[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.09.15 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

- 본 강의에서 제작하여 제공하는 **PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료**의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.
- 본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

Algorithm Design Example

- **Maximum Subsequence Sum (MSS) Problem**

  Given N (possibly negative) integers $A_0, A_1, \cdots, A_{N-1}$, find the maximum value of $\sum_{k=i}^{j} A_k$ for $0 \le i \le j \le N-1$. (For convenience, the maximum subsequence sum is 0 if all the integers are negative.)

- Example
  - (-2, 11, -4, 13, -5, -2) → MSS = 20

Figure 2.2 Running times of several algorithms for maximum subsequence sum (in seconds)

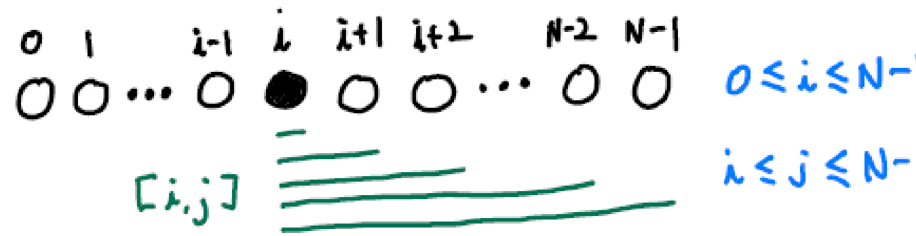
Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	N = 10	0.00103	0.00045	0.00066	0.00034
	N = 100	0.47015	0.01112	0.00486	0.00063
	N = 1,000	448.77	1.1233	0.05843	0.00333
	N = 10,000	NA	111.13	0.68631	0.03042
	N = 100,000	NA	NA	8.0113	0.29832

Max. Sum Subsequence versus **Max. Subsequence Sum**

Three Approaches for Max. Subsequence Sum Problem

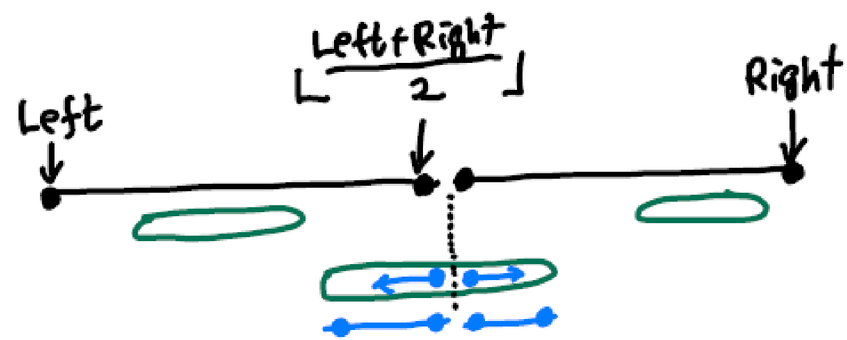
- Approach I: Simple Counting
  - Algorithms I & 2

$\boxed{O(N^3)}$ $\boxed{O(N^2)}$



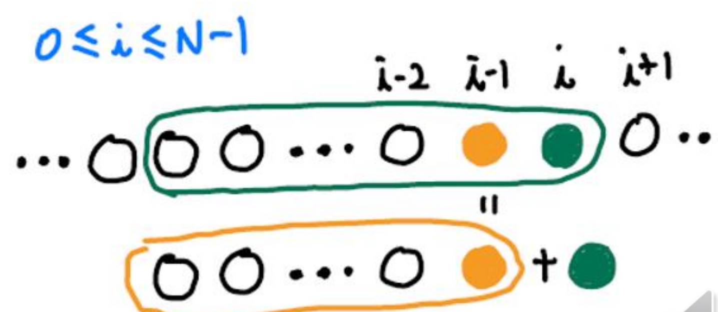
- Approach II: Divide and Conquer
  - Algorithm 3

$\boxed{O(N \log N)}$



- Approach III: Dynamic Programming
  - Algorithm 4

$\boxed{O(N)}$



Kadane's algorithm

Maximum Subsequence Sum: **Algorithm 4**

```
if (ThisSum < 0)
    ThisSum = 0;
else if (ThisSum > maxSum)
    MaxSum = ThisSum;
```

- Strategy
  - Use the **Dynamic Programming** strategy.
    - Idea

$B[i]$: the sum of a maximum subsequence that ends at index i

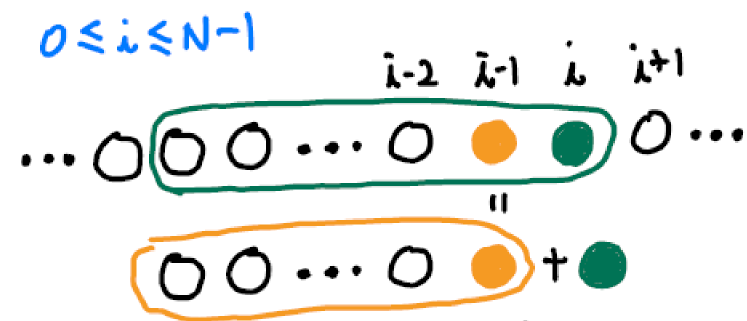
$$\longrightarrow B[i] = \max\{\, B[i-1] + A[i],\, 0 \,\}$$

```
        int
        MaxSubsequenceSum( const int A[ ], int N )
        {
            int ThisSum, MaxSum, j;

/* 1*/      ThisSum = MaxSum = 0;
/* 2*/      for( j = 0; j < N; j++ )
            {
/* 3*/          ThisSum += A[ j ];

/* 4*/          if( ThisSum > MaxSum )
/* 5*/              MaxSum = ThisSum;
/* 6*/          else if( ThisSum < 0 )
/* 7*/              ThisSum = 0;
            }
/* 8*/      return MaxSum;
        }
```

$\boxed{O(N)}$



optimal substructure
→ dynamic programming

- 만약에 sum이 음수라도 무방하고 1개 이상의 원소로 구성된 Subsequence (subarray)를 구하는 문제라면?

C Implementation

Maximum sum rectangle in a 2D matrix (DP-27) by GeeksforGeeks

```
int kadane(int* arr, int* start, int* finish, int n) {
    int sum = 0, maxSum = INT_MIN;
    *finish = -1;

    int local_start = 0;
    for (int i = 0; i < n; ++i) {
        sum += arr[i];
        if (sum < 0) {
            sum = 0; local_start = i+1;
        }
        else if (sum > maxSum) {
            maxSum = sum;
            *start = local_start; *finish = i;
        }
    }
    if (*finish != -1) return maxSum; // at least one non-negative number.

    // When all numbers in the array are negative
    maxSum = arr[0]; *start = *finish = 0;
    for (int i = 1; i < n; i++)  {
        if (arr[i] > maxSum) {
            maxSum = arr[i]; *start = *finish = i;
        }
    }
    return maxSum;
}
```

```
if (ThisSum < 0)
    ThisSum = 0;
else if (ThisSum > maxSum)
    MaxSum = ThisSum;
```

Empty subsequence를 허용하면 0을 리턴 (원래 문제)
Empty subsequence를 허용하지 않으면 음수 중 가장 큰 원소를 리턴

So, why do we bother with the time complexity?

Figure 2.2 Running times of several algorithms for maximum subsequence sum (in seconds)

Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	$N = 10$	0.00103	0.00045	0.00066	0.00034
	$N = 100$	0.47015	0.01112	0.00486	0.00063
	$N = 1{,}000$	448.77	1.1233	0.05843	0.00333
	$N = 10{,}000$	NA	111.13	0.68631	0.03042
	$N = 100{,}000$	NA	NA	8.0113	0.29832

Maximum Sum Subrectangle in 2D Array

- **Problem**
  - Given an mxn array of integers, find a subrectangle with the largest sum. (In this problem, we assume that a subrectangle is **any contiguous sub-array of size 1x1 or greater** located within the whole array.)

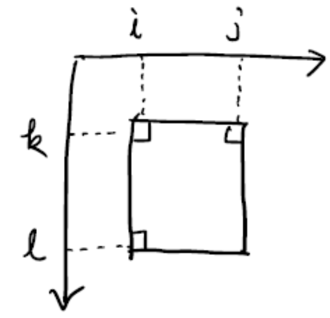
- **Note**
  - What is the **input size** of this problem? → **(m, n)**
    - If m = n → **n**
  - How many subrectangles are there in an mxn array?

  - For the case of m = n,
    - Design an $O(n^6)$ algorithm.
    - Design an $O(n^5)$ algorithm.
    - Design an $O(n^4)$ algorithm.
    - Design an $O(n^3)$ algorithm.

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

(i, j, k, l) such that $0 \le i \le j \le n-1$ and $0 \le k \le l \le m-1$

- How many subrectangles are there in an mxn array?



For an mxn rectangle,

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=0}^{m-1} \sum_{l=k}^{m-1} 1$$

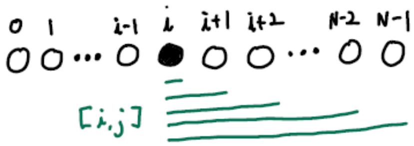
$$= \left(\sum_{k=0}^{m-1} \sum_{l=k}^{m-1} 1 \right) \left(\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1 \right) = \left\{ \sum_{k=0}^{m-1} (m-k) \right\} \left\{ \sum_{i=0}^{n-1} (n-i) \right\}$$

$$= \frac{m(m+1)}{2} \frac{n(n+1)}{2} = O(m^2 n^2)$$

$$= O(n^4) \text{ if } m = n$$

$$\uparrow \frac{1}{4} n^4$$

[1D case]



$0 \le i \le N-1$

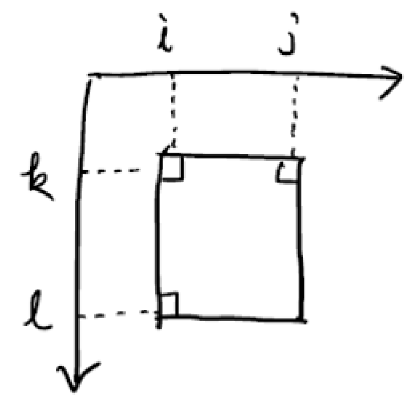
$i \le j \le N-1$

Maximum Sum Subrectangle: A Naïve Approach

- For each subrectangle, find its sum.

n = m 가정

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=0}^{n-1}\sum_{l=k}^{n-1}(j-i+1)(l-k+1)$$



$$=\underbrace{\left\{\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)\right\}}_{=A}\left\{\sum_{k=0}^{n-1}\sum_{l=k}^{n-1}(l-k+1)\right\}$$

$$A = 1\cdot n + 2(n-1) + 3(n-2) + \cdots + n\cdot 1$$

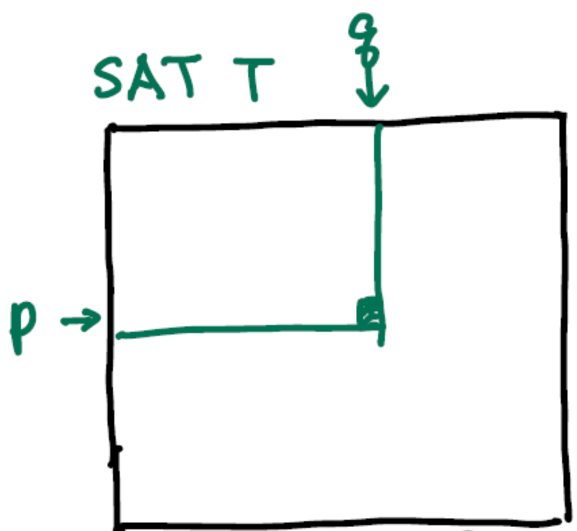
$$= \sum_{i=1}^{n} i(n-i+1) = \underbrace{n\sum_{i=1}^{n} i}_{\frac{1}{2}n^3} - \underbrace{\sum_{i=1}^{n} i^2}_{\frac{1}{3}n^3} + \sum_{i=1}^{n} i \approx \frac{1}{6}n^3$$

$$\Rightarrow O\left(\frac{1}{36}n^6\right)$$

$$\boxed{O(n^6)}$$

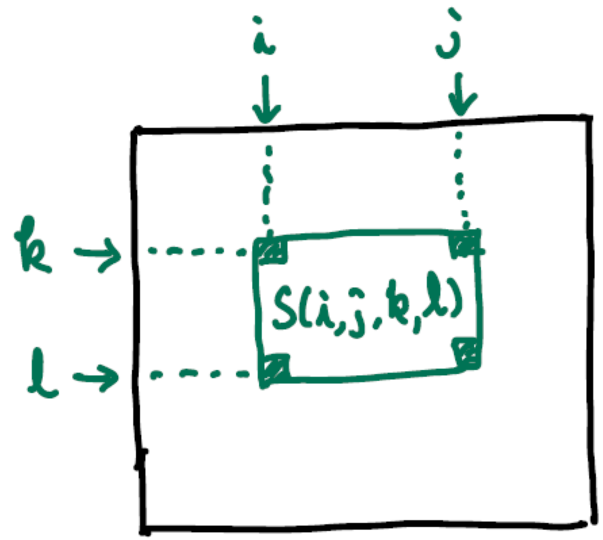
Maximum Sum Subrectangle: Summed Area Table

- **Table construction**: O(n²)
- **Sum comparisons**: O(n⁴)



$$T(p,q) = \sum_{i=0}^{p} \sum_{j=0}^{q} A(i,j)$$

$O(n^2)$



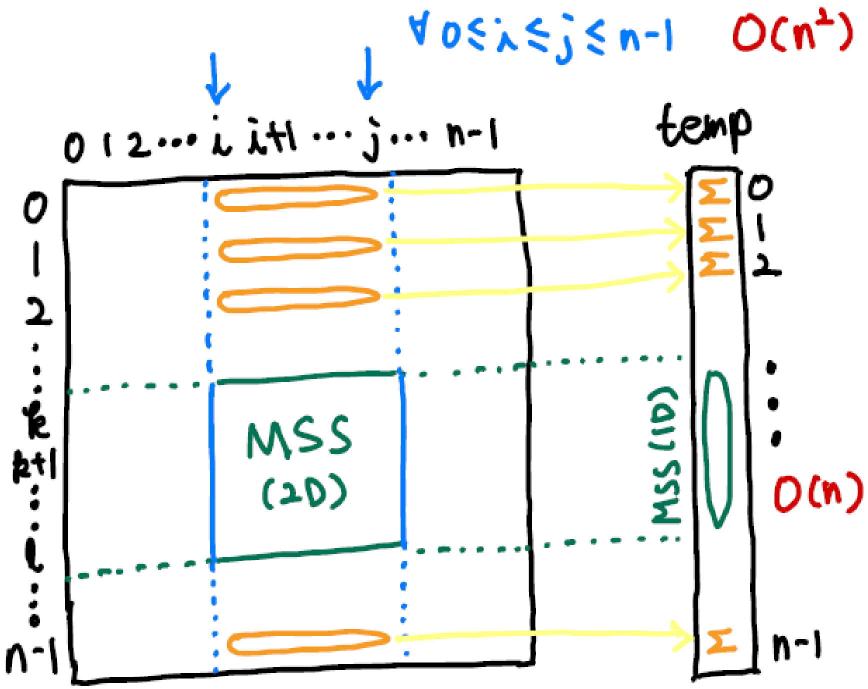
$$\forall\ 0 \le i \le j \le n-1,\quad 0 \le k \le l \le n-1 \ \leftarrow O(n^4)$$

$$S(i,j,k,l) = T(l,j) - T(k-1,j) - T(l,i-1) + T(k-1,i-1) \ \leftarrow O(1)$$

$\boxed{O(n^4)}$

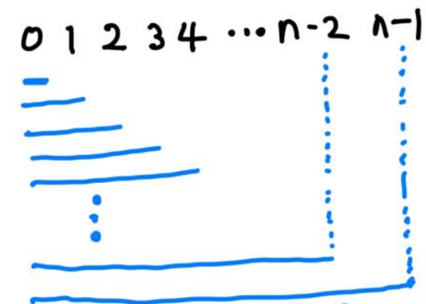
Maximum Sum Subrectangle: Kadane Algo.-Based

- **Idea**
  - MSS(2D)의 해당 열은 어디이건 i에서 j까지 임.
  - 가능한 모든 (i, j) 조합에 대하여 MSS(1D)를 Kadane 알고리즘을 사용하여 찾음.
    - 그렇게 하기 위하여, …



※ $n + (n-1) + (n-2) + \cdots + 1$

$= \frac{n(n+1)}{2} = O(n^2)$



각 구간마다 $O(n)$ 비용 필요

전체 비용 : $\sim n^2 \times n$

$\rightarrow O(n^3)$

0	-2	-7	0	-2
9	2	-6	2	11
-4	1	-4	1	-3
-1	8	0	-2	7

$\boxed{O(n^3) \text{ or } O(mn^2)}$

```c
void findMaxSum(int M[][COL]) {
    int maxSum = INT_MIN, finalLeft, finalRight, finalTop, finalBottom;
    int left, right, i;
    int temp[ROW], sum, start, finish;

    for (left = 0; left < COL; ++left) {
        memset(temp, 0, sizeof(temp));
        for (right = left; right < COL; ++right) {
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];
            sum = kadane(temp, &start, &finish, ROW);

            if (sum > maxSum) {
                maxSum = sum;
                finalLeft = left; finalRight = right;
                finalTop = start; finalBottom = finish;
            }
        }
    }
    printf("(Top, Left) (%d, %d)\n", finalTop, finalLeft);
    printf("(Bottom, Right) (%d, %d)\n", finalBottom, finalRight);
    printf("Max sum is: %d\n", maxSum);
}
```

C Implementation

$\boxed{O(n^3)}$

Maximum sum rectangle in a 2D matrix (DP-27) by GeeksforGeeks