U$

```
def comp_MaxML(L, U, X):
    sum = 0
    MaxML = 0
    for i in range(U – L + 1):
        sum = sum + X[U - i ]
        if sum > MaxML :
            MaxML = sum
    return MaxML
```

O(n) time

# How to compute MaxMR

**sum**(L, i) = **sum**(L, i-1) **+ X[i],** $L \leq i \leq U$

```
def comp_MaxMR(L, U, X):
    sum = 0
    MaxMR = 0
    for i in range(L, U+1):
        sum = sum + X[i ]
        if sum > MaxMR :
            MaxMR = sum
    return MaxMR
```

O(n) time

```
def max_sub(L, U, X):
    if L == U:
        return max(0, X[L])        basis case
    m = (L + U) / 2
    MaxL = max_sub(L, m, X)
    MaxR = max_sub(m+1, U, X)      Divide the problem
    MaxML = comp_MaxML(L, m, X)
    MaxMR = comp_MaxMR(m+1, U, X)  Combine solutions
    MaxM = max(0, MaxML+MaxMR)     O(n) per round
    return max(MaxL, MaxR, MaxM)
```

How many rounds of recursion ?

[ 31, -41, 59, 26, -53, 58, 97, -93, -23, 84]

[ 31, -41, 59, 26, -53][58, 97, -93, -23, 84]

[ 31, -41,59][ 26, -53][58, 97,-93,][ -23, 84]

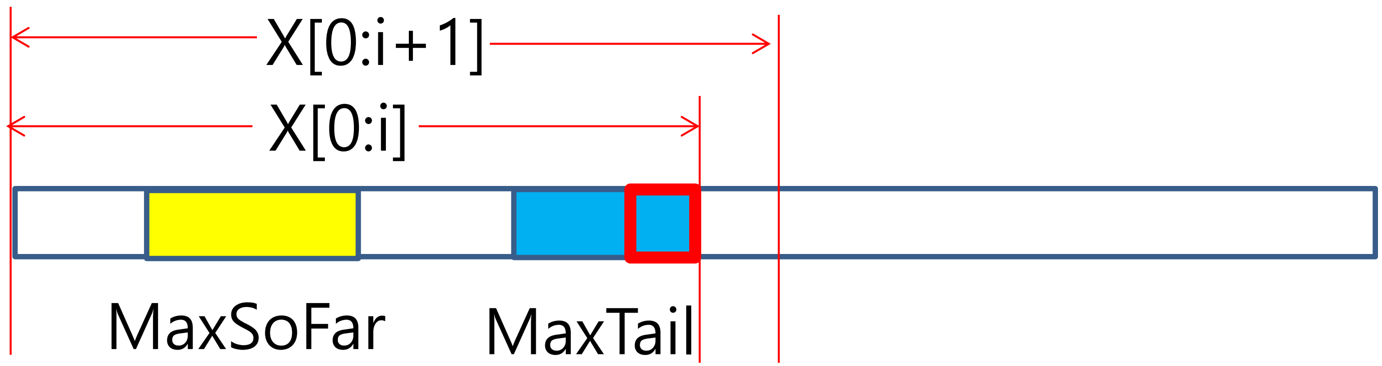
[ 31,-41][59][26][ -53][58, 97][-93][-23][ 84]

[31][-41]                    [58][97]

$n / 2^k = 1 \Longrightarrow k = \log_2 n$ rounds

$\therefore$ O(n $\log_2$ n) time

## Basic idea




MaxSoFar: The sum of the maximum subsequence in X[0 : i]
MaxTail : The sum of the maximum subsequence that ends at X[i-1]

Given MaxSoFar and MaxTail for X[0 : i], how can we find those for X[0 : i+1] ?

X = [ 31, -41, 59, 26, $\boxed{-53}$, 58, 97, -93, -23, 84]

For X[0 : 5],

MaxSoFar = 59 + 26 = 85

MaxTail = 59 + 26 + (-53) = 32

**What is MaxSoFar and MaxTail for X[0:6]?**

MaxTail = max((0, MaxTail+X[5])

= max(0, 32+58) =90

MaxSoFar = max(MaxSoFar, MaxTail) Why?

= max(85, 90) =90

O(1) time

# Recursive equations

$$\text{MaxTail[i]} = \begin{cases} \text{max(0, X[0]) if i = 0} \\ \text{max(0, MaxTail[i-1] + X[i]), otherwise} \end{cases}$$

$$\text{MaxSoFar[i]} = \begin{cases} \text{max(0, MaxTail[0]) if i =0} \\ \text{max(MaxSoFar[i-1], MaxTail[i]), otherwise} \end{cases}$$

```
def max_sub(X):
    MaxTail = 0
    MaxSoFar = 0
    for i in range(len(X)):
        MaxTail = max(0, MaxTail + X[i])
        MaxSoFar = max(MaxSoFar, MaxTail)
    return MaxSoFar
```

O(1) time

O(n) time

**Example:** X = [ 31, -41, 59, 26, -53, 58, 97, -93, -23, 84]

MaxTail = max(0, Maxtail + X[i])

MaxSoFar = max(MaxSoFar, MaxTail)

| i | MaxTail | MaxSoFar |
|---|---|---|
| 0 | 31 | 31 |
| 1 | 0 | 31 |
| 2 | 59 | 59 |
| 3 | 85 | 85 |
| 4 | 32 | 85 |
| 5 | 90 | 90 |
| 6 | 187 | 187 |
| 7 | 94 | 187 |
| 8 | 71 | 187 |
| 9 | 155 | 187 |