3.4 Median (Select k-th)

- Median
  - The 50th percentile element of a list
  - Ex: Median of [45, 1, 10, 30, 25] → 25

- Generalized problem:
  - Select the k-th smallest element among a set of n unsorted elements $\{a_1, a_2, ..., a_n\}$.
- Key idea:
  - Use **partition ( )** in quick sort algorithm
    - The partition ( ) returns the position of the pivot → m
    - If k < m, then select k-th in the left subset
    - Else, find select (k-m)-th in the right subset

3.4 Median (Select k-th)

- Select (Divide & Conquer)

```
int select_kth ( int k, int s, int e )
{
    if ( s == e )
        return A[s];

    int m = partition ( s, e );

    if ( k == m )
        return A[k];

    if ( k < m )
       return select_kth ( k, s, m – 1 );
    else if ( k > m )
       return select_kth ( k-m, m + 1, e );
}
```

3.4 Median (Select k-th)

- Selection (Performance analysis)
  - Recurrence relation

$$T(n) = n + T(n/2)$$

3.4 Median (Select k-th)

- Comparison

	degenerate case	divide	conquer	combine	performance
tournament	n = 1 (s = e)	m = (s+e)/2	champ (s,m); champ (m+1,e);	win (LW, RW);	2T(n/2) + O(1) = O(n)
binary search	n = 1 (s = e)	m = (s+e)/2	bs (s, m-1); or bs (m+1, e);	-	T(n/2) + O(1) = O(log n)

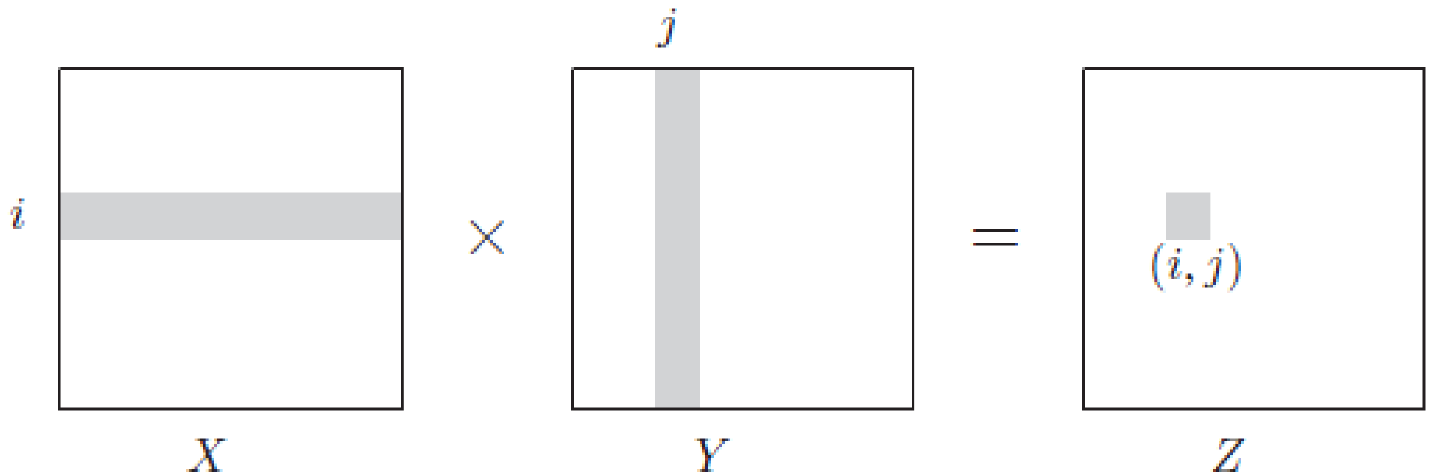
integer multiplication	n = 1	s = n/2	mult (w+x, y+z);
			mult (w, y);
			mult (x, z);
				p 2^n +
				(r – p – q) 2^s +
				q;
					3T(n/2) + O(n)
					= O($n^{log\_2\ 3}$)

merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)
quick sort	n = 1 (s>=e)	m = partition ( );	qs (s, m-1); qs (m+1, e);	-	2T(n/2) + O(n) = O(n log n)
median (find k-th)	n = 1 (s>=e)	m = partition ( );	select (k, s, m-1); or select(k-m, m+1, e);	-	T(n/2) + O(n) = O(n)
matrix multiplication					

3.5 Matrix Multiplication

- Multiplying two matrices: Z = XY
  - X & Y: n × n matrices
  - Time complexity: $O(n^3)$

$$Z_{i,j} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$



3.5 Matrix Multiplication

- Multiplying two matrices: Z = XY
  - X & Y: $n \times n$ matrices

```
void mat_mult( int **Z, int **X, int **Y, int n )
{
    int i, j, k;

    for ( i = 0; i < n; i++ )
        for ( j = 0; j < n; j++ )
            for ( k = 0, Z[i][j] = 0; k < n; k++ )
                Z[i][j] += X[i][k]*Y[k][j];
}
```

3.5 Matrix Multiplication

- Improve the performance using DnC
  - Divide X & Y into 2 × 2 groups

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \; Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

  - 8 (n/2) × (n/2) multiplications
  - Matrix addition → $O(n^2)$

3.5 Matrix Multiplication

- Performance analysis
  - 8 $(n/2) \times (n/2)$ multiplications → 8 $T(n/2)$
  - Matrix addition → $O(n^2)$

$$T(n) = 8T(n/2) + O(n^2)$$

$$= O(n^3)$$

  - No improvement !!

3.5 Matrix Multiplication

- Decomposing and assembling multiplications

$$P_1 = A(F - H)$$
$$P_2 = (A + B)H$$
$$P_3 = (C + D)E$$
$$P_4 = D(G - E)$$
$$P_5 = (A + D)(E + H)$$
$$P_6 = (B - D)(G + H)$$
$$P_7 = (A - C)(E + F)$$

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$
$$= \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

3.5 Matrix Multiplication

- Improvement of performance
  - 7 (n/2) × (n/2) multiplications → 7 T(n/2)
  - Matrix addition → O(n^2)

$$T(n) = 7T(n/2) + O(n^2)$$

$$= O(n^{\log_2 7})$$

$$\approx O(n^{2.81})$$

3.5 Matrix Multiplication

- Comparison

	degenerate case	divide	conquer	combine	performance
tournament	n = 1 (s = e)	m = (s+e)/2	champ (s,m); champ (m+1,e);	win (LW, RW);	2T(n/2) + O(1) = O(n)
binary search	n = 1 (s = e)	m = (s+e)/2	bs (s, m-1); or bs (m+1, e);	-	T(n/2) + O(1) = O(log n)

integer multiplication	n = 1	s = n/2	mult (w+x, y+z);
			mult (w, y);
			mult (x, z);
				p 2^n +
				(r – p – q) 2^s +
				q;
					3T(n/2) + O(n)
					= O($n^{log\_2 3}$)

merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)
quick sort	n = 1 (s>=e)	m = partition ( );	qs (s, m-1); qs (m+1, e);	-	2T(n/2) + O(n) = O(n log n)
median (find k-th)	n = 1 (s>=e)	m = partition ( );	select (k, s, m-1); or select(k-m, m+1, e);	-	T(n/2) + O(n) = O(n)
matrix multiplication	n = 1	X → A,B,C,D Y → E,F,G,H	P_1, ..., P_7	combine P_1 ~ P_7	7T(n/2)+O(n^2) = O($n^{log\_2 7}$)

퀴즈 5

- 다음 설명 중에서 올바른 것을 모두 고르시오.

(a) divide & conquer에 기반한 두 수의 곱셈과 두 행렬의 곱셈은 그 시간 복잡도가 같다.

(b) k번째로 작은 수를 구하는 divide & conquer 알고리즘은 combine 과정이 필요 없다.

(c) k번째로 작은 수를 구하는 divide & conquer 알고리즘의 시간 복잡도는 merge sort의 merge 연산의 시간 복잡도와 같다.

(d) 행렬 곱 연산의 bruteforce algorithm은 2중 for-loop로 구현된다.

3. Divide & Conquer

3.0 Introduction

3.1 Recurrence relation

3.2 Multiplication

3.3 Sorting

3.4 Medians

3.5 Matrix multiplication

Contents