알고리즘

3. Divide & Conquer

상명대학교 컴퓨터과학과

민 경 하

Contents

3. Divide & Conquer

3.0 Introduction

3.1 Recurrence relation

3.2 Multiplication

3.3 Sorting

3.4 Medians

3.5 Matrix multiplication

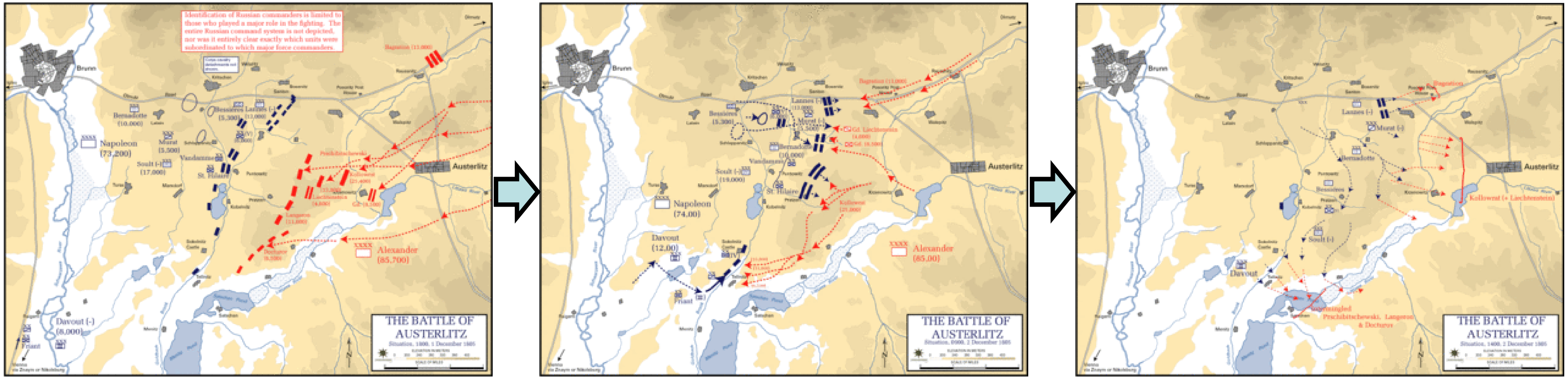
3.0 Introduction

(1) Battle of Austerlitz

3.0 Introduction

(1) Battle of Austerlitz (1805)

- Battle of three emperors
- France (65,000) VS Austria + Russia (80,000)
- The end of the 3rd anti-France aliance



(2) Tournament

- Ex) Elite-8 of Worldcup 2018
  - URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO
- Champion by league
  - How many games do they play?
- Champion by tournament
  - How many games do they play?

```
int tournament ( int n, int *team )
```

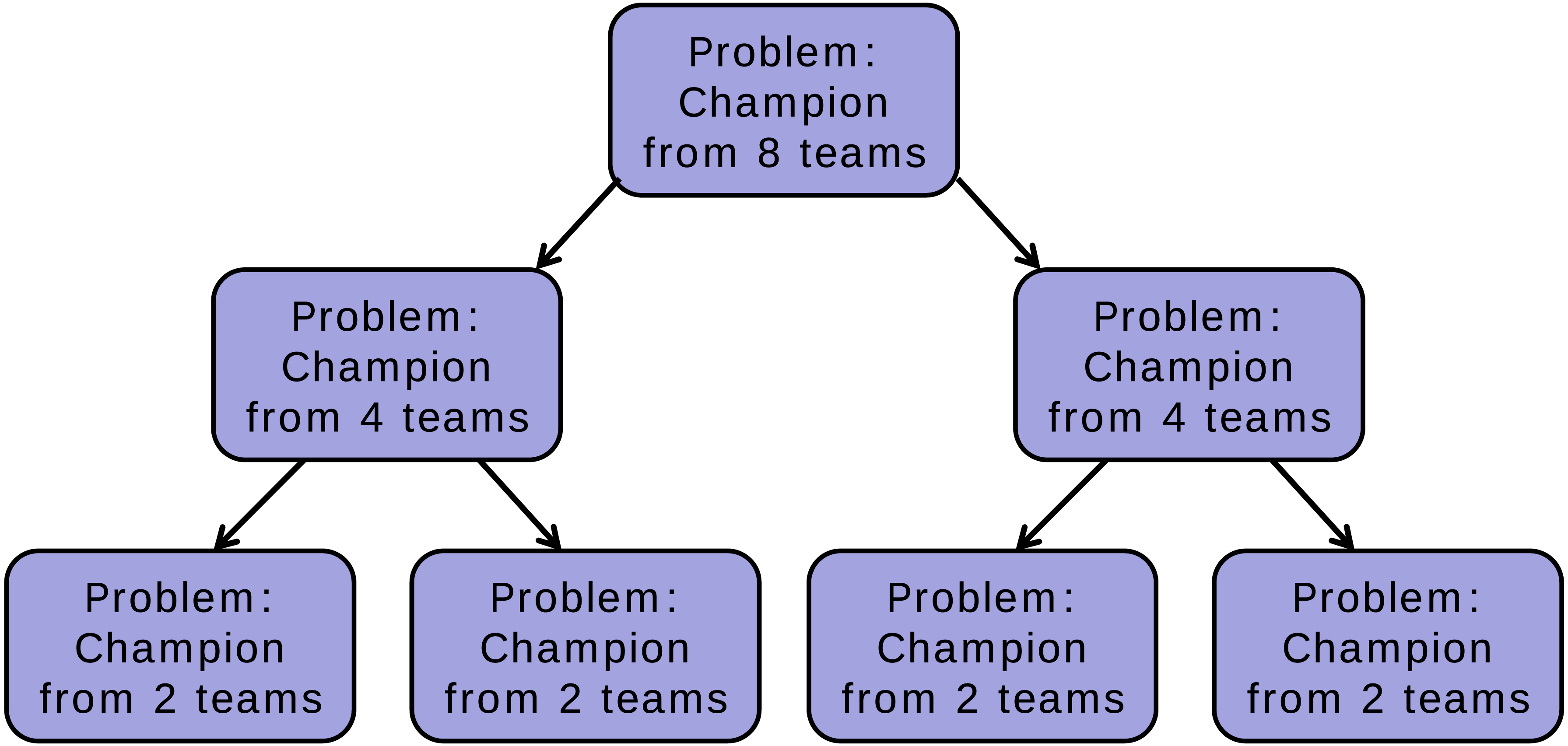
(2) Tournament

– Champion by tournament

```
int tournament ( int n, int *team )
{
    int i, j;
    for ( i = n; i > 1; i /= 2 ) {
        for ( j = 0; j < i; j += 2)
            win[j/2] = winner ( team[j], team[j+1] );
        copy ( team, win );
    }
    return win[0];
}
```

(2) Tournament

– Champion by tournament



(2) Tournament

– Champion by tournament

```
int Champion8 ( {URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO} )
{
    Lwinner = Champion4 ( {URG, FRA, BRA, BEL} );
    Rwinner = Champion4 ( {SWE, ENG, RUS, CRO} );

    return Winner ( Lwinner, Rwinner );
}
```

(2) Tournament

– Champion by tournament

```
int Champion4 ( {URG, FRA, BRA, BEL} )
{
    Lwinner = Champion2 ( {URG, FRA} );
    Rwinner = Champion2 ( {BRA, BEL} );

    return Winner ( Lwinner, Rwinner );
}
```

```
int Champion4 ( {SWE, ENG, RUS, CRO} )
{
    Lwinner = Champion2 ( {SWE, ENG} );
    Rwinner = Champion2 ( {RUS, CRO} );

    return Winner ( Lwinner, Rwinner );
}
```

(2) Tournament

– Champion by tournament

```
int Champion2 ( {URG, FRA} )
{
    Lwinner = Champion1 ( {URG} );      //     Unnecessary
    Rwinner = Champion1 ( {FRA} );      //     Unnecessary

    return Winner (URG, FRA);
}
```

```
int Champion2 ( {BRA, BEL} )
{
    Lwinner = Champion1 ( {BRA} );      //     Unnecessary
    Rwinner = Champion1 ( {BEL} );      //     Unnecessary

    return Winner (BRA, BEL);
}
```

(2) Tournament

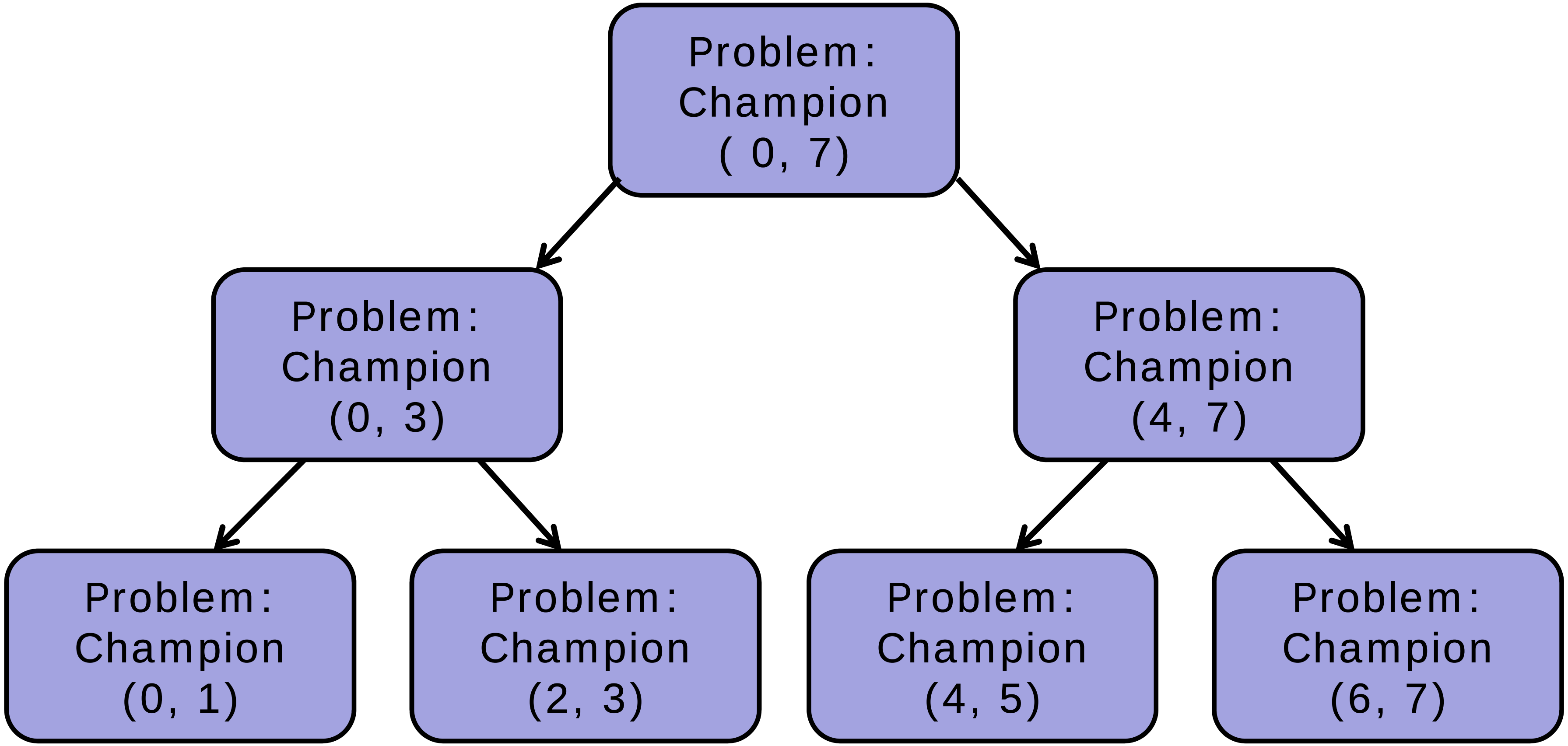
– Champion by tournament

```
char elit8[8] = {URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    int m = (s + e)/2;
    Lwinner = Champion ( s, m ); //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//    m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

(2) Tournament

- Champion by tournament



(2) Tournament

– Champion by tournament

```
char elit8[8] = {URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    int m = (s + e)/2;
    Lwinner = Champion ( s, m );  //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//     m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

– Do we miss something?

(2) Tournament

– Champion by tournament

```
char elit8[8] = {URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    if ( s == e )
        return s;

    int m = (s + e)/2;
    Lwinner = Champion ( s, m ); //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//    m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

– Do we miss something? → **degenerate case**

(2) Tournament

– Champion by tournament

```
char elit8[8] = {URG, FRA, BRA, BEL, SWE, ENG, RUS, CRO};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    if ( s == e )
        return s;

    int m = (s + e)/2;

    Lwinner = Champion ( s, m );
    Rwinner = Champion ( m+1, e );

    return Winner ( Lwinner, Rwinner );
}
```

Degenerate case

Divide

Conquer

Combine

(3) Key idea of divide & conquer

- Solve a problem of *n* inputs by splitting the input into *k* subsets

- Three steps of divide & conquer
  - Divide
    - Breaking a problem into subproblems
  - Conquer
    - Recursively solving these subproblems
  - Combine (optional)
    - Appropriately combining their answers

- 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					
integer multiplication					
merge sort					
quick sort					
median					
matrix multiplication					

(4) Abstract algorithm for DnC (recursive)

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

– Most divide & conquer algorithms are implemented using recursive call

(5) Three check points

- Same format

- Reduced problem size

- Degenerate case

- Same format

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

- Reduced problem size

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

  - $|q - p| \rightarrow |m - p| + |q - m - 1|$
  - $n \rightarrow n/2 + n/2$

- Degenerate case

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

- What is degenerate?
  - extraordinary
  - different from the ordinary cases

(6) Performance analysis for DnC

- T (n): time complexity of DnC ( ) for n inputs
- g (n): for small input
- f_1 (n): for DIVIDE ( )
- f_2 (n): for COMBINE ( )

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ 2T(n/2) + f_1(n) + f_2(n), & \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ aT(n/b) + O(n^d), & \text{otherwise} \end{cases}$$

(7) The simplest divide & conquer algorithm

- Binary search:
  - Let $A = \{a_1 , \ldots a_n\}$ be a list of elements which are sorted in nondecreasing order.
  - Determine whether x is in A or not. If x is in A, then find i such that $a_i = x$.

3.0 Introduction

- Bruteforce search (linear search)

```
int linear_search( int s, int e, int A[], int x )
{
    for ( i = s; i <= e; i++ ) {
        if ( A[i] == x )
            return i;
    }

    return NONE;
}
```

  – Performance analysis?

3.0 Introduction

- Binary search (Divide & Conquer)

```
int binary_search ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return binary_search ( s, m – 1, A, x );
    else
        return binary_search ( m + 1, e, A, x );
}
```

- Binary search → check three key points

```
int binary_search ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return binary_search ( s, m – 1, A, x );
    else
        return binary_search ( m + 1, e, A, x );
}
```

- Binary search (Divide & Conquer)
  - Performance analysis

$$T(n) = \begin{cases} 1, & \text{for } n = 1 \\ T(n/2) + 1, & \text{otherwise} \end{cases}$$

$$T(n) = \log n$$

- 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					
merge sort					
quick sort					
median					
matrix multiplication					

퀴즈 1

- 다음 중 divide & conquer에 대한 설명으로 적절하지 않은 것을 모두 고르시오.

  (a) Tournament 문제의 경우 divide & conquer를 이용하는 경우와 이용하지 않는 경우의 시간 복잡도가 같다.

  (b) Divide & conquer의 모든 문제는 divide, conquer, combine의 3 단계를 모두 수행해야 해결된다.

  (c) Divide & conquer에는 degenerate case를 고려하지 않아도 되는 문제가 있다.

  (d) Binary search의 시간 복잡도는 divide & conquer로 해결하는 Tournament 문제의 시간 복잡도와 같다.