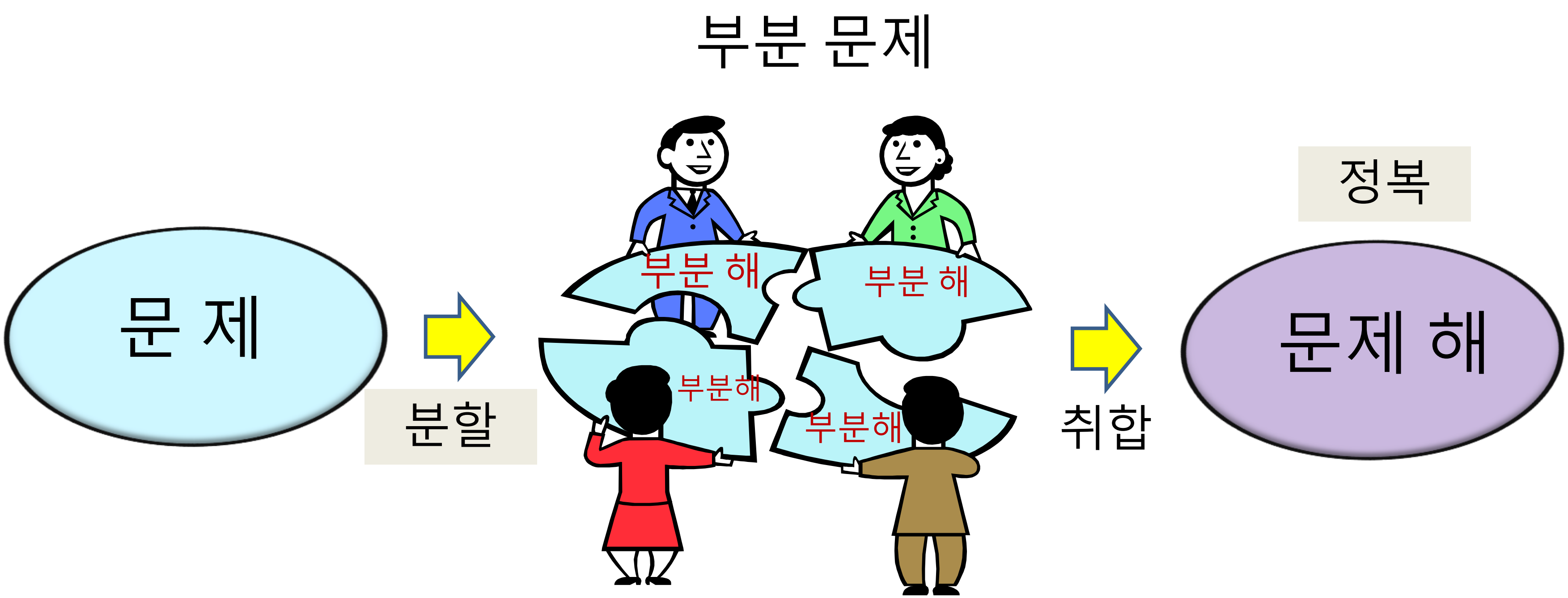
제3장 분할 정복 알고리즘

분할 정복 (Divide-and-Conquer) 알고리즘

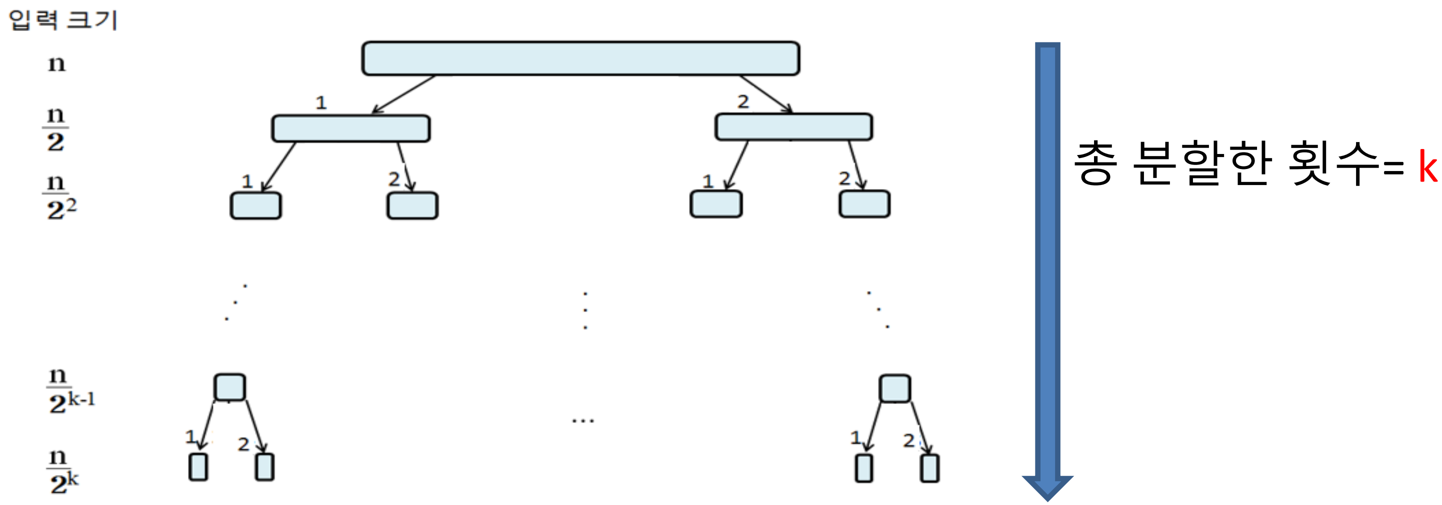
- 문제의 입력을 분할하여 문제를 해결 (정복)하는 방식의 알고리즘
  - 분할한 입력에 대하여 동일한 알고리즘을 적용하여 해를 계산하며, 이들의 해를 취합하여 원래 문제의 해를 얻는다.
  - 분할된 입력에 대한 문제를 부분문제 (subproblem)라고 하고, 부분 문제의 해를 부분해라고 한다.
  - 부분문제는 더 이상 분할할 수 없을 때까지 계속 분할한다.

하둡 → 분할 알고리즘



분할 과정

- 크기가 n인 입력을 2개로 분할
  - 분할된 각각의 부분 문제를 계속해서 2개로 분할해나가는 과정을 가정해보자

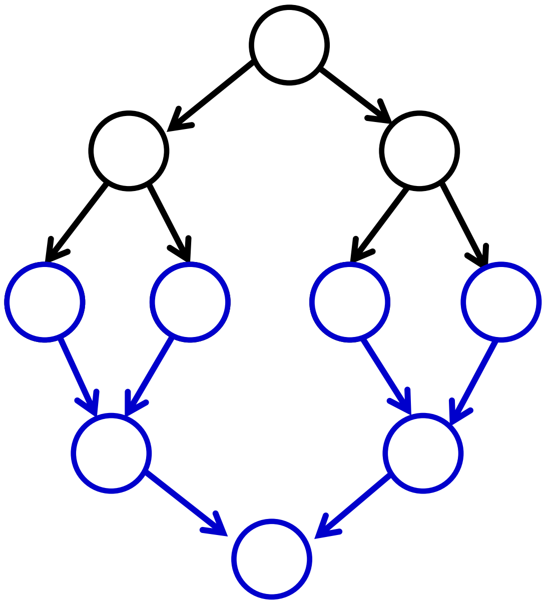


- 이슈(n과 k의 관계)
  - 입력 크기가 n일 때 총 몇 번 분할하여야 더 이상 분할할 수 없는 크기인 1이 될까?
  - k번 분할 후 각각의 입력 크기가 $n/2^k$
  - $n/2^k = 1$일 때 더 이상 분할할 수 없다.
  - $k = \log_2 n$이다.

정복 과정

- 대부분의 분할 정복 알고리즘은 문제의 입력을 단순히 분할만 해서는 해를 구할 수 없다.
  - 분할된 부분 문제들의 부분해를 찾는다.
  - 부분해를 취합하여 전체문제의 해를 구해 나간다
  - 이 것을 정복이라 함

분할 과정

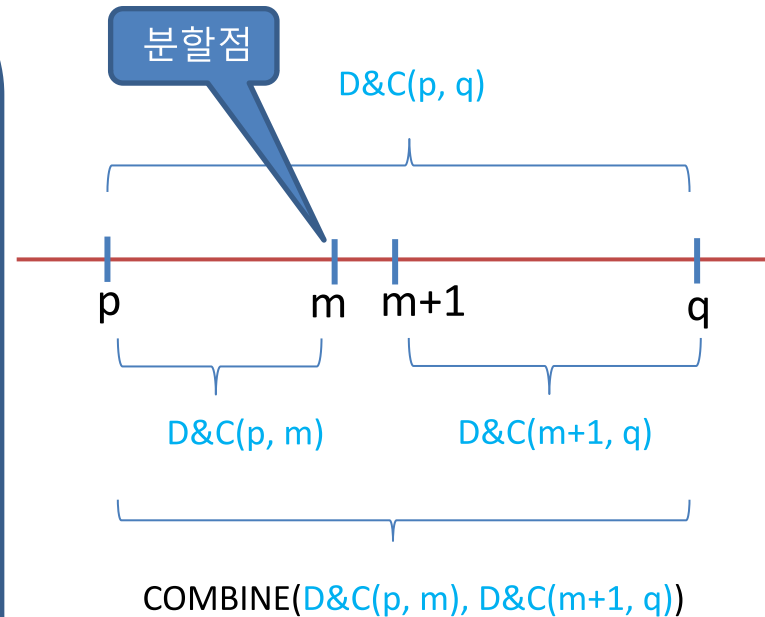


정복 (취합) 과정

Divide & Conquer Method (분할 정복 기법)

- 다음의 D&C(p, q)는 A[p], · · · , A[q] 사이의 문제를 해결하는 프로시져라 하자
- 알고리즘(분할정복기법에 대한 추상화)

```
D&C(p, q)
{
    int A[ ];
    int m, p, q, n; // 1≤ p ≤ q ≤ n
    {
            if (gain_sol(p, q))  // 더 이상 나누지 않고도 해를 구할 수 있는지
                                    여부를 체크하는 부울함수
                return(solution(p,q));
            else
            {
                m = DIVIDE(p, q) ;          // p ≤ m ≤ q
                return(COMBINE(D&C(p, m), D&C(m+1, q)))  //recursive call
            }
}
```



- D&C(1, n) 을 호출하여 시작됨

예제 : 최대값 최소값 찾기

- n개의 입력 데이터 중 최대값과 최소값을 찾는 전통적인 알고리즘
  - n개의 입력 데이터가 전역변수 A[1], · · · , A[n]에 저장되어 있다고 가정

```
Max_Min_Basic(int max, int min)
{
    int i, n;
    max = A[1];
    min = A[1];
    for(i=2; i<n; i++)
    {
            if (A[i] > max) max = A[i];
            if (A[i] < min) min = A[i];
    }
}
```

```
if (A[i] > max) max = A[i];
else if (A[i] < min) min = A[i];
```

Complexity (비교회수)

- Best Case(오름차순 정렬 되어 있을 때) : $2(n-1)$ → $n-1$
- Worst Case (내림차순 정렬 되어 있을 때) : $2(n-1)$ → $2(n-1)$
- Average Case : $2(n-1)$ → $(3n/2) - 1$

Divide & Conquer Method에 의한 최대최소 찾기

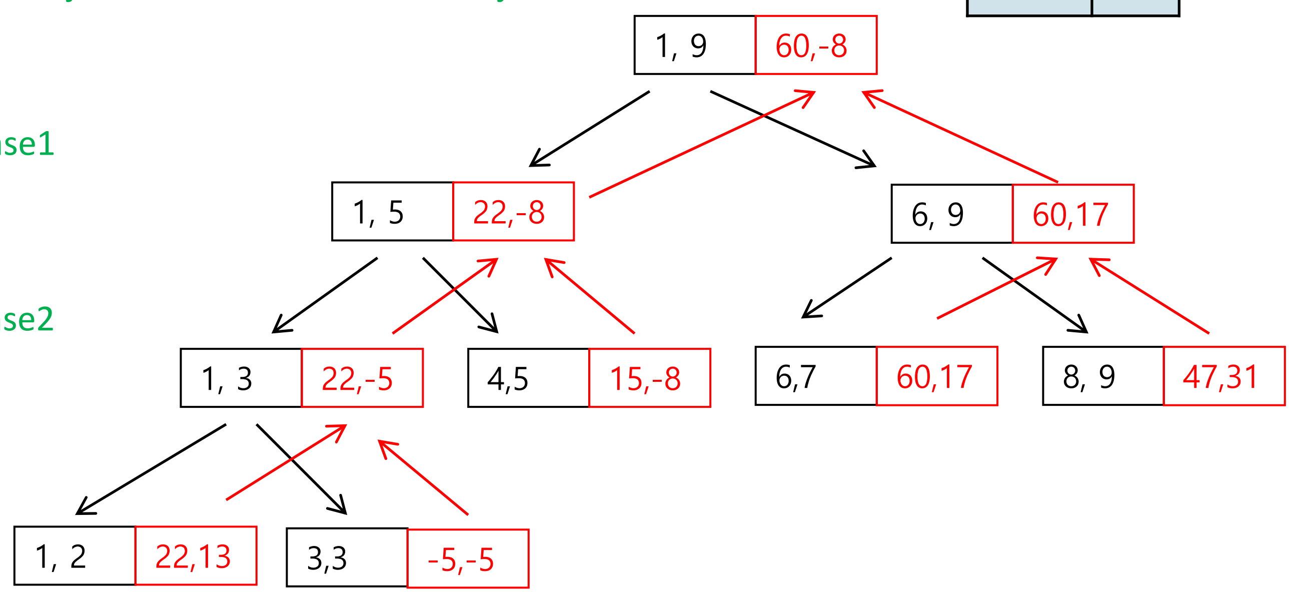
- mid값을 기준으로 두 그룹으로 분할 한 다음 두 그룹의 결과를 결합하여 결과 도출
  - 다음의 MAX_MIN함수는 A[i], · · · , A[j] 사이의 최대최소를 구하는 프로시져
  - max_min(1, n) 을 호출하여 시작됨
  - 알고리즘(분할정복기법으로 구성한 최대최소 찾기)

```
MAX_MIN(i, j, *fmax, *fmin)  // 인수 i와 j는 정수이며  범위는 1≤ i ≤ j ≤ n
{
    int , j, *fmax, *fmin;

①   if (i=j) {              // 분할 불가 case1
        *fmax = A[i];
        *fmin = A[i]; }

②   else if(i=j-1)  {       // 분할 불가 case2
            if  (A[i] < A[j]) {
                *fmax = A[j];
                *fmin = A[i];   }
            else {
                *fmax = A[i];
                *fmin = A[j];  }}
③   else  {
      mid = (i+j)/2;
      MAX_MIN(i, mid, &gmax, &gmin)          } 분할
      MAX_MIN(mid+1, j, &hmax, &hmin)        }
      fmax = max(&gmax, &hmax);              } 정복
      fmin = min(&gmin, &hmin): }            }
}
```

Max	60
Min	-8



A[i]	1	2	3	4	5	6	7	8	9
	22	13	-5	-8	15	60	17	31	47

Divide & Conquer Copmplexity 계산

- T(n)을 데이터끼리의 비교회수라 하자

$T(n)$:

```
MAX_MIN(i, j, *fmax, *fmin) {
     int , j, *fmax, *fmin;

     if (i=j) {
         *fmax = A[i];
         *fmin = A[i]; }

     else if(i=j-1)  {
                  if  (A[i] < A[j]) {
                      *fmax = A[j];
                      *fmin = A[i];   }
                  else {
                      *fmax = A[i];
                      *fmin = A[j];  }}
     else  {
       mid = (i+j)/2;
       MAX_MIN(i, mid, &gmax, &gmin)
       MAX_MIN(mid+1, j, &hmax, &hmin)
       fmax = max(&gmax, &hmax);
       fmin = min(&gmin, &hmin): }
}
```

+1

$T(\lfloor \frac{n}{2} \rfloor)$

$T(\lceil \frac{n}{2} \rceil)$

$+2$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 & n > 2 \quad (\text{+2는 } f_{max}, f_{min} \text{을 구하는비교회수}) \\ 1 & n = 2 \end{cases}$$

Divide & Conquer Copmplexity

- T(n)을 데이터끼리의 비교회수라 하자

$$T(n) = \begin{cases} T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + 2 & n > 2 \quad (+2\text{는 } f_{max}, f_{min} \text{을 구하는 비교회수}) \\ 1 & n = 2 \end{cases}$$

- n을 2의 거듭제곱이라 가정(n=2^k)

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 2 \\ &= 2(2T(\frac{n}{4}) + 2) + 2 \\ &= 2^2 T(\frac{n}{2^2}) + 2^2 + 2 \\ &\vdots \\ &= 2^{k-1}T(2) + \sum_{1 \le i \le k-1} 2^i \\ &= 2^{k-1} + 2^k - 2 \\ &= \frac{3}{2}n - 2 \end{aligned}$$

$\frac{n}{2^{k-1}} = \frac{2^k}{2^{k-1}} = 2$

$\leadsto 2^1 + 2^2 + \cdots 2^{k-1} = \frac{2(2^{k-1}-1)}{2-1} = 2(2^{k-1}-1) = 2 \cdot 2^{k-1} - 2$

$\Rightarrow$ 복잡도 : $O(n)$

- 여기서 T(n) Best, Worst, Average Case 모두에 해당
- Max_Min_Basic의 2n – 2에 비하여 작음을 알 수 있다
- 그러나 순환적 방법은 추가적인 stack 공간 등 오버헤드가 발생하기 때문에 반드시 좋다고 볼 수는 없다.
- 반복에 의한 방법과 비교해봐야 함.

Divide & Conquer Complexity (index 비교까지 포함)

- (A[i],A[j]) 비교와 (i,j) 비교 시간이 같다고 가정하자
- index 비교까지 포함한 비교회수를 C(n)을 라 하자
- 알고리즘에서의 비교는 i≥j-1 하나로 대체될 수 있다. 따라서

$$C(n) = \begin{cases} 2C(\frac{n}{2}) + 3, & n > 2 \\ 2, & n = 2 \end{cases}$$

- n을 2의 거듭제곱이라하고 임의의 자연수 k에 대하여 $n=2^k$라고 하면

$$\begin{aligned} C(n) &= 2C(\frac{n}{2}) + 3 \\ &= 4C(\frac{n}{4}) + 6 + 3 \\ &\vdots \\ &= 2^{k-1}C(2) + 3\sum_{i=0}^{k-2} 2^i \\ &= 2^k + 3 \times 2^{k-1} - 2 \\ &= \frac{5}{2}n - 3 \end{aligned}$$

- 여기서 C(n) Best, Worst, Average Case 모두에 해당
- Max_Min_Basic의 3n – 3에 비하여 작음을 알 수 있다

반복과 순환의 비교 1

- n!을 구하는 문제를 생각해보자

$$n! = \begin{cases} 1 & n = 1 \\ n * (n-1)! & n \geq 2 \end{cases}$$

- 순환적(재귀적, recursive) 방법

```
int factorial(int n)
{
    if( n == 1 ) return(1);
    else return (n * factorial(n-1 );
}
```

T(n) = T(n-1) + 1

주요연산 : 곱셈

O(n)

- 반복적(iterative) 방법

```
int factorial(int n)
{
     int i;
     long result = 1;

     for(i = 1; i <= n; i++)
          result *= i;
     return result;
}
```

T(n) = T(n-1) + 1

주요연산 : 곱셈

O(n)

반복과 순환의 비교 2

- 피보나치 수열

$$f(n) = \begin{cases} (f(n-1) + f(n-2) & n > 2 \\ 1 & n = 1 \\ 0 & n = 0 \end{cases}$$

- 순환적(재귀적, recursive) 방법

```
int fibo(int n)
{
    if( n == 0 )
          return(0)
    if( n == 1 )
        return(1)
    else return fibo(n-1) + fibo(n-2)

}
```

$O(2^n)$

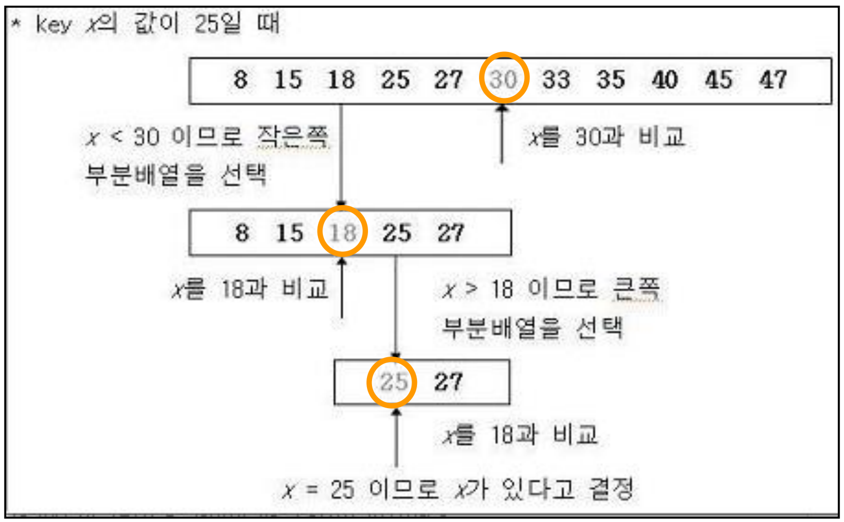
- 반복적(iterative) 방법

```
int fibo(int n)
{
     f[0]=0; f[1]=1;
     for(i = 2; i <= n; i++)
          f[i]=f[i-1] + f[i-2];
      return f[n];
}
```

$O(n)$

Divide & Conquer Method (사례)

- Binary Search



- Complexity를 구해보자

중앙값과 목표값과의 비교

$$T(n) = T\left(\frac{n}{2}\right) + 1\,, n > 2$$
$$= 1, \qquad n=1$$

$$T(n) = T\left(\frac{n}{2^k}\right) + k$$

$n = 2^k$ 라고 하면
$T(n) = T(1) + k = k+1$

$k = \log_2 n$이므로
$T(n) = \log_2 n + 1$

분할 정복 알고리즘의 분류

- 문제가 a개로 분할되고, 부분 문제의 크기가 1/b로 감소하는 알고리즘:

  - a=b=2인 경우 합병 정렬 (3.1절), 최근접 점의 쌍 찾기 (3.4절), 공제선 문제 (연습문제 43)

  - a=3, b=2 큰 정수의 곱셈 (연습문제 26)

  - a=4, b=2 큰 정수의 곱셈 (연습문제 25)

  - a=7, b=2인 경우, 스트라센(Strassen)의 행렬 곱셈 알고리즘 (연습문제 27)

- 문제가 2개로 분할, 부분문제의 크기가 일정하지 않은 크기로 감소하는 알고리즘: 퀵 정렬 (3.2절)

- 문제가 2개로 분할, 그 중에 1개의 부분 문제는 고려할 필요 없으며, 부분 문제의 크기가 1/2로 감소하는 알고리즘: 이진탐색 (1.2절)

- 문제가 2개로 분할, 그 중에 1개의 부분문제는 고려할 필요 없으며, 부분문제의 크기가 일정하지 않은 크기로 감소하는 알고리즘: 선택 문제 알고리즘 (3.3절)

- 부분문제의 크기가 1, 2개씩 감소하는 알고리즘: 삽입 정렬 (6.3절), 피보나치 수 (3.5절) 등

3.1 합병 정렬

- 합병 정렬 (Merge Sort)은 입력을 2개의 부분 문제로 분할
  - 부분문제의 크기가 1/2로 감소하는 분할 정복 알고리즘
- n개의 숫자들을 n/2개씩 2개의 부분문제로 분할
  - 각각의 부분문제를 재귀적으로 합병 정렬한 후,
  - 2개의 정렬된 부분을 합병하여 정렬 (정복)한다.
- 합병 과정이 (문제를) 정복하는 과정임
  - 2개의 각각 정렬된 숫자들을 1개의 정렬된 숫자들로 합치는 것

배열 A: 6 14 18 20 29
배열 B: 1 2 15 25 30 45
⇨ 합병배열 C: 1 2 6 14 15 18 20 25 29 30 45

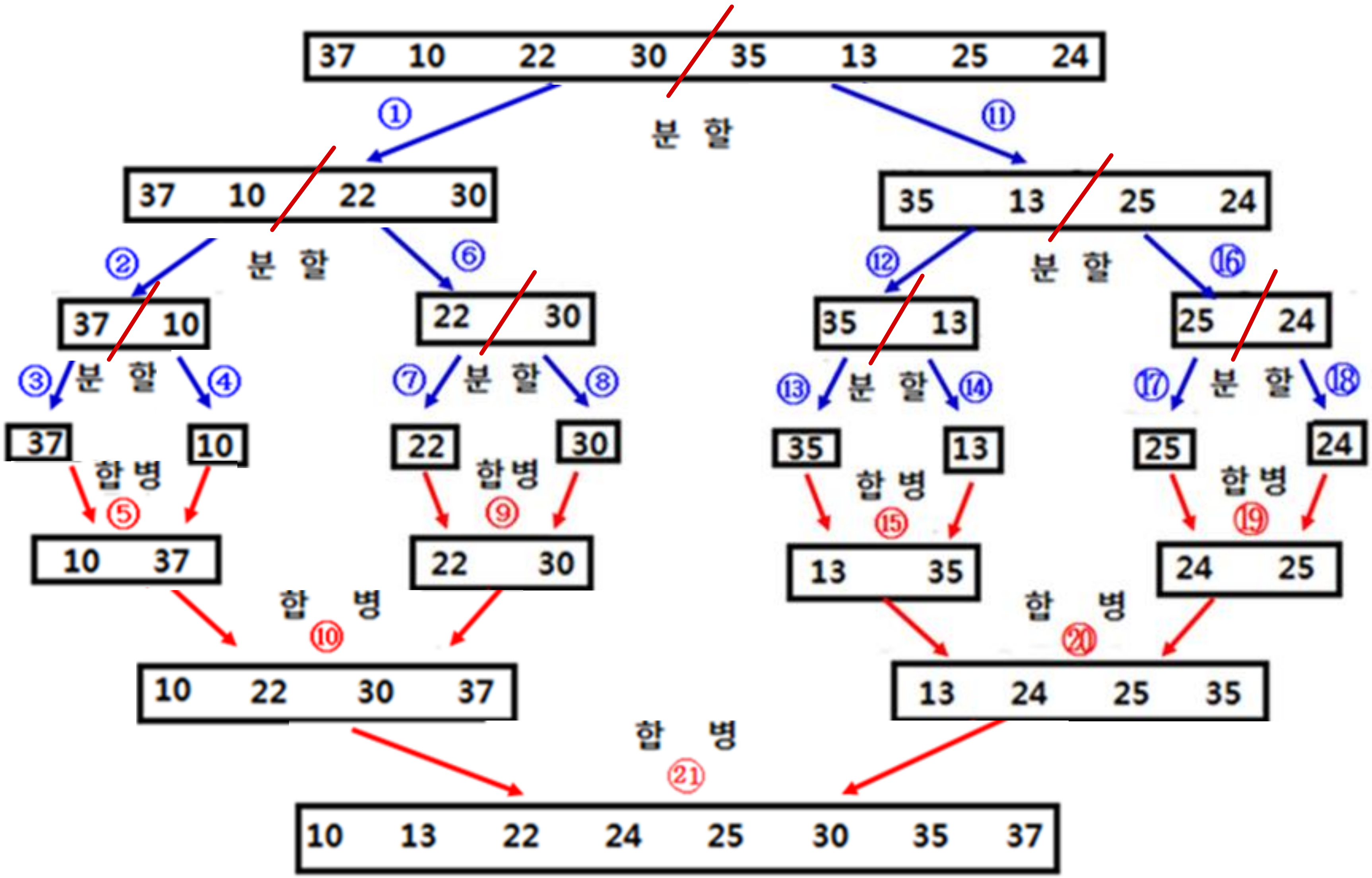
합병 정렬 알고리즘

입력: A[p]~A[q]

출력: 정렬된 A[p]~A[q]

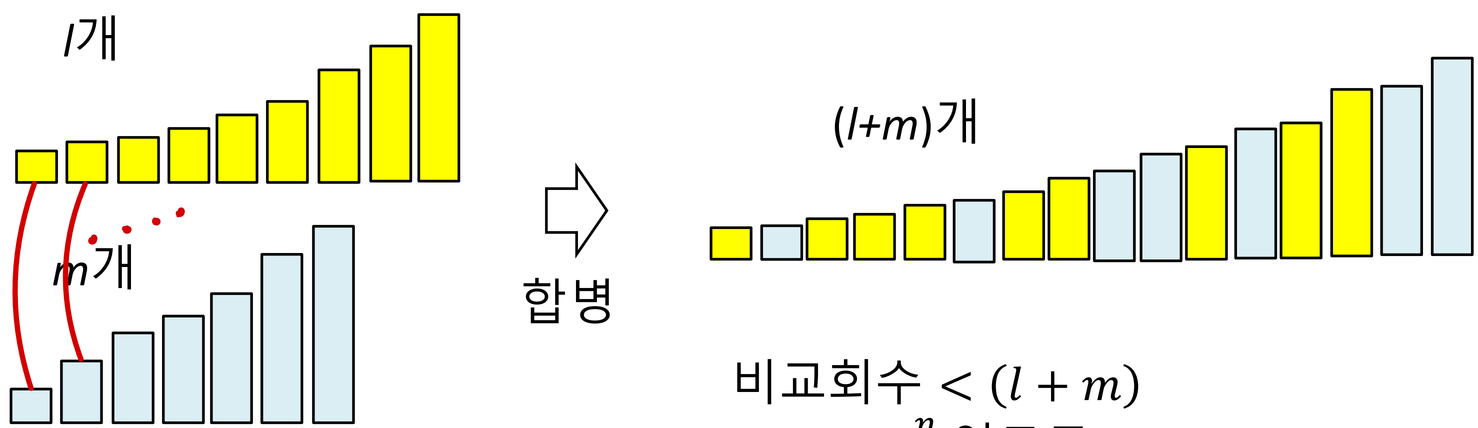
```
MergeSort(A,p,q) {
        if ( p > q ) {   // 배열의 원소의 수가 2개 이상이면
                k = ⌊(p+q)/2⌋   // k=반으로 나누기 위한 중간 원소의 인덱스
                MergeSort(A,p,k)       // 앞부분 재귀 호출
                MergeSort(A,k+1,q)    // 뒷부분 재귀 호출
                Merge(A,p,q)            // 합병.
        }
}
```

- 입력 크기가 n=8인 배열 A=[37, 10, 22, 30, 35, 13, 25, 24]



시간복잡도

- 분할하는 부분은 배열의 중간 인덱스 계산과 2번의 재귀 호출
  - 데이터 크기에 무관하므로 O(1) 시간 소요
- 합병의 수행 시간은 입력의 크기에 비례.
  - 2개의 정렬된 배열 A와 B의 크기가 각각 l과 m이라면, 최대 비교 횟수= (l+m-1).

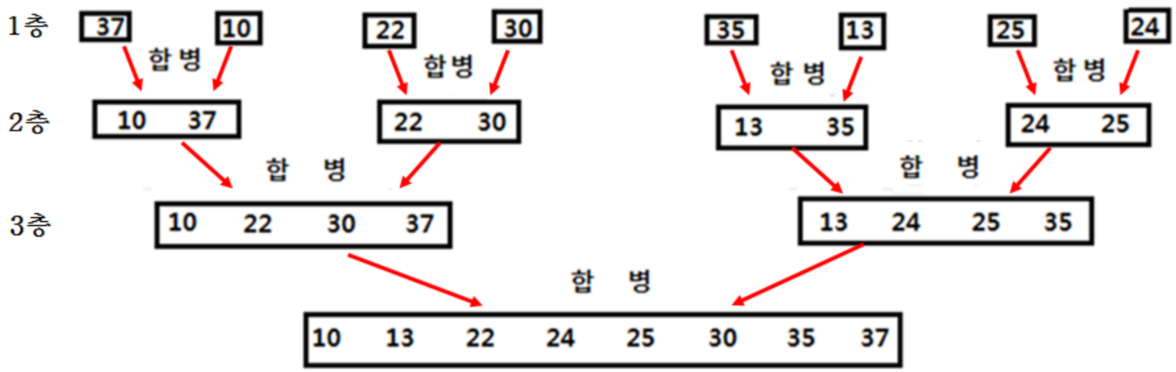


비교회수 $< (l + m)$

$l \cong m \cong \frac{n}{2}$ 이므로

비교회수 < n

- 따라서 n개의 데이터에 대한 이원합병의 수행 시간은 O(n)
- 이원합병은 몇 번 수행할까?



- 각 층을 살펴보면 모든 숫자(즉, n=8개의 숫자)가 합병에 참여
- 합병은 입력 크기에 비례하므로 각 층에서 수행된 비교 횟수는 O(n)

- 층수는?
- 층수를 세어보면, 8개의 숫자를 반으로, 반의 반으로, 반의 반의 반으로 나눈다.
- 이 과정을 통하여 3층이 만들어진다.

입력 크기	예	층
n	8	
n/2	4	1층
n/4 = $n/2^2$	2	2층
n/8 = $n/2^3$	1	3층

- 입력의 크기가 n일 때 몇 개의 층이 만들어질까?
- n을 계속하여 1/2로 나누다가, 더 이상 나눌 수 없는 크기인 1이 될 때 분할을 중단
- 따라서 k번 1/2로 분할했으면 k개의 층이 생김
- $n=2^k$이라 하면 층수 k= $\log_2 n$
- 합병 정렬의 시간복잡도:

(층수) x O(n) = $\log_2 n$ x O(n) = $O(n\log_2 n)$

시간복잡도의 또 다른 유도과정

$$
\begin{aligned}
T(n) &\le 2T(\frac{n}{2}) + \mathrm{O}(n) \\
&\le 2(2T(\frac{n}{4}) + \frac{n}{2}) + \mathrm{O}(n) = 2^2 T(\frac{n}{2^2}) + 2\mathrm{O}(n) \\
&\le 2^2(2T(\frac{n}{2^3}) + \frac{n}{2^2}) + 2\mathrm{O}(n) = 2^3 T(\frac{n}{2^3}) + 3\mathrm{O}(n) \\
&\cdots \\
&\le 2^k T(\frac{n}{2^k}) + k\,\mathrm{O}(n) = nT(1) + k\,\mathrm{O}(n) = nT(1) + \log n \cdot \mathrm{O}(n) \\
&= nT(1) + \mathrm{O}(kn) = n + \mathrm{O}(n \log n) \\
&\le \mathrm{O}(n \log n)
\end{aligned}
$$

k= $\log_2 n$

- 합병 정렬의 단점
  - 합병 정렬의 공간 복잡도: O(n) 이지만
    - 입력을 위한 메모리 공간 (입력 배열)외에 추가로 입력과 같은 크기의 공간 (임시 배열)이 별도로 필요.
    - 2개의 정렬된 부분을 하나로 합병하기 위해, 합병된 결과를 저장할 곳이 필요하기 때문
- 합병 정렬의 응용
  - 외부정렬의 기본이 되는 정렬 알고리즘
  - 연결 리스트에 있는 데이터를 정렬할 때
    - 합병 정렬
  - 배열에 있는 데이터를 정렬할 때
    - 퀵 정렬이나 힙 정렬
  - 하둡과 같은 분산처리 시스템에서 합병 정렬 알고리즘이 활용

C++ 프로그램의 예

```
public int[] sort() {
        this.mergeSort(a, 0, a.length - 1);

        return a;
}

void mergeSort(int[] a, int x, int y) {
        if (y - x > 0) {
            int middle = (x + y) / 2;
            this.mergeSort(a, x, middle);
            this.mergeSort(a, middle + 1, y);
            this.merge(a, x, y);
        }
}
```

```
void merge(int[] a, int start, int end) {
        int[] tmp = new int[a.length];
        int tmpIndex = start;

        int middle = (start + end) / 2;

        int s = start, e = middle + 1;
        while ((s <= middle) && (e <= end)) {
            int sNum = a[s];
            int eNum = a[e];

            if (sNum < eNum) {
                    tmp[tmpIndex++] = a[s++];
            } else {
                    tmp[tmpIndex++] = a[e++];
            }

        }

        while (s <= middle) {
            tmp[tmpIndex++] = a[s++];
        }

        while (e <= end) {
            tmp[tmpIndex++] = a[e++];
        }

        this.copySrcToDst(tmp, a, start, end);
}
```

Divide & Conquer Method 에서 자주 등장하는 복잡도의 예

점화식	복잡도	알고리즘 사례
$T(n) = T\,(n-1) + c$	$O(n)$	Factorial계산
$T(n) = T\,\left(\frac{n}{2}\right) + c$	$\mathrm{O}(\log_2 n\,)$	이진탐색
$T(n) = 2T\,\left(\frac{n}{2}\right) + c$	$O(n)$	최대최소구하기
$T(n) = 2T\,\left(\frac{n}{2}\right) + cn$	$\mathrm{O}(n\log_2 n\;)$	합병정렬, 퀵정렬
$T(n) = T\,(n-1) + cn$	$O(n^2)$	선택정렬
$T(n) = cT\,\left(\frac{n}{m}\right) + c$	$\mathrm{O}(\,\log_m c\;)$	스트라센 행렬곱
$T(n) = nT\,(n-1) + c$	$O(n!)$	역행렬

행렬의 곱셈

값이 같아야 함

정의 ❷-5 $A=[a_{ij}]$가 $m\times n$ 행렬이고, $B=[b_{ij}]$가 $n\times p$ 크기의 행렬일 때 행렬 A와 B의 행렬의 곱(multiplication)은 $AB=C=[c_{ij}]$로써 다음과 같이 정의되는 $m\times p$ 행렬이 된다.

$$c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}b_{nj}$$

$$i=1,2,\cdots,m$$

$$j=1,2,\cdots,p$$

$n\times p$

$$\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2q} \\ \vdots & \vdots & & \vdots & & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{nq} \end{bmatrix}$$

$$AB=\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_{m\times n}\underbrace{\begin{bmatrix} c_{11} & c_{12} & & c_{1p} \\ c_{21} & c_{22} & & c_{2p} \\ & & & \\ & & c_{ij} & \\ & & & \\ c_{m1} & c_{m2} & & c_{mp} \end{bmatrix}}_{m\times p}=C$$

mnp번의 곱셈이 필요함

두 $n\times n$행렬의 경우 n^3

행렬의 곱셈의 복잡도는 $O(n^3)$

행렬곱셈의 분할연산

- 다음과 같은 2 x 2 크기의 A,B 행렬이 있다고 가정하자.
  - 이 때, 행렬의 곱은 다음과 같이 나타낼 수 있음

$$AB = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

- 만약 위 행렬의 각 원소인 a_{ij}, b_{ij}가 스칼라 값이 아니라 행렬이라면 어떻게 될까?

$$AB = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \cdot \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \hline A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{array}\right]$$

행렬곱셈 분할연산의 예 (1)

$$A = \left[\begin{array}{cc|cc} 1 & -1 & 3 & 1 \\ 2 & 1 & 2 & 1 \\ \hline 0 & 5 & 3 & 1 \\ -1 & 2 & 1 & 1 \end{array}\right] \qquad B = \left[\begin{array}{cc|cc} 1 & 0 & 2 & 1 \\ 2 & -1 & 3 & 3 \\ \hline 5 & 1 & 2 & -1 \\ 0 & 2 & 1 & 0 \end{array}\right]$$

$$A_{11} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix} \quad A_{12} = \begin{bmatrix} 3 & 1 \\ 2 & 1 \end{bmatrix} \qquad B_{11} = \begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} \quad B_{12} = \begin{bmatrix} 2 & 1 \\ 3 & 3 \end{bmatrix}$$

$$A_{21} = \begin{bmatrix} 0 & 5 \\ -1 & 2 \end{bmatrix} \quad A_{22} = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \qquad B_{21} = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix} \quad B_{22} = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}$$

$$A_{11}B_{11} + A_{12}B_{21} = \begin{bmatrix} 14 & 6 \\ 14 & 3 \end{bmatrix} \qquad A_{11}B_{21} + A_{12}B_{22} = \begin{bmatrix} 6 & -5 \\ 12 & 3 \end{bmatrix}$$

$$A_{21}B_{11} + A_{22}B_{21} = \begin{bmatrix} 25 & 0 \\ 8 & 1 \end{bmatrix} \qquad A_{21}B_{12} + A_{22}B_{22} = \begin{bmatrix} 22 & 12 \\ 7 & 4 \end{bmatrix}$$

$$AB = \begin{bmatrix} 14 & 6 & 6 & -5 \\ 14 & 3 & 12 & 3 \\ 25 & 0 & 22 & 12 \\ 8 & 1 & 7 & 4 \end{bmatrix}$$

행렬곱셈 분할연산의 예 (2)

$$A = \left[\begin{array}{ccc|cc} 2 & -3 & 1 & 0 & -4 \\ 1 & 5 & -2 & 3 & -1 \\ \hline 0 & -4 & -2 & 7 & -1 \end{array}\right] = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \left[\begin{array}{cc} 6 & 4 \\ -2 & 1 \\ -3 & 7 \\ \hline -1 & 3 \\ 5 & 2 \end{array}\right] = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} A_{11}B_1 + A_{12}B_2 \\ A_{21}B_1 + A_{22}B_2 \end{bmatrix}$$

$$A_{11}B_1 = \begin{bmatrix} 2 & -3 & 1 \\ 1 & 5 & -2 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ -2 & 1 \\ -3 & 7 \end{bmatrix} = \begin{bmatrix} 15 & 12 \\ 2 & -5 \end{bmatrix}$$

$$A_{12}B_2 = \begin{bmatrix} 0 & -4 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} -1 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} -20 & -8 \\ -8 & 7 \end{bmatrix}$$

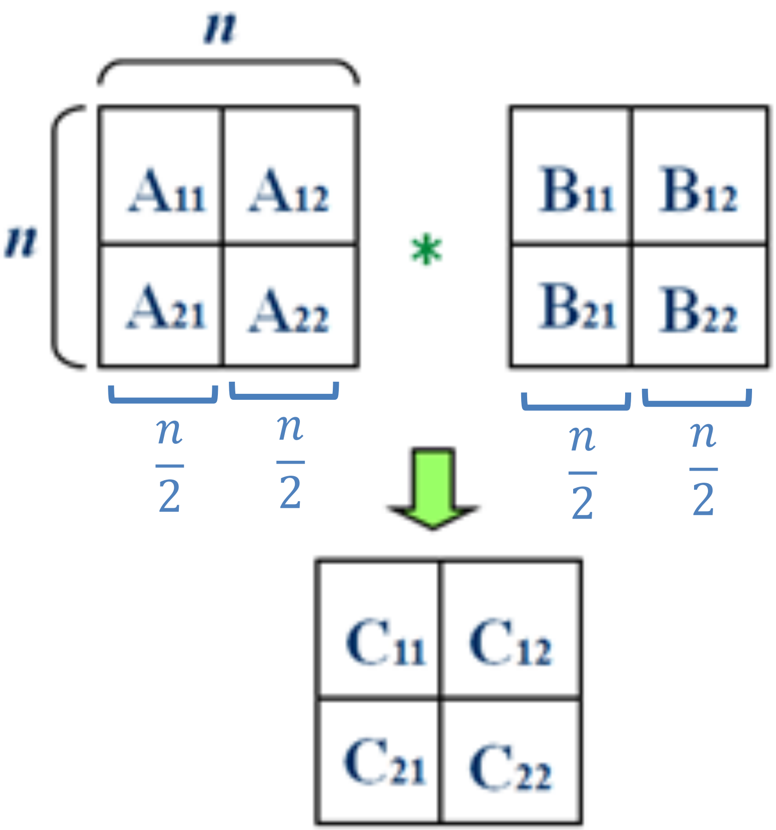
$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} A_{11}B_1 + A_{12}B_2 \\ A_{21}B_1 + A_{22}B_2 \end{bmatrix} = \left[\begin{array}{cc} -5 & 4 \\ -6 & 2 \\ \hline 2 & 1 \end{array}\right]$$

행렬곱셈 분할연산의 예 (3)

$$A = \left[\begin{array}{cc|ccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 2 & -1 & 4 & 2 & 1 \\ 3 & 1 & -1 & 7 & 5 \end{array}\right] = \begin{bmatrix} I_2 & O_{23} \\ P & Q \end{bmatrix} \qquad B = \left[\begin{array}{cc} 4 & -2 \\ 5 & 6 \\ \hline 7 & 3 \\ -1 & 0 \\ 1 & 6 \end{array}\right] = \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$AB = \begin{bmatrix} I & O \\ P & Q \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} IX + OY \\ PX + QY \end{bmatrix} = \begin{bmatrix} X \\ PX + QY \end{bmatrix} = \left[\begin{array}{cc} 4 & -2 \\ 5 & 6 \\ \hline 30 & 8 \\ 8 & 27 \end{array}\right]$$

행렬곱셈의 분할연산 - 행(열)의 차수가 매우 클 때



$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$(\frac{n}{2})^3 \times 8$번의 곱셈이 필요함

즉, n^3번의 곱셈연산이 필요함

여전히 행렬의 곱셈의 복잡도는 $O(n^3)$

그러나 부분결과를 얻을 때는 매우 유용

Volker Strassen 알고리즘 (1968년)

$$\overbrace{\begin{array}{c}\end{array}}^{n} \quad n\left[\begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline\end{array}\right] * \begin{array}{|c|c|}\hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline\end{array}$$

$M_1 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$M_2 = (A_{21} + A_{22}) * B_{11}$

$M_3 = A_{11} * (B_{12} - B_{22})$

$M_4 = A_{22} * (B_{21} - B_{11})$

$M_5 = (A_{11} + A_{12}) * B_{22}$

$M_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = M_1 + M_4 - M_5 + M_7$

$C_{12} = M_3 + M_5$

$C_{21} = M_2 + M_4$

$C_{22} = M_1 + M_3 - M_2 + M_6$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$(\frac{n}{2})^3 \times 7$번의 곱셈이 필요함. 즉, $\frac{7}{8}n^3$번의 곱셈연산이 필요함

덧셈까지 고려한 복잡도는

$T(n) = 7T\left(\frac{n}{2}\right)$ [redacted]

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

$= O(n^{2.81})$

Strassen 의 천재성

Strassen 의 방법

$M_1 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$M_2 = (A_{21} + A_{22}) * B_{11}$

$M_3 = A_{11} * (B_{12} - B_{22})$

$M_4 = A_{22} * (B_{21} - B_{11})$

$M_5 = (A_{11} + A_{12}) * B_{22}$

$M_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = M_1 + M_4 - M_5 + M_7$

$C_{12} = M_3 + M_5$

$C_{21} = M_2 + M_4$

$C_{22} = M_1 + M_3 - M_2 + M_6$

유전알고리즘으로 수십년 만에 찾은 또다른 방법

$M_1 = A_1 + A_2)(B_3 + B_4)$

$M_2 = A_1 - A_2)(B_3 - B_4)$

$M_3 = A_2 - A_4)(B_1 - B_2 - B_3 + B_4)$

$M_4 = A_2 + A_4)(B_1 + B_2 + B_3 + B_4)$

$M_5 = A_1 - A_4)(B_1 + B_4)$

$M_6 = A_1 + A_2 - A_3 - A_4)(B_1 - B_3)$

$M_7 = A_1 - A_2 + A_3 - A_4)(B_1 + B_3)$

$C_{11} = -0.5P_1 + 0.5P_2 - 0.5P_3 + 0.5P_4 + P_5$

$C_{12} = 0.5P_1 + 0.5P_2$

$C_{21} = -0.5P_1 - 0.5P_2 + 0.5P_3 + 0.5P_4 - 0.5P_6 + 0.5P_7$

$C_{22} = 0.5P_1 - 0.5P_2 - P_5 + 0.5P_6 + 0.5P_7$

이 외에 여러가지 방법이 존재하지만 스트라센 알고리즘보다 나은 복잡도를 가지는 방법은 아직 못 찾았음.

큰 정수의 곱 구하기

a=4, b=2 의 경우

- Multiply two n-bit integers x and y.
  - Divide step: split x and y into high-order and low-order bits

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$
$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R$$

- We can then define $x*y$ by multiplying the parts and adding:

$$\begin{aligned} xy &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R. \end{aligned}$$

So, $T(n) = 4T(n/2) + n = O(n^2)$.

a=3, b=2 의 경우

- The mathematician Carl Friedrich Gauss (1777-1855) once noticed that the product of two complex numbers involves 4 multiplications,

$$(a + bi)(c + di) = ac - bd + (bc + ad)i$$

but it can be done with just 3, since

$$(bc + ad) = (a + b)(c + d) - ac - bd$$

$$(a+bi)(c+di) = ac-bd+[(a+b)(c+d) - ac - bd]i$$

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R)$$
$$= x_L y_L\, 2^n + [(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R]2^{n/2} + x_R y_R$$

중복 중복

since $(a+b)(c+d) = ac + bd + [(a+b)(c+d) - ac - bd]$

- So, $T(n) = 3T(n/2) + n$, which implies $T(n)$ is $O(3^{\log_2 n}) = O(n^{\log_2 3}) = O(n^{1.59})$.