

## 2장 분할정복법 (divide-and-conquer)

# 분할정복(Divide-and-Conquer)식 설계 전략

- 분할(Divide): 해결하기 쉽도록 문제를 여러 개의 작은 부분으로 나눈다.
- 정복(Conquer): 나눈 작은 문제를 각각 해결한다.
- 통합(Combine): (필요하다면) 해결된 해답을 모은다.

이러한 문제 해결 방법을 **하향식(top-down)** 접근방법이라고 한다.

# 이분검색(binary search): 재귀적 방식

- 문제: 크기가  $n$ 인 정렬된 배열  $S$ 에  $x$ 가 있는지를 결정하라.
- 입력: 자연수  $n$ , 비내림차순으로 정렬된 배열  $S[1..n]$ , 찾고자 하는 항목  $x$
- 출력: *location*,  $x$ 가  $S$ 의 어디에 있는지의 위치. 만약  $x$ 가  $S$ 에 없다면 0
- 설계전략:
  - ✓  $x$ 가 배열의 중간에 위치하고 있는 항목과 같으면,  $x$  찾음. 그렇지 않으면:
  - ✓ **분할**: 배열을 반으로 나누어서  $x$ 가 중앙에 위치한 항목보다 작으면 왼쪽에 위치한 배열 반쪽을 선택하고, 그렇지 않으면 오른쪽에 위치한 배열 반쪽을 선택한다.
  - ✓ **정복**: 선택된 반쪽 배열에서  $x$ 를 찾는다.
  - ✓ **통합**: (필요 없음)

# 이분검색(Binary Search): 재귀 알고리즘

```
index location (index low, index high) {
    index mid;

    if (low > high)
        return 0; // 찾지 못했음
    else {
        mid = (low + high) / 2 // 정수 나눗셈 (나머지 버림)
        if (x == S[mid])
            return mid; // 찾았음
        else if (x < S[mid])
            return location(low, mid-1); // 왼쪽 반을 선택함
        else
            return location(mid+1, high); // 오른쪽 반을 선택함
    }
}

...
locationout = location(1, n);
...
```

## Discussion

1. 입력 파라미터  $n, S, x$ 는 알고리즘 수행 중 변하지 않는 값이다. 따라서 함수를 재귀호출(recursive call)할 때 마다 이러한 변하지 않는 파라미터를 가지고 다니는 것은 극심한 낭비이다. 따라서  $n, S, x$ 를 전역(global) 변수로 지정하고, 재귀호출에는 인덱스만 넘겨 줌

```
index location (index low, index high) {  
    index mid;  
  
    if (low > high)  
        return 0;  
    else {  
        mid = (low + high) / 2  
        if (x == S[mid])  
            return mid;  
        else if (x < S[mid])  
            return location(low, mid-1);  
        else  
            return location(mid+1, high);  
    }  
}
```

2. 재귀 알고리즘(recursive algorithm)에서 모든 재귀호출이 알고리즘의 마지막(꼬리) 부분에서 이루어 질 때 꼬리 재귀호출(tail recursion)이라고 함
- 그 알고리즘은 반복 알고리즘(iterative algorithm)으로 변환하기가 수월하다. 일반적으로 재귀 알고리즘은 재귀 호출할 때마다 그 당시의 상태를 활성 레코드(activation records) 스택에 저장해 놓아야 하는 반면, 반복 알고리즘은 그럴 필요가 없기 때문에 일반적으로 더 효율적이다(빠르다). 그렇다고 반복 알고리즘의 계산복잡도가 재귀 알고리즘보다 좋다는 의미는 아니다. 반복 알고리즘이 상수적(constant factor)으로만 좋다(빠르다)는 말이다.

```
index location (index low, index high) {
    index mid;

    if (low > high)
        return 0;
    else {
        mid = (low + high) / 2
        if (x == S[mid])
            return mid;
        else if (x < S[mid])
            return location(low, mid-1);
        else
            return location(mid+1, high);
    }
}
```

# 최악의 경우 시간복잡도 분석

- 단위연산:  $x$ 와  $S[mid]$ 의 비교
- 입력 크기: 배열의 크기  $n (= high - low + 1)$
- 단위연산으로 설정한 조건 문을 2번 수행하지만, 사실상 비교는 한번 이루어진다고 봐도 된다. 그 이유는:
  - (1) 어셈블리 언어로는 하나의 조건 명령으로 충분히 구현할 수 있기 때문이기도 하고;
  - (2)  $x$ 를 찾기 전까지는 항상 2개의 조건 문을 수행하므로 하나로 묶어서 한 단위로 취급을 해도 되기 때문이기도 하다. 이와 같이 단위연산은 최대한 효율적으로(빠르게) 구현된다고 일반적으로 가정하여, 1 단위로 취급을 해도 된다.

```
index location (index low, index high) {  
    index mid;  
  
    if (low > high)  
        return 0;  
    else {  
        mid = (low + high) / 2  
        if (x == S[mid])  
            return mid;  
        else if (x < S[mid])  
            return location(low, mid-1);  
        else  
            return location(mid+1, high);  
    }  
}
```

- 경우 1: 검색하게 될 반쪽 배열의 크기가 항상 정확하게  $n/2$  이 되는 경우

$$W(n) = W(n/2) + 1, n > 1 \text{ 이고, } n = 2^k, (k \geq 1) \quad W(1) = 1$$

이 식의 해는 다음과 같이 구할 수 있다.

$$W(1)=1$$

$$W(2)=W(1)+1=2$$

$$W(4)=W(2)+1=3$$

$$W(8)=W(4)+1=4$$

$$W(16)=W(8)+1=5$$

.

.

$$W(2^k)=k+1$$

.

.

$$W(n)=\lg n + 1$$

반복대입법 (iterative substitution or iteration)

$$W(n) = W(n/2) + 1$$

$$= (W(n/2^2) + 1) + 1$$

$$= W(n/2^2) + 2$$

$$= \bullet \bullet$$

$$= \bullet \bullet$$

$$= W(n/2^k) + k, (n = 2^k \text{ 가정})$$

$$= 1 + k$$

$$= \lg n + 1$$



## 연습문제

$$W(n) = W(n / 2) + n, n \geq 2, n = 2^k, k \geq 1$$

$$W(1) = 1$$

## 추정 후 증명방법(substitution)

$$W(n) = W(n/2) + 1, n > 1 \text{ 이고, } n = 2^k, (k \geq 1)$$

$$W(1) = 1 \text{ 일 때}$$

$W(n) = \lg n + 1$  을 수학적귀납법 사용하여 증명

**귀납출발점:**  $n = 1$ 이면,  $W(1) = 1 = \lg 1 + 1$ .

**귀납가정:** 2의 거듭제곱(power)인 양의 정수  $n$ 에 대해서,  $W(n) = \lg n + 1$ 라고 가정한다.

**귀납단계:**  $W(2n) = \lg(2n) + 1$ 임을 보이면 된다. 재현식을 사용하면,

$$\begin{aligned} W(2n) &= W(n) + 1 && \text{재현식에의해서} \\ &= \lg n + 1 + 1 && \text{귀납가정에의해서} \\ &= \lg n + \lg 2 + 1 \\ &= \lg(2n) + 1 \end{aligned}$$

그러므로  $W(n) = \lg n + 1$

- **경우 2: 일반적인 경우 - 반쪽 배열의 크기는  $\lfloor \frac{n}{2} \rfloor$ 이 됨**

$\lfloor y \rfloor$ 란  $y$ 보다 작거나 같은 수 중 최대 정수를 나타낸다고 할 때,  $n$ 에 대해서 가운데 첨자는  $mid = \left\lfloor \frac{1+n}{2} \right\rfloor$  이 되는데, 이 때 각 부분배열의 크기는 다음과 같다.

n	왼쪽 부분배열의 크기	mid	오른쪽 부분배열의 크기
짝수	$n/2 - 1$	1	$n/2$
홀수	$(n-1)/2$	1	$(n-1)/2$

위의 표에 의하면 알고리즘이 다음 단계에 찾아야 할 항목의 개수는 기껏해야  $\lfloor \frac{n}{2} \rfloor$  개가 된다. 따라서 다음과 같은 재현식으로 표현할 수 있다.

$$W(n) = 1 + W(\lfloor \frac{n}{2} \rfloor) \quad n > 1 \text{ 일 때}$$

$$W(1) = 1$$

- 이 재현식의 해가  $W(n) = \lfloor \lg n \rfloor + 1$ 가 됨을  $n$ 에 대한 수학적귀납법으로 증명한다.

증명: 수학적귀납법

귀납출발점:  $n = 1$ 이면, 다음이 성립한다.

$$\lfloor \lg n \rfloor + 1 = \lfloor \lg 1 \rfloor + 1 = 0 + 1 = 1 = W(1)$$

귀납가정:  $n > 1$ 이고,  $1 < k < n$ 인 모든  $k$ 에 대해서,  $W(k) = \lfloor \lg k \rfloor + 1$ 가 성립한다고 가정한다.

귀납단계: (1)  $n$ 이 짝수이면 (즉,  $\lfloor \frac{n}{2} \rfloor = \frac{n}{2}$ ),

$$\begin{aligned}
 W(n) &= 1 + W(\lfloor \frac{n}{2} \rfloor) && \text{재현식에 의해서} \\
 &= 1 + \lfloor \lg \lfloor \frac{n}{2} \rfloor \rfloor + 1 && \text{귀납가정에 의해서} \\
 &= 2 + \lfloor \lg \lfloor \frac{n}{2} \rfloor \rfloor \\
 &= 2 + \lfloor \lg \frac{n}{2} \rfloor && n \text{이 짝수이므로} \\
 &= 2 + \lfloor \lg n - 1 \rfloor \\
 &= 2 + \lfloor \lg n \rfloor - 1 \\
 &= 1 + \lfloor \lg n \rfloor
 \end{aligned}$$

- (2)  $n$ 이 홀수이면 (즉,  $\lfloor \frac{n}{2} \rfloor = \frac{n-1}{2}$  ),

$$\begin{aligned}
 W(n) &= 1 + W(\lfloor \frac{n}{2} \rfloor) \\
 &= 1 + \lfloor \lg \lfloor \frac{n}{2} \rfloor \rfloor + 1 \\
 &= 2 + \lfloor \lg \lfloor \frac{n}{2} \rfloor \rfloor \\
 &= 2 + \lfloor \lg \frac{n-1}{2} \rfloor \\
 &= 2 + \lfloor \lg(n-1) - 1 \rfloor \\
 &= 2 + \lfloor \lg(n-1) \rfloor - 1 \\
 &= 1 + \lfloor \lg(n-1) \rfloor \\
 &= 1 + \lfloor \lg n \rfloor
 \end{aligned}$$

재현식에 의해서  
귀납가정에 의해서

$n$ 이 홀수이므로

$n$ 이 짝수라면  $n$ 이 2의 지수승값일 수도 있다.  
if  $n=16$ ,  $\text{floor}(\lg(16-1))=3$ ,  $\text{floor}(\lg(16))=4$   
if  $n=15$ ,  $\text{floor}(\lg(15-1))=3$ ,  $\text{floor}(\lg(15))=3$

$n$ 이 홀수이므로

따라서,  $W(n) = \lfloor \lg n \rfloor + 1 \in \Theta(\lg n)$ .

- floor function(바닥(마루)함수)

$\lfloor x \rfloor$  실수  $x$  에 대해,  $x$  보다 작거나 같은 정수 중 가장 큰 정수

(예)  $\lfloor 3.1 \rfloor = 3$        $\lfloor -3.1 \rfloor = -4$

- ceiling function(천장함수)

$\lceil x \rceil$  실수  $x$  에 대해,  $x$  보다 크거나 같은 정수 중 가장 작은 정수

(예)  $\lceil 2.5 \rceil = 3$        $\lceil -2.5 \rceil = -2$

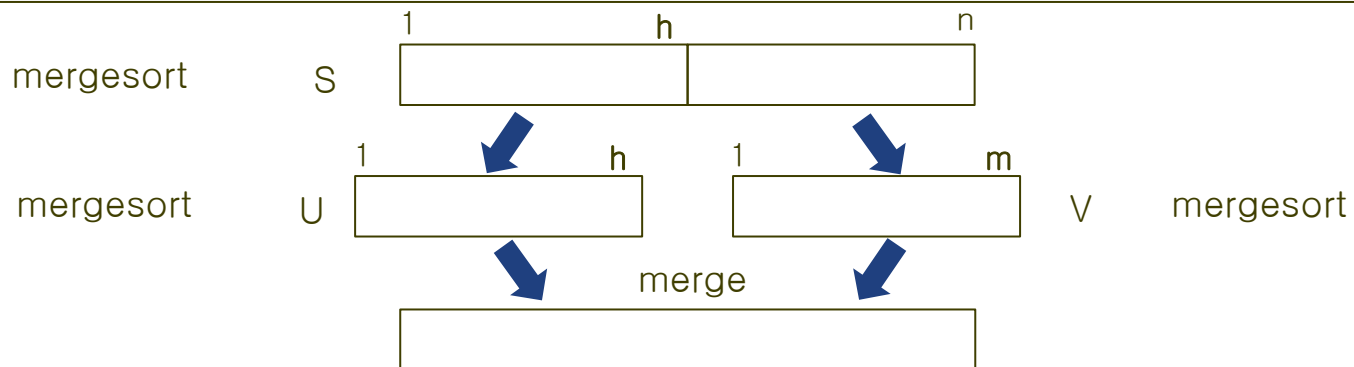
# 합병 정렬 (mergesort)

- 문제:  $n$ 개의 정수를 비내림차순으로 정렬하시오.
- 입력: 정수  $n$ , 크기가  $n$ 인 배열  $S[1..n]$
- 출력: 비내림차순으로 정렬된 배열  $S[1..n]$
- 보기: 27, 10, 12, 20, 25, 13, 15, 22

# 합병정렬

- 알고리즘:

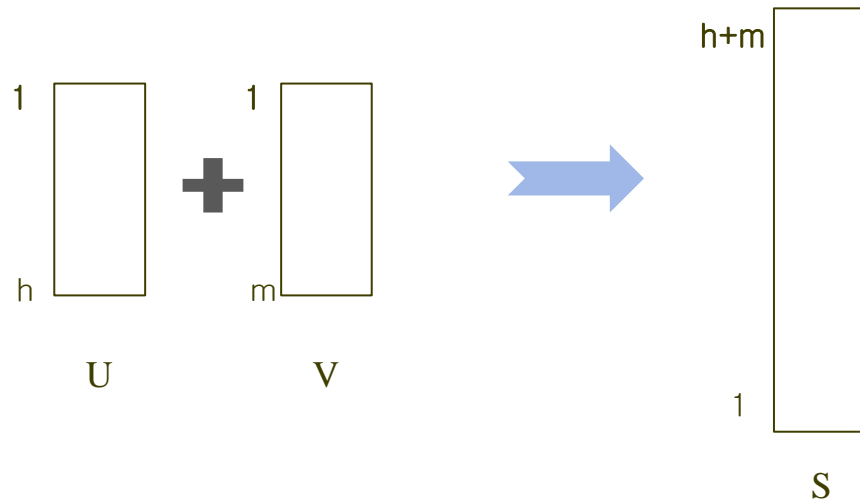
```
void mergesort (int n, keytype S[]) {  
    const int h = n / 2, m = n - h;  
    keytype U[1..h], V[1..m];  
  
    if (n > 1) {  
        copy S[1] through S[h] to U[1] through U[h];  
        copy S[h+1] through S[n] to V[1] through V[m];  
        mergesort(h, U);  
        mergesort(m, V);  
        merge(h, m, U, V, S);  
    }  
}
```

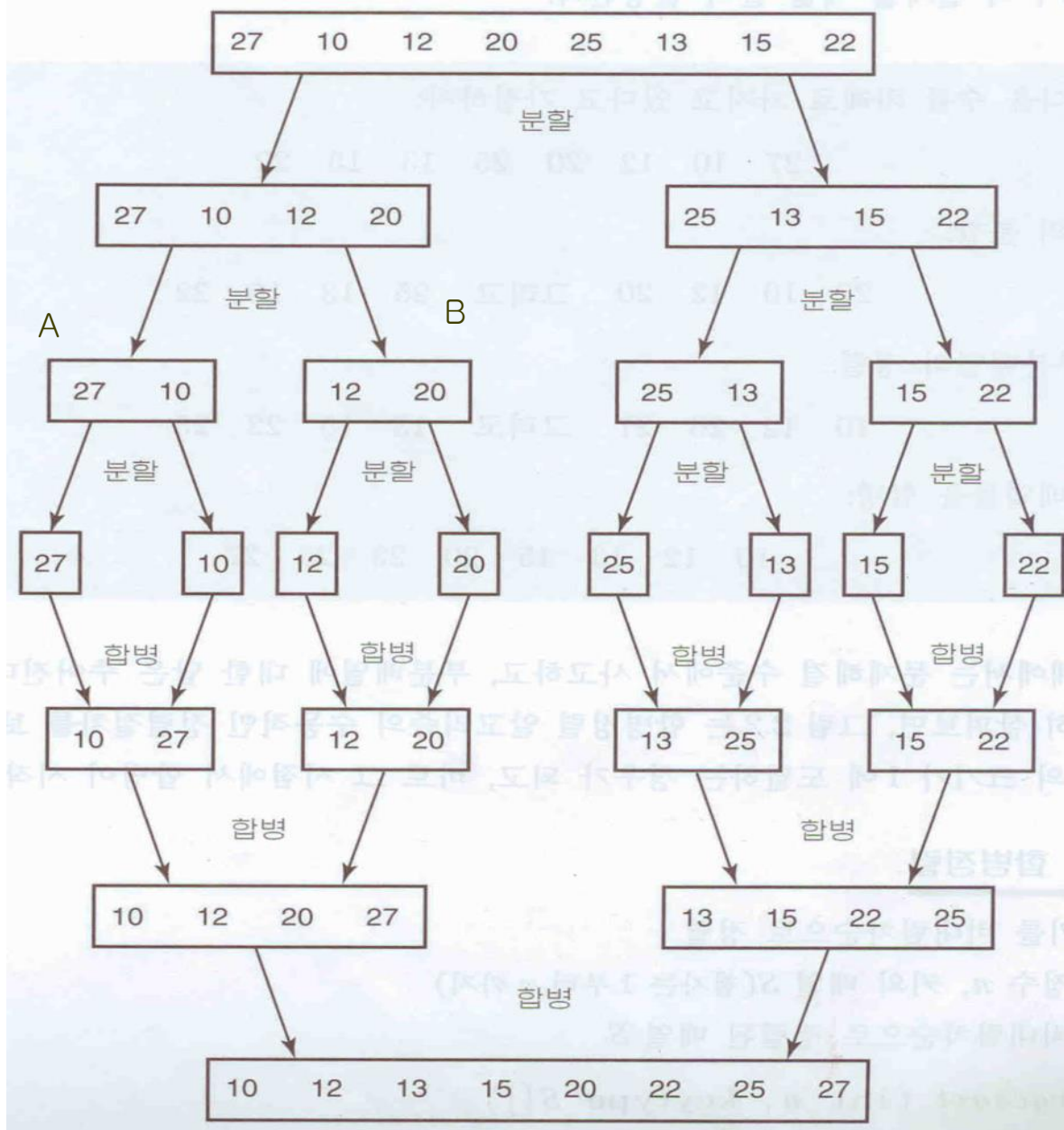




# 합병(merge)

- 문제: 두 개의 정렬된 배열을 하나의 정렬된 배열로 합병하시오.
- 입력: (1) 양의 정수  $h, m$ , (2) 정렬된 배열  $U[1..h]$ ,  $V[1..m]$
- 출력:  $U$ 와  $V$ 에 있는 키들을 하나의 배열에 정렬한  $S[1..h+m]$





● Fig 2.2 The steps done by a human when sorting with Mergesort

$k$	$U$	$V$	$S$ (결과)
1	<b>10</b> 12 20 27	<b>13</b> 15 22 25	10
2	10 <b>12</b> 20 27	<b>13</b> 15 22 25	10 12
3	10 12 <b>20</b> 27	<b>13</b> 15 22 25	10 12 13
4	10 12 <b>20</b> 27	13 <b>15</b> 22 25	10 12 13 15
5	10 12 <b>20</b> 27	13 15 <b>22</b> 25	10 12 13 15 20
6	10 12 20 <b>27</b>	13 15 <b>22</b> 25	10 12 13 15 20 22
7	10 12 20 <b>27</b>	13 15 22 <b>25</b>	10 12 13 15 20 22 25
—	10 12 20 27	13 15 22 25	10 12 13 15 20 22 25 27 ← 최종값

\* 비교되는 아이템은 진하게 표시되어 있다.

- 표 2.1     2개의 배열  $U$ 와  $V$ 를 하나의 배열  $S$ 로 합병하는 예

```

void merge(int h, int m, const keytype U[], const keytype V[],
           keytype S[]) {
    index i, j, k;
    i = 1; j = 1; k = 1;
    while (i <= h && j <= m) {
        if (U[i] < V[j]) {
            S[k] = U[i];
            i++;}
        else {
            S[k] = V[j];
            j++;}
        k++;
    }
    if (i > h)
        copy V[j] through V[m] to S[k] through S[h+m];
    else
        copy U[i] through U[h] to S[k] through S[h+m];
}

```

# 시간복잡도 분석

- 합병 알고리즘의 최악의 경우 시간복잡도 분석
  - ✓ 단위연산:  $U[i]$ 와  $V[j]$ 의 비교
  - ✓ 입력크기: 2개의 입력 배열에 각각 들어 있는 항목의 개수:  $h$ 와  $m$
  - ✓ 분석:  $i = h+1$ 이고,  $j = m$ 인 상태로 루프(loop)에서 빠져 나가는 때가 최악의 경우로서( $V$ 에 있는 처음  $m - 1$ 개의 항목이  $S$ 의 앞부분에 위치하고,  $U$ 에 있는  $h$ 개의 모든 항목이 그 뒤에 위치하는 경우), 이 때 단위연산의 실행 횟수는  $h + m - 1$ 이다. 따라서, 최악의 경우 합병하는 시간복잡도는  $W(h,m) = h + m - 1$ .
  - ✓ (예)  $U: 4\ 5\ 6\ 7$      $V: 1\ 2\ 3\ 8$

# 시간복잡도 분석

- 합병정렬 알고리즘의 최악의 경우 시간복잡도 분석
  - ✓ 단위연산: 합병 알고리즘 merge에서 발생하는 비교
  - ✓ 입력크기: 배열 S에 들어 있는 항목의 개수  $n$
  - ✓ 분석: 최악의 경우 수행시간은  $W(h,m) = W(h) + W(m) + h + m - 1$ 이 된다. 여기서  $W(h)$ 는 U를 정렬하는데 걸리는 시간,  $W(m)$ 은 V를 정렬하는데 걸리는 시간, 그리고  $h + m - 1$ 은 합병하는데 걸리는 시간이다. 정수  $n$ 을  $2^k, (k \geq 1)$ 이라고 가정하면,  $h = \frac{n}{2}, m = \frac{n}{2}$  이 된다. 따라서 최악의 경우 재현식은:  $W(n) = 2W(\frac{n}{2}) + n - 1 \quad n > 1$ 이고,  $n = 2^k (k \geq 1)$   
 $W(1) = 0$

이 재현식의 해는 2장의 끝 도사정리의 2번을 적용하면,

$$W(n) \in \Theta(n \lg n)$$

$$W(n) = 2W\left(\frac{n}{2}\right) + n - 1, \text{ for } n \geq 2 \text{ with } W(1) = 0.$$

Assume that  $n = 2^k$ .  $k = \log_2 n = \lg n$

$$W(n) = 2W\left(\frac{n}{2}\right) + n - 1$$

$$= 2\left(2W\left(\frac{n}{2^2}\right) + \frac{n}{2} - 1\right) + n - 1 = 2^2 W\left(\frac{n}{2^2}\right) + (n - 2) + (n - 1)$$

$$= 2^2\left(2W\left(\frac{n}{2^3}\right) + \frac{n}{2^2} - 1\right) + (n - 2) + (n - 1) = 2^3 W\left(\frac{n}{2^3}\right) + (n - 2^2) + (n - 2) + (n - 1)$$

.....

$$= 2^k W\left(\frac{n}{2^k}\right) + (n - 2^{k-1}) + (n - 2^{k-2}) + \cdots + (n - 2) + (n - 1)$$

$$= kn - (1 + 2 + 2^2 + \cdots + 2^{k-2} + 2^{k-1})$$

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

$$= n \lg n - (n - 1)$$

# 시간복잡도 분석

- $n$ 이 2의 승(power)의 형태가 아닌 경우의 재현식

$$W(n) = W(\lfloor \frac{n}{2} \rfloor) + W(\lceil \frac{n}{2} \rceil) + n - 1 \quad n > 1 \text{ 일 때}$$

$$W(1) = 0$$

그러나 이 재현식의 정확한 해를 구하기는 복잡하다.

그러나, 앞의 이분검색 알고리즘의 분석에서도 보았듯이,  $n = 2^k$ 라고 가정해서 해를 구하면, 이 재현식의 해와 같은 카테고리의 시간복잡도를 얻게 된다.

따라서 앞으로 이와 비슷한 재현식의 해를 구할 때,  $n = 2^k$ 라고 가정해서 구해도 점근적으로는 같은 해를 얻게 된다.

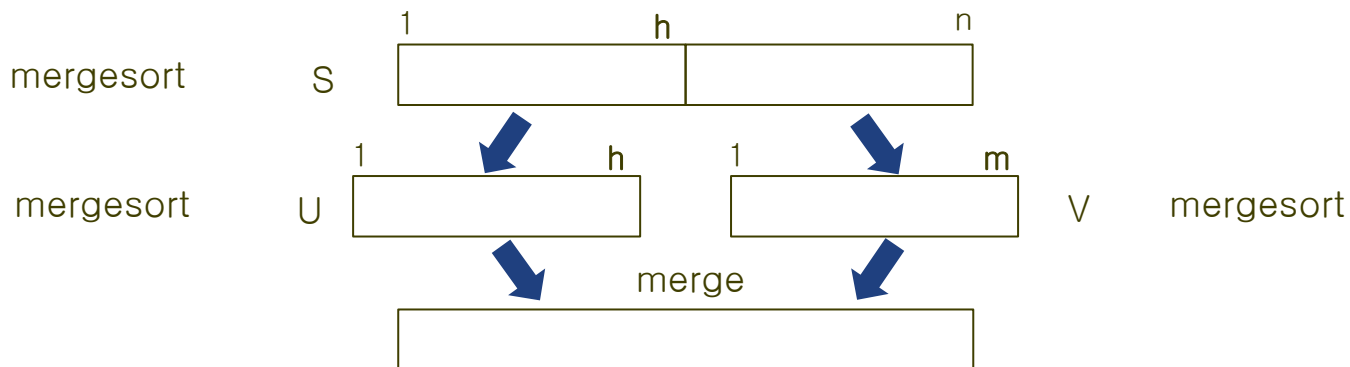


# 공간복잡도 분석

- 추가적인 저장장소를 사용하지 않고 정렬하는 알고리즘

## - 제자리정렬(in-place sort) 알고리즘

- 합병정렬 알고리즘은 제자리정렬 알고리즘이 아님. 입력배열  $S$ 이외에  $U$ 와  $V$ 를 추가로 만들어서 사용
- 하단의 재귀호출이 종료될 때까지 상위의 재귀호출이 생성하는 공간이 유지되어야 함.



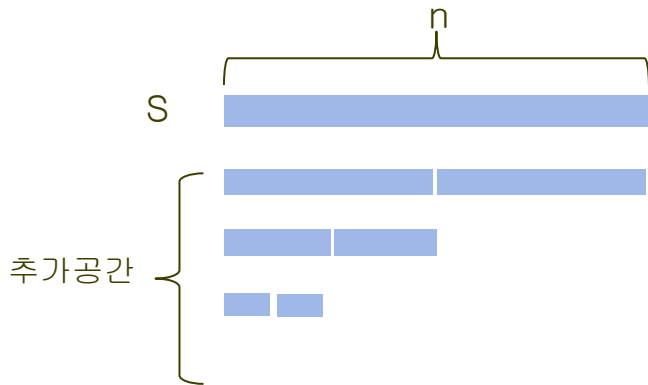
# 공간복잡도 분석

- 추가적인 저장장소: mergesort를 재귀호출할 때마다 크기가  $S$ 의 반이 되는  $U$ 와  $V$ 가 추가적으로 필요. (Merge 는 추가적인 저장장소 불필요).

처음  $S$ 의 크기가  $n$ 이면, 추가적으로 필요한  $U$ 와  $V$ 의 저장장소 크기의 합은  $n$ 이 된다. 다음 재귀 호출에는  $n/2$ 의 추가적으로 필요한 총 저장장소의 크기는

$$n + \frac{n}{2} + \frac{n}{4} + \dots = 2n \quad 2n \in \Theta(n)$$

- 추가적으로 필요한 저장장소가  $n$ 이 되도록, 즉, 공간복잡도가  $n$ 이 되도록 알고리즘을 향상시킬 수 있다(다음 절의 알고리즘). 그러나 합병정렬 알고리즘이 제자리정렬 알고리즘이 될 수는 없다.



```
void mergesort (int n, keytype S[]) {  
    .....  
  
    if (n > 1) {  
        copy S[1] through S[h] to U[1] through U[h];  
        copy S[h+1] through S[n] to V[1] through V[m];  
        mergesort (h,U);  
        mergesort (m,V);  
        merge (h,m,U,V,S);  
    }  
}
```



# 공간복잡도가 향상된 알고리즘

## ● 합병정렬(mergesort)

- ✓ 문제:  $n$ 개의 정수를 비내림차순으로 정렬하시오.
- ✓ 입력: 정수  $n$ , 크기가  $n$ 인 배열  $S[1..n]$
- ✓ 출력: 비내림차순으로 정렬된 배열  $S[1..n]$
- ✓ 알고리즘:

```
void mergesort2(index low, index high) {  
    index mid;  
    if (low < high) {  
        mid = (low + high) / 2;  
        mergesort2(low, mid);  
        mergesort2(mid+1, high);  
        merge2(low, mid, high);  
    }  
}  
  
...  
mergesort2(1, n);  
...
```

# 공간복잡도가 향상된 알고리즘

- 합병(merge2)

- ✓ 문제: 두 개의 정렬된 배열을 하나의 정렬된 배열로 합병하시오.

- ✓ 입력: (1) 첨자 low, mid, high,


(2) 부분 배열  $S[\text{low}..\text{high}]$ , 여기서  $S[\text{low}..\text{mid}]$ 와  $S[\text{mid}+1..\text{high}]$ 는 이미 각각 정렬이 완료되어 있음.

- ✓ 출력: 정렬이 완료된 부분배열  $S[\text{low}..\text{high}]$

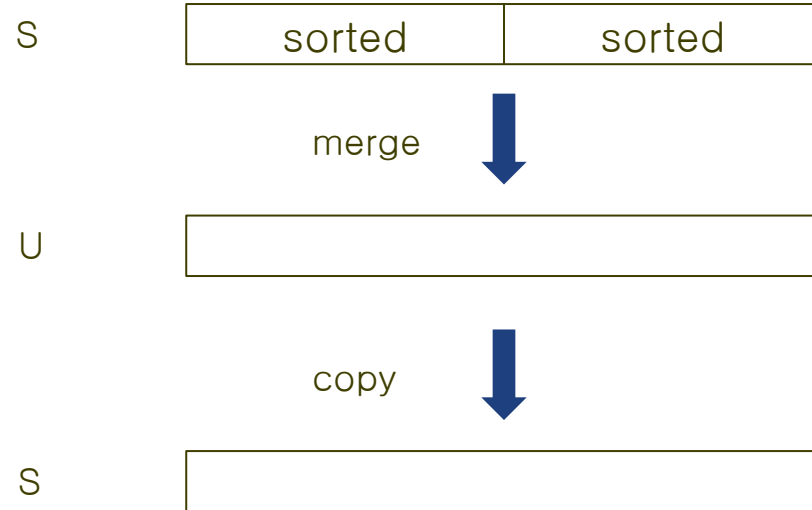
```
void mergesort2(index low, index high) {  
    index mid;  
    if (low < high) {  
        mid = (low + high) / 2;  
        mergesort2(low, mid);  
        mergesort2(mid+1, high);  
        merge2(low, mid, high);  
    }  
}
```

● 알고리즘:

```
void merge2(index low, index mid, index high) {  
    index i, j, k; keytype U[low..high]; // 합병하는데 필요한 지역 배열  
    i = low; j = mid + 1; k = low;  
    while (i <= mid && j <= high) {  
        if (S[i] < S[j]) {  
            U[k] = S[i];  
            i++;  
        }  
        else {  
            U[k] = S[j];  
            j++;  
        }  
        k++;  
    }  
    if (i > mid)  
        copy S[j] through S[high] to U[k] through U[high];  
    else  
        copy S[i] through S[mid] to U[k] through U[high];  
    copy U[low] through U[high] to S[low] through S[high];  
}
```



## merge2



추가 공간

복사는 나중에 수행

복사는 나중에 수행

2

A

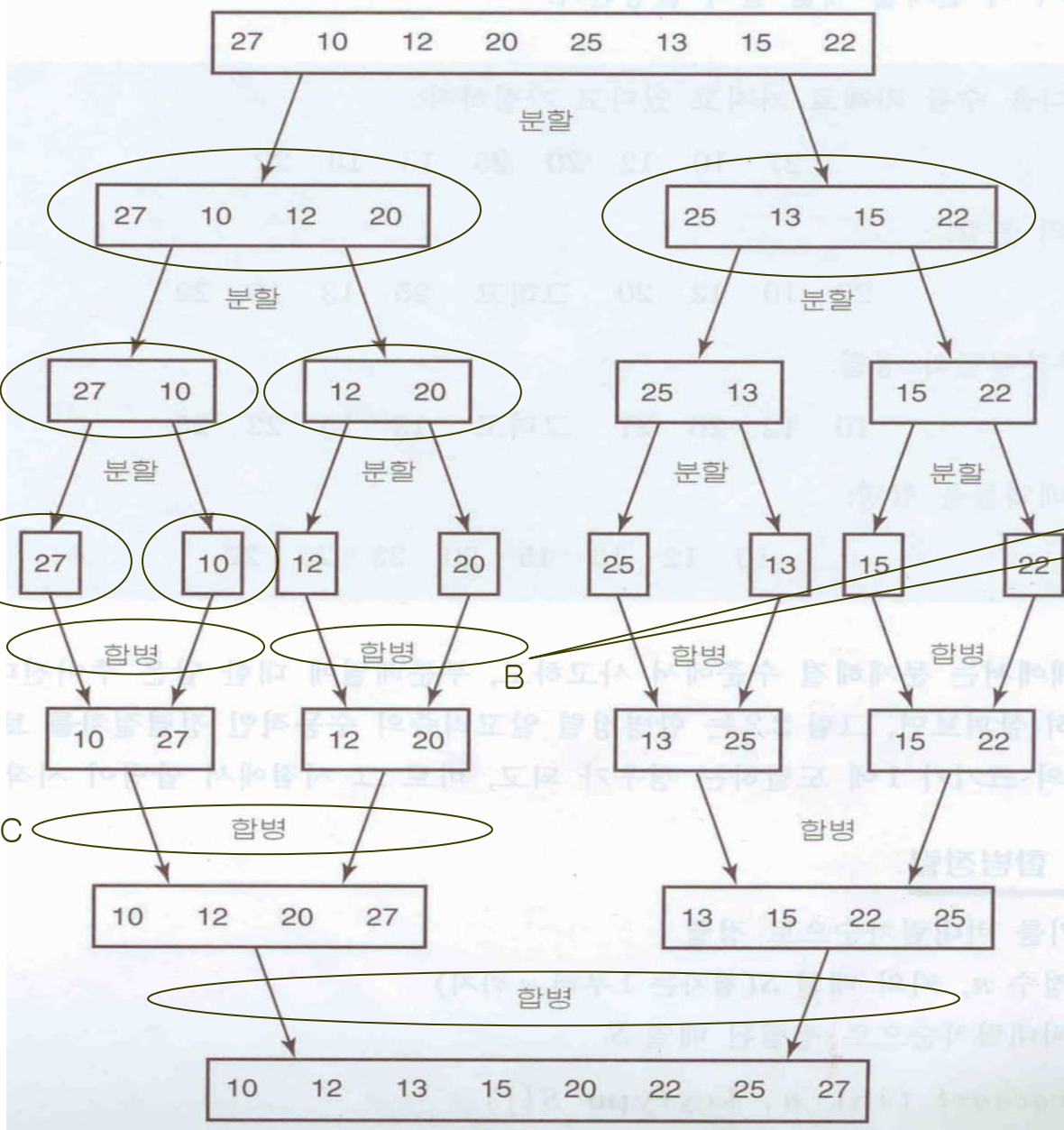
B

A단계에 사용  
된 공간을 재  
활용

4 (A+B에서 사용된  
공간 포함)

C

8(C를 포함)



● mergesort2의 절차. Additional space is  $n$ .

총추가 공간 8



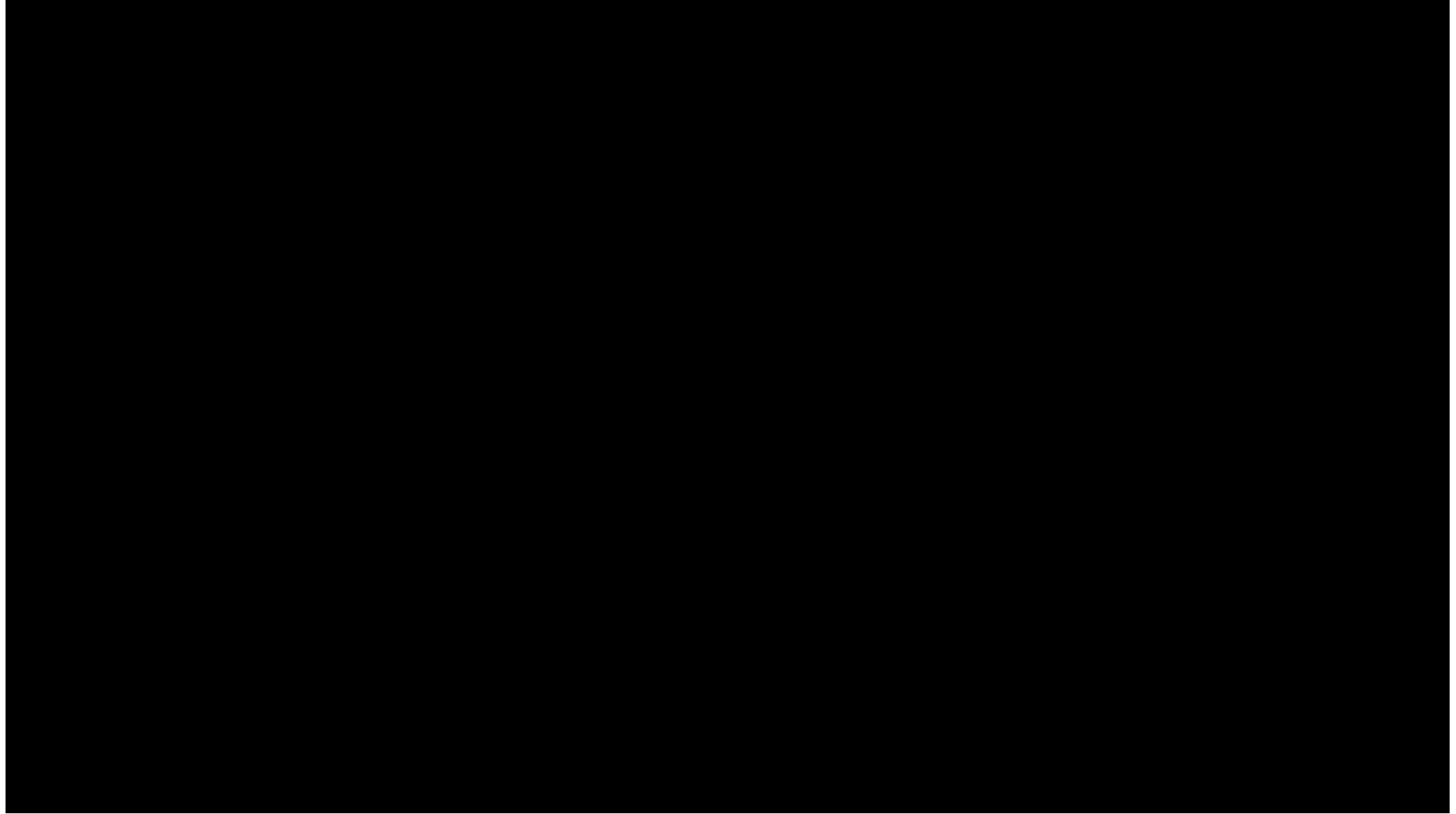
# 빠른정렬(Quicksort)

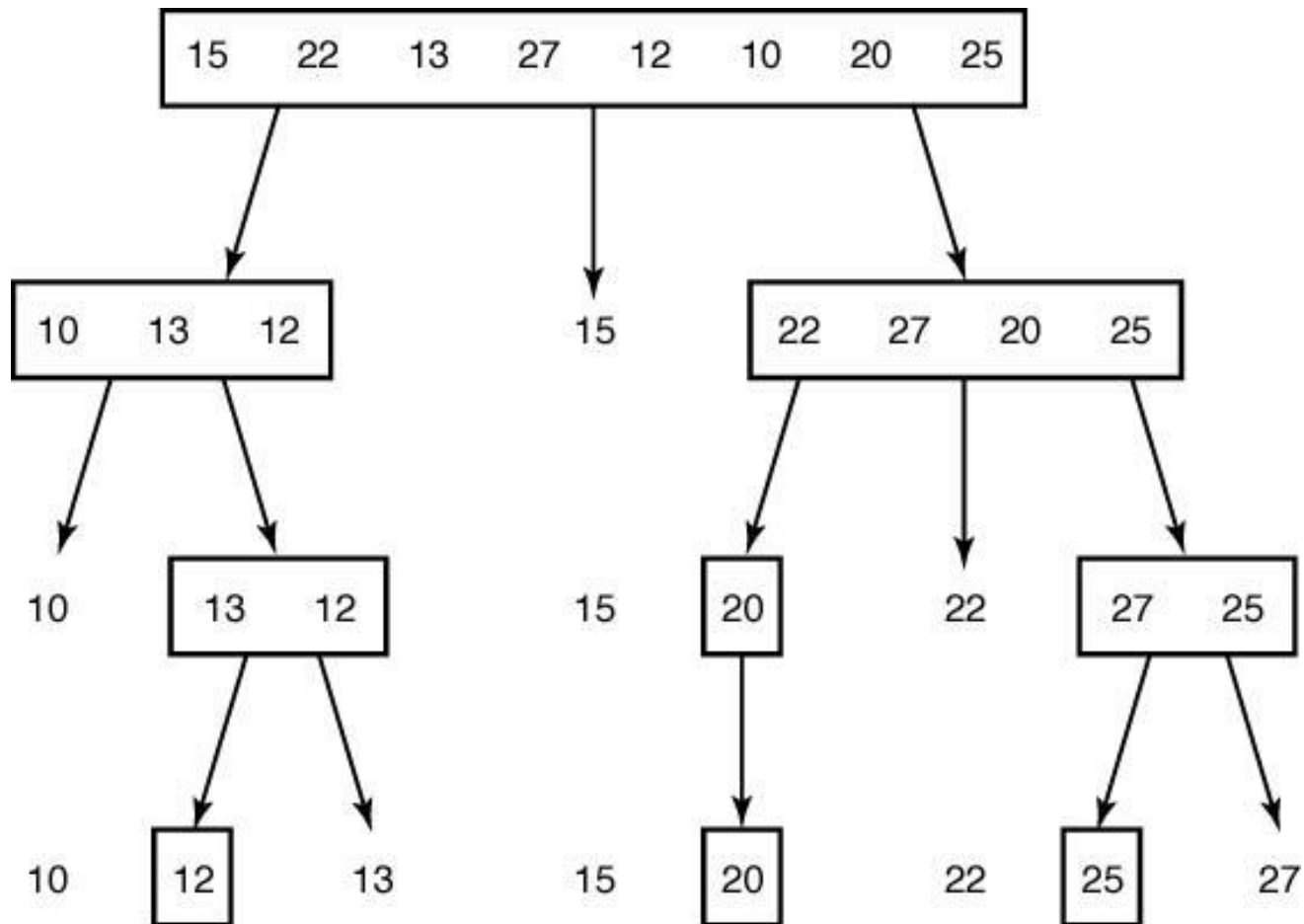
- 1962년에 영국의 호아(C.A.R. Hoare)의 의해서 고안
- 빠른정렬(quicksort)란 이름이 오해의 여지가 있음. 왜냐하면 사실 절대적으로 가장 빠른 정렬 알고리즘이라고 할 수는 없기 때문이다. 차라리 “분할교환정렬(partition exchange sort)”라고 부르는 게 더 정확함.
- 보기: 15 22 13 27 12 10 20 25

빠른정렬 영상

[https://www.youtube.com/watch?time\\_continue=2&v=cVMKXKoGu\\_Y](https://www.youtube.com/watch?time_continue=2&v=cVMKXKoGu_Y)

## 선택정렬과 빠른정렬





- 그림 2.3 빠른정렬 알고리즘의 수행절차. 부분배열은 네모로 둘러싸여 있는 데 반해, 기준 아이템은 그렇지 않다.

# 빠른정렬 알고리즘

- 문제:  $n$ 개의 정수를 비내림차순으로 정렬
- 입력: 정수  $n > 0$ , 크기가  $n$ 인 배열  $S[1..n]$
- 출력: 비내림차순으로 정렬된 배열  $S[1..n]$
- 알고리즘:

```
void quicksort (index low, index high) {  
    index pivotpoint;  
    if (high > low) {  
        partition(low, high, pivotpoint);  
        quicksort(low, pivotpoint-1);  
        quicksort(pivotpoint+1, high);  
    }  
}
```

# 분할 알고리즘

- 문제: 빠른정렬을 하기 위해서 배열 S를 둘로 나눈다.
- 입력: (1) 첨자 low, high (2) S의 부분배열 (첨자는 low에서 high)
- 출력: 첨자 low에서 high까지의 S의 부분배열의 기준점(pivot point), pivotpoint

```
void partition (index low, index high, index& pivotpoint) {  
    index i, j;  
    keytype pivotitem;  
    pivotitem = S[low];    //pivotitem으로 첫번째 항목을 고른다  
    j = low;  
    for(i = low + 1; i <= high; i++)  
        if (S[i] < pivotitem) {  
            j++;  
            exchange S[i] and S[j];  
        }  
    pivotpoint = j;  
    exchange S[low] and S[pivotpoint]; // pivotitem 값을 pivotpoint에 넣는다  
}
```

j: pivotitem 보다 작은 그룹의 제일 우측끝 데이터의 위치

<i>i</i>	<i>j</i>	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	
—	—	15	22	13	27	12	10	20	25	← 초기값
2	1	15	22	13	27	12	10	20	25	
3	1	15	22	13	27	12	10	20	25	
4	2	15	13	22	27	12	10	20	25	
5	2	15	13	22	27	12	10	20	25	
6	3	15	13	12	27	22	10	20	25	
7	4	15	13	12	10	22	27	20	25	
8	4	15	13	12	10	22	27	20	25	
—	4	10	13	12	15	22	27	20	25	← 최종값

```

void partition (index low, index high,
index& pivotpoint) {
    index i, j; keytype pivotitem;
    pivotitem = S[low];
    j = low;
    for(i = low + 1; i <= high; i++)
        if (S[i] < pivotitem) {
            j++;
            exchange S[i] and S[j];
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint]
}

```

\* 비교되는 아이템은 진하게 표시되어 있다. 바로 교환된 아이템은 상자로 둘러싸여 있다.

## ● 표 2.2 partition 프로시저의 예

pivot  
point

# 분할알고리즘(partition) 분석

- 분할 알고리즘의 모든 경우를 고려한 시간복잡도 분석
  - ✓ 단위연산:  $S[i]$ 와 pivotitem과의 비교
  - ✓ 입력크기: 부분배열이 가지고 있는 항목의 수,  $n = high - low + 1$
  - ✓ 분석: 배열의 첫번째 항목만 제외하고 모든 항목을 한번씩 비교하므로,  $T(n) = n - 1$ 이다.

```
void partition (index low, index high,
index& pivotpoint) {
    index i, j; keytype pivotitem;
    pivotitem = S[low];
    j = low;
    for(i = low + 1; i <= high; i++)
        if (S[i] < pivotitem) {
            j++;
            exchange S[i] and S[j];
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint]
}
```

# quicksort 분석

```
void partition (index low, index high,
index& pivotpoint) {
    index i, j; keytype pivotitem;
    pivotitem = S[low];
    j = low;
    for(i = low + 1; i <= high; i++)
        if (S[i] < pivotitem) {
            j++;
            exchange S[i] and S[j];
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint]
}
```

- 빠른정렬 알고리즘의 최악의 경우를 고려한 시간복잡도 분석
  - ✓ 단위연산: 분할알고리즘의  $S[i]$ 와 pivotitem과의 비교
  - ✓ 입력크기: 배열이  $S$ 가 가지고 있는 항목의 수,  $n$
  - ✓ 분석: 입력이 비내림차순으로 정렬이 되어 있는 경우가 최악. 첫번째(기준점) 항목보다 작은 항목은 없으므로, 크기가  $n$ 인 배열은 크기가 0인 부분배열은 왼쪽에 오고, 크기가  $n-1$ 인 부분배열은 오른쪽에 오도록 하여 계속 쪼개진다. 따라서,  $T(n) = T(0) + T(n - 1) + n - 1$

그런데,  $T(0) = 0$ 이므로, 재현식은 다음과 같이 된다.

$$T(n) = T(n - 1) + n - 1, n > 0 \text{이면}$$

$$T(0) = 0$$

[예]  $S = [1, 2, 3, 4, 5, 6, 7, 8]$



# 분석

이 재현식을 풀면,

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

$$T(n - 1) = T(n - 2) + n - 2$$

$$T(n - 2) = T(n - 3) + n - 3$$

...

$$T(2) = T(1) + 1$$

$$T(1) = T(0) + 0$$

$$T(0) = 0$$

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

결론적으로 빠른정렬 알고리즘의 최악의 시간복잡도는  $n(n-1)/2$ .

그러면 시간이 더 많이 걸리는 경우는 있을까? 이 경우가 최악의 경우이며, 따라서 이 보다 더 많은 시간이 걸릴 수가 없다는 사실을 수학적으로 엄밀하게 증명해 보자.

- 모든 정수  $n$ 에 대해서,  $W(n) \leq \frac{n(n-1)}{2}$  임을 증명하시오.

증명: (수학적귀납법)

귀납출발점:  $n = 0$ 일 때,  $W(0) \leq \frac{0(0-1)}{2}$

귀납가정:  $0 \leq k < n$ 인 모든  $k$ 에 대해서,  $W(k) \leq \frac{k(k-1)}{2}$

귀납단계:  $W(n) \leq \frac{n(n-1)}{2}$

$W(n) \leq W(p-1) + W(n-p) + n-1$  pivotpoint 값이  $p$ 인 경우 재현식에 의해서

$\leq \frac{(p-1)(p-2)}{2} + \frac{(n-p)(n-p-1)}{2} + n-1$  귀납가정에 의해서

$$= \frac{p^2 - 3p + 2 + (n-p)^2 - n + p + 2n - 2}{2}$$

$$= \frac{p^2 + (n-p)^2 + n - 2p}{2}$$

여기서  $p$ 가  $n-1$ 일 때 최대값을 가진다. 따라서

$$\max_{1 \leq p \leq n-1} (p^2 + (n-p)^2) = 1^2 + (n-1)^2 = n^2 - 2n + 2$$

가 되고, 결과적으로

$$W(n) \leq \frac{p^2 + (n-p)^2 + n - 2p}{2} \leq \frac{n^2 - 2n + 2 + n - 2}{2} = \frac{n^2 - n}{2} = \frac{n(n-1)}{2}$$

가 된다. 따라서 최악의 경우 시간복잡도는

$$W(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

## ● 평균의 경우를 고려한 시간복잡도 분석

- ✓ 단위연산: 분할알고리즘의  $S[i]$ 와 pivotitem과의 비교
- ✓ 입력크기: 배열  $S$ 가 가지고 있는 항목의 수,  $n$
- ✓ 분석:  $A(n)$ 을  $n$ 개의 데이터를 정렬하는데 걸리는 평균시간이라고 한다. pivotitem이 정렬 후  $p$ 번째 데이터가 될 확률은  $1/n$ . 기준점이  $p$ 일 때 두 부분배열을 정렬하는데 걸리는 평균시간은  $[A(p-1) + A(n-p)]$ 이고, 분할하는데 걸리는 시간은  $n-1$ 이므로, 평균적인 시간복잡도는

$$\begin{aligned} A(n) &= \sum_{p=1}^n \frac{1}{n} [A(p-1) + A(n-p)] + n-1 \\ &= \frac{1}{n} [(A(0) + A(n-1)) \\ &\quad + (A(1) + A(n-2)) \\ &\quad \dots \\ &\quad + (A(n-2) + A(1)) \\ &\quad + (A(n-1) + A(0))] + n-1 \\ &= \frac{2}{n} \sum_{p=1}^n A(p-1) + n-1 \end{aligned}$$

양변을  $n$ 으로 곱하면,

$$nA(n) = 2 \sum_{p=1}^n A(p-1) + n(n-1) \quad (1)$$

$n$ 대신  $n-1$ 을 대입하면,

$$(n-1)A(n-1) = 2 \sum_{p=1}^{n-1} A(p-1) + (n-1)(n-2) \quad (2)$$

(1)에서 (2)를 빼면,

$$nA(n) - (n-1)A(n-1) = 2A(n-1) + 2(n-1)$$

간단히 정리하면,

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

여기서,

$$a_n = \frac{A(n)}{n+1}$$

라고 하면, 다음과 같은 재현식을 얻을 수가 있다.

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)} \quad n > 0 \text{ 이면}$$

$$a_0 = 0$$

그러면,

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)}, \quad a_{n-1} = a_{n-2} + \frac{2(n-2)}{(n-1)n}, \quad \dots, \quad a_2 = a_1 + \frac{1}{3}, \quad a_1 = a_0 + 0$$

따라서, 해는

$$\begin{aligned} a_n &= \sum_{i=1}^n \frac{2(i-1)}{i(i+1)} \\ &= 2 \left( \sum_{i=1}^n \frac{1}{i+1} - \sum_{i=1}^n \frac{1}{i(i+1)} \right) \end{aligned}$$

여기에서 오른쪽 항은 무시해도 될 만큼 작으므로 무시한다.

$\ln n = \log_e n$ 이고,

$$\sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \int_1^n \frac{1}{x} dx = \ln n$$

이므로, 해는  $a_n \approx 2 \ln n$ . 그리고  $\lg n = \ln n / \ln 2$  따라서,

$$\begin{aligned} A(n) &= (n+1)a_n \\ &\approx (n+1)2 \ln n \\ &= (n+1)2(\ln 2)(\lg n) \\ &\approx 1.38(n+1) \lg n \quad (\ln 2 \approx 0.693) \\ &\in \Theta(n \lg n) \end{aligned}$$

$$a_n = \frac{A(n)}{n+1}$$

Quicksort는 평균적으로  $O(n \lg n)$  시간의 우수한 알고리즘

Best 경우: 문제가 매번 반씩으로 나누어질 때

$$T(n) = 2T(n/2) + n - 1 \in \Theta(n \lg n)$$

# 행렬 곱셈(matrix multiplication)

## ● 단순한 행렬곱셈 알고리즘

- ✓ 문제:  $n \times n$  크기의 행렬의 곱을 구하시오.
- ✓ 입력: 양수  $n$ ,  $n \times n$  크기의 행렬 A와 B
- ✓ 출력: 행렬 A와 B의 곱인 C

```
void matrixmult (int n, const number A[][], const number B[][],
                  number C[][]) {
    index i, j, k;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            C[i][j] = 0;
            for (k = 1; k <= n; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
}
```

```

void matrixmult (. . . ) {
    index i, j, k;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            C[i][j] = 0;
            for (k = 1; k <= n; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
}

```

## ● 시간복잡도 분석 I:

- ✓ 단위연산: 가장 안쪽의 루프에 있는 곱셈하는 연산
- ✓ 입력크기: 행과 열의 수,  $n$
- ✓ 모든 경우 시간복잡도 분석: 총 곱셈의 횟수

$$T(n) = n \times n \times n = n^3 \in \Theta(n^3)$$

## ● 시간복잡도 분석 II (알고리즘을 약간 수정)

- ✓ 단위연산: 가장 안쪽의 루프에 있는 덧셈하는 연산
- ✓ 입력크기: 행과 열의 수,  $n$
- ✓ 모든 경우 시간복잡도 분석: 총 덧셈의 횟수

$$T(n) = (n-1) \times n \times n = n^3 - n^2 \in \Theta(n^3)$$



## 2 × 2 행렬 곱셈(단순한 방법):

- 문제: 두 2 × 2 행렬  $A$ 와  $B$ 의 곱(product)  $C$ ,

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

- 시간복잡도 분석: 8번의 곱셈과 4번의 덧셈이 필요

# 쉬트라센(Strassen)의 방법

- 문제: 두  $2 \times 2$  행렬  $A$ 와  $B$ 의 곱(product)  $C$ ,

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- 쉬트라센(Strassen)의 해:

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

여기서

$$m_1 = (a_{11} + a_{22}) \times (b_{11} + b_{22})$$

$$m_2 = (a_{21} + a_{22}) \times b_{11}$$

$$m_3 = a_{11} \times (b_{12} - b_{22})$$

$$m_4 = a_{22} \times (b_{21} - b_{11})$$

$$m_5 = (a_{11} + a_{12}) \times b_{22}$$

$$m_6 = (a_{21} - a_{11}) \times (b_{11} + b_{12})$$

$$m_7 = (a_{12} - a_{22}) \times (b_{21} + b_{22})$$

- 시간복잡도 분석: 쉬트라센의 방법은 7번의 곱셈과 18번의 덧셈/뺄셈을 필요.
- 언뜻 보서는 전혀 좋아지지 않았다!
- 그러나 행렬의 크기가 커지면 쉬트라센의 방법이 효율적임.

# $n \times n$ 행렬 곱셈: 슈트라센의 방법

- 문제:  $n$ 이 2의 거듭제곱이고, 각 행렬을 4개의 부분행렬(submatrix)로 나누고 가정하자. 두  $n \times n$  행렬  $A$ 와  $B$ 의 곱  $C$ :

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

- 슈트라센(Strassen)의 해:

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

여기서

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

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

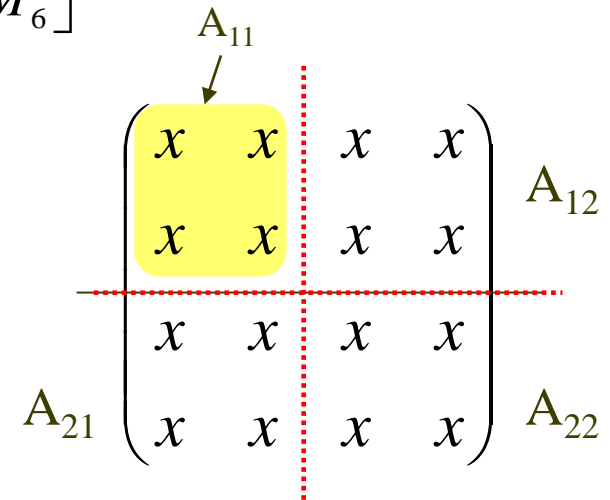
$$M_3 = A_{11} \times (B_{12} - B_{22})$$

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

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

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


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



# 쉬트라센의 알고리즘

- 문제:  $n$ 이 2의 거듭제곱일 때,  $n \times n$  크기의 두 행렬의 곱을 구하시오.
- 입력: 정수  $n$ ,  $n \times n$  크기의 행렬 A와 B
- 출력: 행렬 A와 B의 곱인 C

```
void strassen (int n, n*n_matrix A, n*n_matrix B, n*n_matrix& C) {  
    if (n <= 임계점)  
        단순한 알고리즘을 사용하여 C = A * B를 계산;  
    else {  
        A를 4개의 부분행렬  $A_{11}, A_{12}, A_{21}, A_{22}$ 로 분할;  
        B를 4개의 부분행렬  $B_{11}, B_{12}, B_{21}, B_{22}$ 로 분할;  
        쉬트라센의 방법을 사용하여 C = A * B를 계산;  
        // 되부르는 호출의 예: strassen(n/2,  $A_{11}+A_{22}$ ,  $B_{11}+B_{22}$ ,  $M_1$ )  
    }  
}
```

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

- 용어: 임계점(threshold)이란? 두 알고리즘의 효율성이 교차하는 문제의 크기.

# 분석

## ● 단순한 방법의 시간복잡도 분석

- ✓  $T(n)$ :  $n \times n$  크기의 행렬 A와 B를 곱하는데 걸리는 시간
- ✓ 단위연산: 곱셈하는 연산
- ✓ 입력크기: 행과 열의 수,  $n$
- ✓ 모든 경우 시간복잡도 분석: 임계값을 1이라고 하자. (임계값은 차수에 전혀 영향을 미치지 않는다.)

재현식은  $T(n) = 8T(\frac{n}{2})$ ,  $n > 1$ 이고,  $n = 2^k$  ( $k \geq 1$ )

$$T(1) = 1$$

이 식을 전개해 보면,

$$T(n) = 8 \times 8 \times \cdots \times 8 \quad (k \text{ 번})$$

$$= 8^k$$

$$= 8^{\lg n}$$

$$= n^{\lg 8}$$

$$= n^3$$

$$\in \Theta(n^3)$$

$$\begin{aligned} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \\ &= \begin{bmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{bmatrix} \end{aligned}$$

## ● 쉬트라센 방법의 시간복잡도 분석 I

- ✓ 단위연산: 곱셈하는 연산
- ✓ 입력크기: 행과 열의 수,  $n$
- ✓ 모든 경우 시간복잡도 분석: 임계값을 1이라고 하자. (임계값은 차수에 전혀 영향을 미치지 않는다.)

재현식은

$$T(n) = 7T\left(\frac{n}{2}\right), \quad n > 1 \text{ 이고, } n = 2^k (k \geq 1)$$
$$T(1) = 1$$

이 식을 전개해 보면,

$$\begin{aligned} T(n) &= 7 \times 7 \times \dots \times 7 \quad (k \text{ 번}) \\ &= 7^k \\ &= 7^{\lg n} \\ &= n^{\lg 7} \\ &= n^{2.81} \\ &\in \Theta(n^{2.81}) \end{aligned}$$

행렬의 크기가 2의 지수가 아닌 경우에는 크기를 2의 지수로 만들기 위해 필요한 만큼의 0 데이터를 넣는다.

## ● 쉬트라센방법의 시간복잡도 분석 II

- ✓ 단위연산: 덧셈/뺄셈하는 연산
- ✓ 입력크기: 행과 열의 수,  $n$
- ✓ 모든 경우 시간복잡도 분석: 위에서와 마찬가지로 임계값을 1이라고 하자. 재현식은

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2, \quad n > 1 \text{ 이고, } n = 2^k (k \geq 1)$$

$$T(1) = 0$$

7회의 곱셈  
문제

18회의 덧셈

도사정리의 3가지 중에서 1번을 이용하면 간단히 해를 구할 수 있다.

$$\begin{aligned} T(n) &= 6n^{\lg_2 7} - 6n^2 \\ &= 6n^{2.81} - 6n^2 \\ &\in \Theta(n^{2.81}) \end{aligned}$$

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

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

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

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

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

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

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

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

	표준알고리즘	쉬트라센 알고리즘
곱셈	$n^3$	$n^{2.81}$
덧셈/뺄셈	$n^3 - n^2$	$6n^{2.81} - 6n^2$

표2.3  $n \times n$  행렬을 곱하는 두 알고리즘의 비교



## Discussion

- 두 개의 행렬을 곱하기 위한 문제에 대해서 시간복잡도가  $\Theta(n^2)$ 이 되는 알고리즘을 만들어 낸 사람은 아무도 없다.
- 게다가 그러한 알고리즘을 만들 수 없다고 증명한 사람도 아무도 없다.
- Shumel Winograd: 덧셈/뺄셈 15회만 수행하는 변형된 쉬트라센 알고리즘 고안

$$T(n) = 5n^{2.81} - 5n^2$$

- Coppersmith와 Winograd(1987): 곱셈을 단위연산으로 한 시간복잡도가

$$T(n) = 5n^{2.38}$$

인 알고리즘 제안- 비효율적

# 큰 정수 계산법

## ● 하드웨어의 용량을 초과하는 정수연산 – 천문학

✓ 정수 배열을 이용한 큰 정수의 표현

✓ (예) 543,127

5	4	3	1	2	7
S[6]	S[5]	S[4]	S[3]	S[2]	S[1]

✓  $n$ : 큰 정수의 숫자(digit) 개수

- 단순 곱셈은  $n^2$ 시간 걸림. 덧셈/뺄셈은 1차 시간에 수행 가능

- 1차시간 가능:  $u \times 10^m$ ,  $u \text{ divide } 10^m$ ,  $u \bmod 10^m$

✓  $567,832 = 567 \times 10^3 + 832$ ,

✓  $9,423,723 = 9423 \times 10^3 + 723$

$$u = \underbrace{x}_{n \text{ digits}} \times 10^m + \underbrace{y}_{\lceil n/2 \rceil \text{ digits}}$$

$$m = \left\lfloor \frac{n}{2} \right\rfloor$$

	aaaa
x	bbbb
<hr/>	
	cccc
	cccc
	cccc
	cccc

$$u = x \times 10^m + y, v = w \times 10^m + z$$

$$\begin{aligned} u \times v &= (x \times 10^m + y) \times (w \times 10^m + z) \\ &= xw \times 10^{2m} + (xz + wy) \times 10^m + yz \end{aligned}$$

## ● 큰 정수 곱셈

- ✓ 문제: 2개의 큰 정수  $u$ 와  $v$ 를 곱하라
- ✓ 입력: 큰 정수  $u$ 와  $v$ , 크기  $n$
- ✓ 출력:  $\text{prod}(u$ 와  $v$ 의 곱)

```
large_integer prod(large_integer u, large_integer v) {  
    large_integer x, y, w, z;  
    int n, m;  
  
    n = maximum(u의 자리수, v의 자리수);  
    if(u == 0 || v == 0) return 0;  
    else if( n <= threshold)  
        return 일반적인 방법으로 구한 u × v ;  
    else{  
        m = ⌊n/2⌋;  
        x = u divide 10m; y = u mod 10m;  
        w = v divide 10m; z = v mod 10m;  
        return prod(x, w) × 102m +  
            (prod(x, z)+prod(w, y)) × 10m + prod(y, z);  
    }  
}
```

- prod 최악의 경우 시간복잡도 분석:

- ✓ 단위연산: 덧셈, 뺄셈,  $\text{divide } 10^m, \text{mod } 10^m, \times 10^m$
- ✓ 입력크기: 정수의 자리수,  $n$
- ✓  $n$ 이 2의 거듭제곱 형태라고 가정
- ✓ 덧셈, 뺄셈,  $\text{divide } 10^m, \text{mod } 10^m, \times 10^m$  에 있는 1차시간 연산은 모두  $cn$ 으로 표시

$$W(n) = 4W\left(\frac{n}{2}\right) + cn, \quad n > s \text{ 이고, } n \text{이 2의 거듭제곱인 경우}$$

$$W(s) = 0$$

$s = \text{threshold}$ 보다 작거나 같은 문제크기.  $W(s)$ 의 단위연산 횟수는 0

$$W(n) \in \Theta(n^{\lg 4}) = \Theta(n^2)$$

Appendix Theorem B.5  
The Master Theorem

● 개선된 방법:

- ✓ 이전 방법에서는  $xw$ ,  $xz+yw$ ,  $yz$ 의 계산 필요 ➔ 4회의 곱셈
- ✓ 개선방법:  $r$  계산 추가

$$r = (x+y) \times (w+z) = xw + (xz+yw) + yz$$

$$xz+yw = r - \textcircled{1} xw - \textcircled{3} yz$$

$$\begin{aligned} u \times v &= (x \times 10^m + y) \times (w \times 10^m + z) \\ &= xw \times 10^{2m} + (xz+yw) \times 10^m + yz \end{aligned}$$

(1) ① 계산 수행

(2) ②, ③ 계산 :  $xw$ ,  $yz$  구함

(3)  $r$ 의 값에서 ②, ③의 계산 결과를 빼줌 :  $xz+yw$  구함

- 결과적으로  $xw$ ,  $xz+yw$ ,  $yz$ 을 계산하는데, 덧셈/뺄셈의 회수는 증가지만, 곱셈은 3회 필요

## ● 큰 정수 곱셈2

- ✓ 문제: 2개의 큰 정수  $u$ 와  $v$ 를 곱하라
- ✓ 입력: 큰 정수  $u$ 와  $v$ , 크기  $n$
- ✓ 출력:  $\text{prod2}(u$ 와  $v$ 의 곱)

```
large_integer prod2(large_integer u, large_integer v) {  
    large_integer x, y, w, z, r, p, q;  
    int n, m;  
    n = maximum(u의 자리수, v의 자리수);  
    if(u == 0 || v == 0) return 0;  
    else if( n <= threshold)  
        return 일반적인 방법으로 구한 u × v ;  
    else{  
        m = ⌊n/2⌋;  
        x = u divide  $10^m$ ; y = u mod  $10^m$ ;  
        w = v divide  $10^m$ ; z = v mod  $10^m$ ;  
        r = prod2(x+y, w+z);  
        p = prod2(x, w);  
        q = prod2(y, z);  
        return  $p \times 10^{2m} + (r-p-q) \times 10^m + q$ ;  
    }  
}
```

● prod2 최악의 경우 시간복잡도 분석:

✓  $\text{prod2}(x+y, w+z)$      $n/2 \leq \text{입력크기} \leq n/2+1$

✓  $\text{prod2}(x, w)$                        $n/2$

✓  $\text{prod2}(y, z)$                        $n/2$

$3W(\frac{n}{2}) + cn \leq W(n) \leq 3W(\frac{n}{2} + 1) + cn$ ,  $n > s$ 이고,  $n$ 이 2의 거듭제곱인 경우

$W(s) = 0$

$W(n) \in \Theta(n^{\lg 3}) \approx \Theta(n^{1.58})$

# 임계값 결정

- ✓ divide-and-conquer 방법에서 큰 문제를 어느 크기의 문제가 될 때까지 분할 할 것인가
- ✓ optimal threshold value를 결정하는 방법
- ✓ 문제 크기가 줄어들면 재귀호출을 계속 수행하는 것 보다는 다른 알고리즘을 활용하는 것이 효과적
- ✓ mergesort2의 분할하고 재합병하는데 걸리는 시간(running time):  $32n \mu s$ 로 가정

$$W(n) = W(\lfloor \frac{n}{2} \rfloor) + W(\lfloor \frac{n}{2} \rfloor) + 32n \mu s, \quad W(1) = 0 \mu s$$

- ✓ 단순화 시키면

$$W(n) = 2W(n/2) + 32n \mu s, n > 1 \text{이고}, n \text{이 } 2 \text{의 거듭제곱인 경우}$$

$$W(1) = 0 \mu s$$

$$W(n) = 32n \lg n \mu s$$

- ✓ 교환정렬을 이용하면

$$W(n) = \frac{n(n-1)}{2} \mu s$$



교환정렬을 호출해야하는 최적의 임계점은

$$\frac{n(n-1)}{2} \mu s < 32n \lg n \mu s \quad \text{를 만족하는 } n.$$

이를 풀면  $n < 591$ .

- 그러나 잘못된 분석임.

$W(n) = 32n \lg n \mu s$ 은 문제 크기가 1 이 될 때까지 분할할 경우의 복잡도

[예 2.7]정확한 분석은 다음의 두 식이 같은 값을 갖는  $t$ 를 찾아야 함.

$$W(n) = \begin{cases} \frac{n(n-1)}{2} \mu s & (n \leq t) \\ W(\lfloor \frac{n}{2} \rfloor) + W(\lceil \frac{n}{2} \rceil) + 32n \mu s & (n > t) \end{cases}$$

$$W(\lfloor \frac{t}{2} \rfloor) + W(\lceil \frac{t}{2} \rceil) + 32t = \frac{t(t-1)}{2}$$

$$\frac{\lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor - 1)}{2} + \frac{\lceil \frac{t}{2} \rceil (\lceil \frac{t}{2} \rceil - 1)}{2} + 32t = \frac{t(t-1)}{2}$$

$t$ 가 threshold이므로  $n = t/2$ 인 경우는  $t(t-1)/2$  사용

교환정렬을 사용하는  
것이 분할하는 것과 같은  
효율이 되는 지점

$t$ 가 짝수;  $t = 128$

홀수;  $t = 128.008$

[결론] 최적  $t = 128$

# 분할정복을 사용하지 말아야 하는 경우

- 크기가  $n$ 인 입력이 2개 이상의 조각으로 분할되며, 분할된 부분들의 크기가 거의  $n$ 에 가깝게 되는 경우  $\Rightarrow$  시간복잡도: 지수(exponential) 시간

$$(ex) \quad T(n) = 2T(n-1) + n$$

- 크기가  $n$ 인 입력이 거의  $n$ 개의 조각으로 분할되며, 분할된 부분의 크기가  $n/c$ 인 경우. 여기서  $c$ 는 상수이다.  $\Rightarrow$  시간복잡도:  $\Theta(n^{\lg n})$

$$(ex) \quad T(n) = nT(n/2) + n$$

# 도사 정리(The Master Theorem)

- $a$ 와  $b$ 를 1보다 큰 상수라 하고,  $f(n)$ 을 어떤 함수라 하고, 음이 아닌 정수  $n$ 에 대해서 정의된 재현식  $T(n)$ 이 다음의 형태를 이룬다고 하자.

$$T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$$

그러면  $T(n)$ 은 다음과 같이 점근적인 한계점(asymptotic bound)을 가질 수 있다.

1. 어떤 상수  $\varepsilon > 0$ 에 대해서, 만약  $f(n) \in O(n^{\log_b a - \varepsilon})$ 이면,  $T(n) \in \Theta(n^{\log_b a})$
2. If  $f(n) \in \Theta(n^{\log_b a})$ , then  $T(n) \in \Theta(n^{\log_b a} \log n)$ .
3. 어떤 상수  $\varepsilon > 0$ 에 대해서, 만약  $f(n) \in \Omega(n^{\log_b a + \varepsilon})$  이고, 어떤 상수  $c < 1$  과 충분히 큰 모든  $n$ 에 대해서,  $a \times f\left(\frac{n}{b}\right) \leq c \times f(n)$ 이면,  
 $T(n) \in \Theta(f(n))$ .

여기서  $\frac{n}{b}$ 은  $\lfloor \frac{n}{b} \rfloor$ 로 여겨도 되고,  $\lceil \frac{n}{b} \rceil$ 으로 여겨도 된다.

# 도사정리 적용의 예

- $T(n) = 9T\left(\frac{n}{3}\right) + n$

여기서  $a = 9, b = 3, f(n) = n$ 이고,  $n^{\log_b a} = n^{\log_3 9} \in \Theta(n^2)$  이므로,  $\varepsilon = 1$ 일 때,  $f(n) \in O(n^{\log_3 9 - \varepsilon})$ 이라고 할 수 있다. 도사정리 1번을 적용하면,  $T(n) \in \Theta(n^{\log_3 9}) = \Theta(n^2)$  이 된다.

- $T(n) = T\left(\frac{2n}{3}\right) + 1$

여기서  $a = 1, b = \frac{3}{2}, f(n) = 1$ 이고,  $n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 \in \Theta(1)$  이므로,  $f(n) \in \Theta(1)$  이라고 할 수 있다. 도사정리 2번을 적용하면,  
 $T(n) \in \Theta(\lg n)$  이 된다.

# 도사정리 적용의 예

- $T(n) = 3T\left(\frac{n}{4}\right) + n \lg n$

여기서  $a = 3, b = 4, f(n) = n \lg n$ 이고,  $n^{\log_b a} = n^{\log_4 3} \in O(n^{0.793})$ 이므로,  $\varepsilon \approx 0.2$ 일 때,  $\varepsilon = 1$ 일 때,  $f(n) \in \Omega(n^{\log_4 3 + \varepsilon})$  이라고 할 수 있다. 도사정리 3번을 적용할 수 있는지 보기 위해서, 충분히 큰  $n$ 에 대해서,  $3f(\frac{n}{4}) \leq c \times f(n)$ 이 성립하는 1보다 작은  $c$ 가 존재하는가를 보아야 한다. 여기서,  $c = \frac{3}{4}$ 이면,  $3\frac{n}{4} \lg(\frac{n}{4}) \leq \frac{3}{4} n \lg n$ 은 충분히 큰  $n$ 에 대해서 항상 성립한다. 따라서  $T(n) \in \Theta(n \lg n)$ 이 된다.

- $T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$

여기서  $a = 2, b = 2, f(n) = n \lg n$ 이고,  $n^{\log_b a} = n^{\log_2 2} \in \Theta(n)$ 이므로,  $f(n) \in \Omega(n^{\log_2 2 + \varepsilon})$  이라고 할 수 있다. 여기서 도사정리 3번을 적용할 수 있는지 보기 위해서, 충분히 큰  $n$ 에 대해서,  $2f(\frac{n}{2}) \leq c \times f(n)$  이 성립하는 1보다 작은  $c$ 가 존재하는가를 보아야 한다. 그러나,  $2\frac{n}{2} \lg(\frac{n}{2}) \leq cn \lg n$ 에서 충분히 큰  $n$ 에 대해서 항상 성립하는  $c$ 는 없다. 왜냐하면, 위의 식을 정리하면  $\frac{\lg n - 1}{\lg n} \leq c$ 가 되고, 어떠한  $c$ 를 선택하더라도 이 부등식은 성립할 수 없다. 따라서 도사정리를 이용하여 해를 구할 수 없다. 이런 경우는 다음의 도사보조정리를 이용하여 해를 구할 수 있다.

# 도사보조정리

- $T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$

에서,  $k \geq 0$ 인 어떤  $k$ 에 대해서  $f(n)$ 이  $\Theta(n^{\log_b a} \lg^k n)$  이면  
 $T(n) \in \Theta(n^{\log_b a} \lg^{k+1} n)$ 이 된다. (증명 생략)

- 도사보조정리의 적용의 예

위의 보기 4번의 해는  $f(n) \in \Theta(n \lg n)$ 이므로  $T(n) \in \Theta(n \lg^2 n)$  이  
된다.

이 절에서 공부한 도사정리는 재현식을 푸는데 상당히 유용하  
게 쓰인다.

# Theorem B.5 in Appendix B.

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k \quad (n > 1 \text{ 이고, } n \text{ 이 } b \text{ 의 거듭제곱이면})$$

$$T(1) = d$$

$$T(n) \in \begin{cases} \Theta(n^k) & (\text{if } a < b^k) \\ \Theta(n^k \lg n) & (\text{if } a = b^k) \\ \Theta(n^{\log_b a}) & (\text{if } a > b^k) \end{cases}$$

(예)

$$\begin{array}{ccc} \boxed{a} & \boxed{b} & \boxed{k} \\ \downarrow & \downarrow & \downarrow \\ T(n) = 8T(n/4) + 5n^2 \\ T(1) = 3 \end{array}$$

$$8 < 4^2 \text{ 이므로 } T(n) \in \Theta(n^2)$$

(예)

$$\begin{array}{ccc} \boxed{a} & \boxed{b} & \boxed{k} \\ \downarrow & \downarrow & \downarrow \\ T(n) = 8T(n/2) + 5n^3 \\ T(1) = 200 \end{array}$$

$$8 = 2^3 \text{ 이므로 } T(n) \in \Theta(n^3 \lg n)$$

(예)

$$\begin{array}{ccc} \boxed{a} & \boxed{b} & \boxed{k} \\ \downarrow & \downarrow & \downarrow \\ T(n) = 9T(n/3) + 5n^1 \\ T(1) = 7 \end{array}$$

$$9 > 3^1 \text{ 이므로 } T(n) \in \Theta(n^{\log_3 9}) = \Theta(n^2)$$

Also,

$$T(n) \leq aT\left(\frac{n}{b}\right) + cn^k \quad (n > 1, n \text{ is a power of } b) \quad T(n) \in \begin{cases} O(n^k) & (\text{if } a < b^k) \\ O(n^k \lg n) & (\text{if } a = b^k) \\ O(n^{\log_b a}) & (\text{if } a > b^k) \end{cases}$$

$$T(n) \geq aT\left(\frac{n}{b}\right) + cn^k \quad (n > 1, n \text{ is a power of } b) \quad T(n) \in \begin{cases} \Omega(n^k) & (\text{if } a < b^k) \\ \Omega(n^k \lg n) & (\text{if } a = b^k) \\ \Omega(n^{\log_b a}) & (\text{if } a > b^k) \end{cases}$$



prod

$$W(n) = 4W\left(\frac{n}{2}\right) + cn, \quad n > s \text{ 이고, } n \text{이 2의 거듭제곱인 경우}$$

$$W(s) = 0$$

$$W(n) \in \Theta(n^{\lg 4}) = \Theta(n^2)$$

prod2

$$3W\left(\frac{n}{2}\right) + cn \leq W(n) \leq 3W\left(\frac{n}{2} + 1\right) + cn, \quad n > s \text{ 이고, } n \text{이 2의 거듭제곱인 경우}$$

$$W(s) = 0$$

$$W(n) \in \Theta(n^{\lg 3}) \approx \Theta(n^{1.58})$$



2장 끝

수고하셨습니다.