# 2장 분할 정복 정리

이진 검색

## ❑ Recursive Binary Search

**Initial Call:**

location( *1*, *n*)

```
public static 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) ;
    }
}
```

## ❑ Worst-Case Time Complexity of Binary Search

- ❑ ***Basic Operation:*** Comparison of x with S[mid]
- ❑ ***Input Size:*** n, the number of items in array
- ❑ ***Assumption:*** $n=2^k$ for some integer $k \geq 0$ → $k = \log_2 n$

k를 n으로 표현

- $W(n) = W(n/2) + 1$ for $n>1$, $W(1) = 1$

$W(2) = 2$

$W(4) = 3$

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

$= ((\ W(n/2^3) + 1\ ) + 1\ )\ + 1$

…

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

k

$= 1 + \boxed{\log_2 n}$

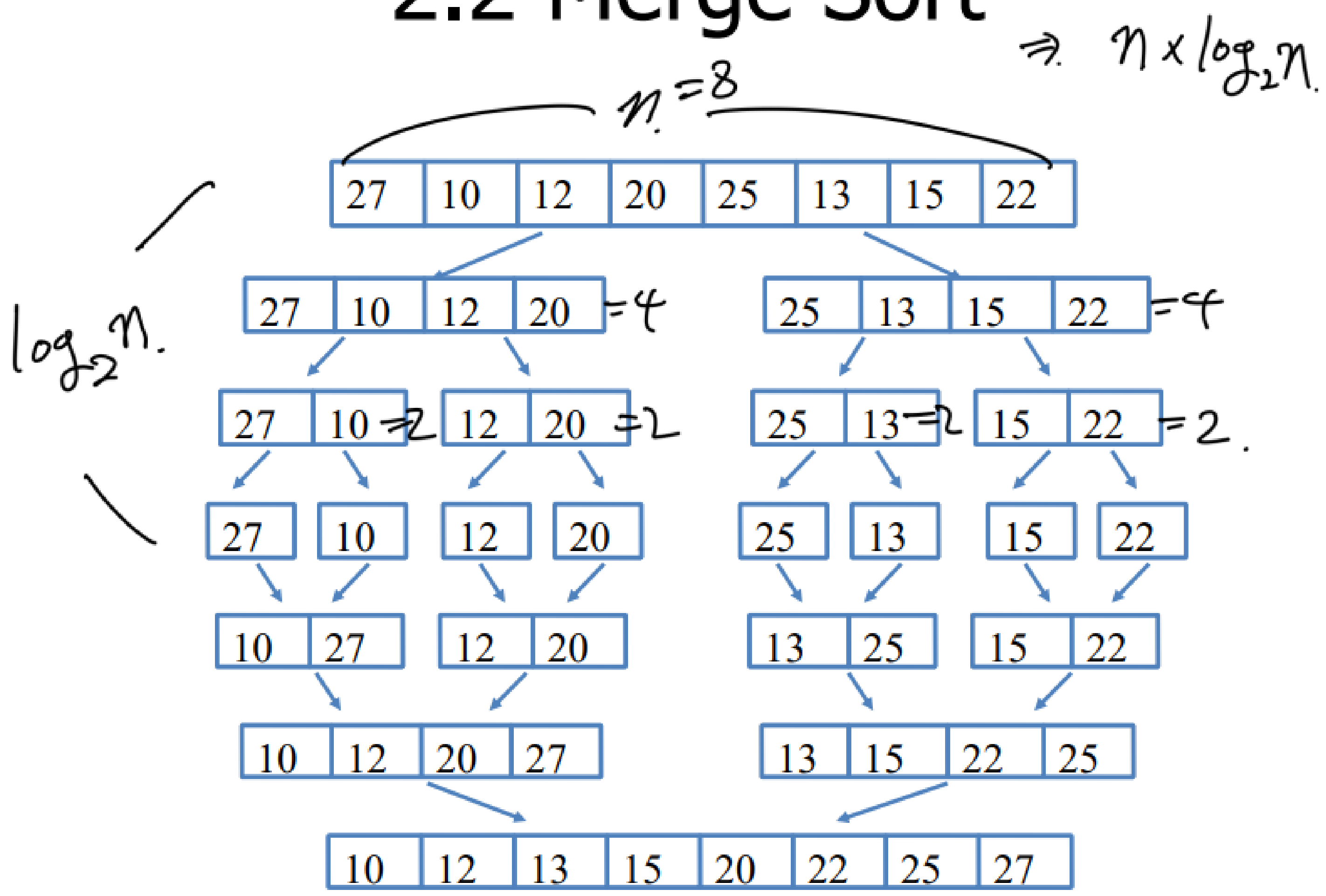
$\rightarrow \theta(\log_2 n)$

---

## Merge Sort

병합 정렬 로직

# 2.2 Merge Sort




분할 과정 슈도 코드

Divide

Conquer

Combine

배열길이

```
public static void mergeSort(int n, keytype[ ] S)
{
    if (n>1) {
        const int h = floor(n/2) , m= n-h;
        keytype[ ]  U = new keytype[1..h],
                    V = new keytype[1..m] ;
        copy S[1:h] to U[1:h] ;
        copy S[h+1:m] to V[1:m] ;
        mergeSort(h,U) ;
        mergeSort(m,V) ;
        merge(h,m,U,V,S) ;
    }
}
```

합치는 과정(Merge) 슈도코드

```
public static void merge(int h, int m, keytype[ ]  U,
                          keytype[ ] V, keytype[ ] S)
{
    index 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++ ;        i는 U배열
                       j는 V배열
    }
    if ( i > h )  i가 U배열 끝에 도달 → V 배열이 남았다는 뜻.
          copy V[ j:m] to S[k:h+m] ;
    else  i≤h : U배열에 복사할 값이 남았다
          copy U[i:h] to S[k:h+m] ;
}
```

### Best Case, Worst Case 시간 복잡도 구하기

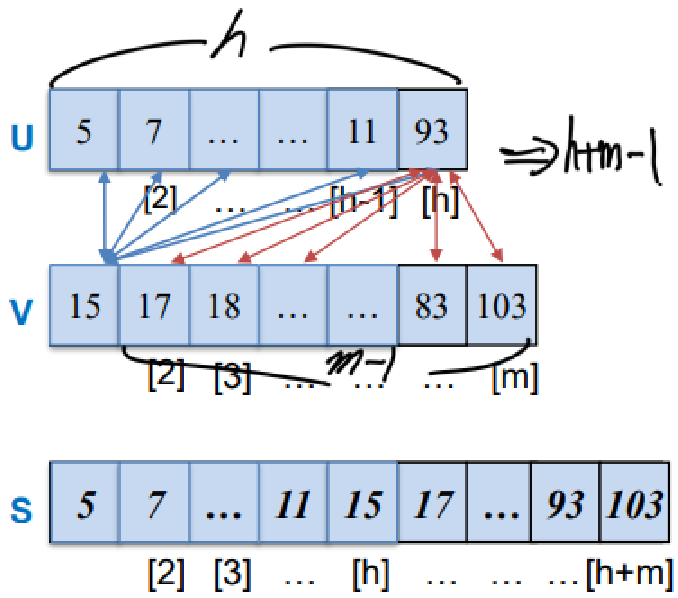
**Best Case** - 비교를 조금하고 한쪽 배열에 많이 남아 바로 복사만 하는 경우 - O(n lg n)

**Worst Case** - 마지막 원소 1개만 남고 모두 비교를 통해서 배열을 채울 경우 - O(n lg n)

→ 마지막 원소 1개만 남고 모두 비교를 통해서 배열 S를 채운 경우.

## ❑ *Worst Case* Time Complexity of **Merge**

-The worst case occurs when every cell in array S (except for the last one) is assigned a number only after a comparison operation




➔ W(h,m) = h + m - 1

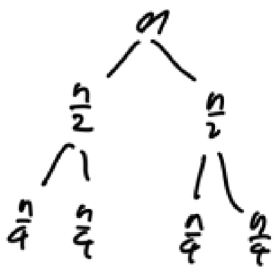
## ❑ *Worst Case* Time Complexity of *MergeSort*

$h+m=n$

*mergeSort*(h,U) ;
*mergeSort*(m,V) ;
*merge*(h,m,U,V,S) ;

➔ $W(n) = W(h) + W(m) + W(h,m)$ ↗ Time to merge

$= W(h) + W(m) + h+m-1$




When ~~$n = 2^k$ for~~ some $k \ge 0$, $h=m=\frac{n}{2}$  $\boxed{k=\log_2 n}$

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

$W(1)=0$

↳ 배열사이즈 1이면 끝.

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

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

$\therefore W(n) = \boxed{n \cdot \lg n - (n-1)} \in \Theta(n \lg n)$

12

여기서 n은 노드수 (배열의 사이즈)

---

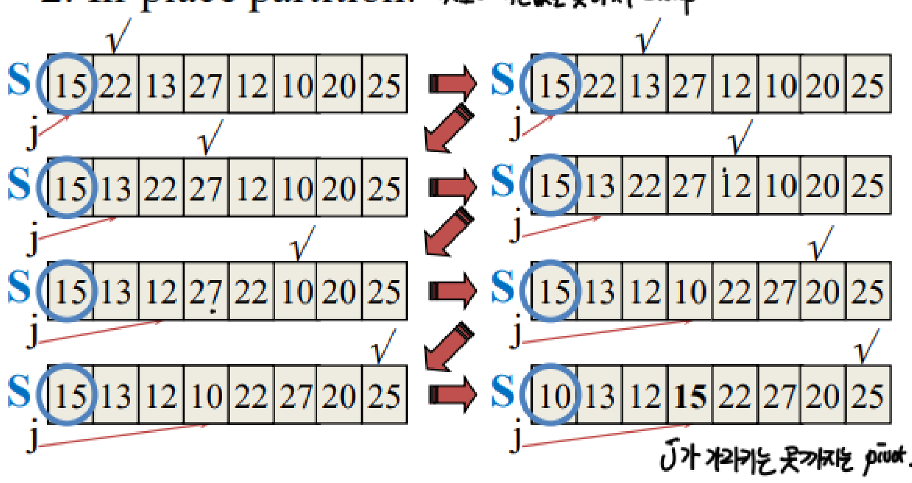
## Quick Sort

알고리즘 로직

파티션 알고리즘이 중요함!

## ❑ *Partition* Algorithm

2. In-place partition: 피봇보다 작은값을 찾아서 Swap




i: loop index for S
($2 \le i \le n$)

j: index for smaller items (j++)

j가 가리키는 곳까지는 pivot 보다 작다.

➔ does **not** require extra space and time to copy back ∴ pivot의 위치와 S[j]를 스왑.
S[low]

```
public static void quickSort(index low,
                             index high)
{
      index pivotPoint ;

      if (high > low)
      {
          pivotPoint = partition(low, high) ;
          quickSort(low, pivotPoint-1) ;
          quickSort( pivotPoint+1 , high ) ;
      }
}
```

## ❑ (In-Place) *Partition* Algorithm

(high-low)-1 = n-1

low start high

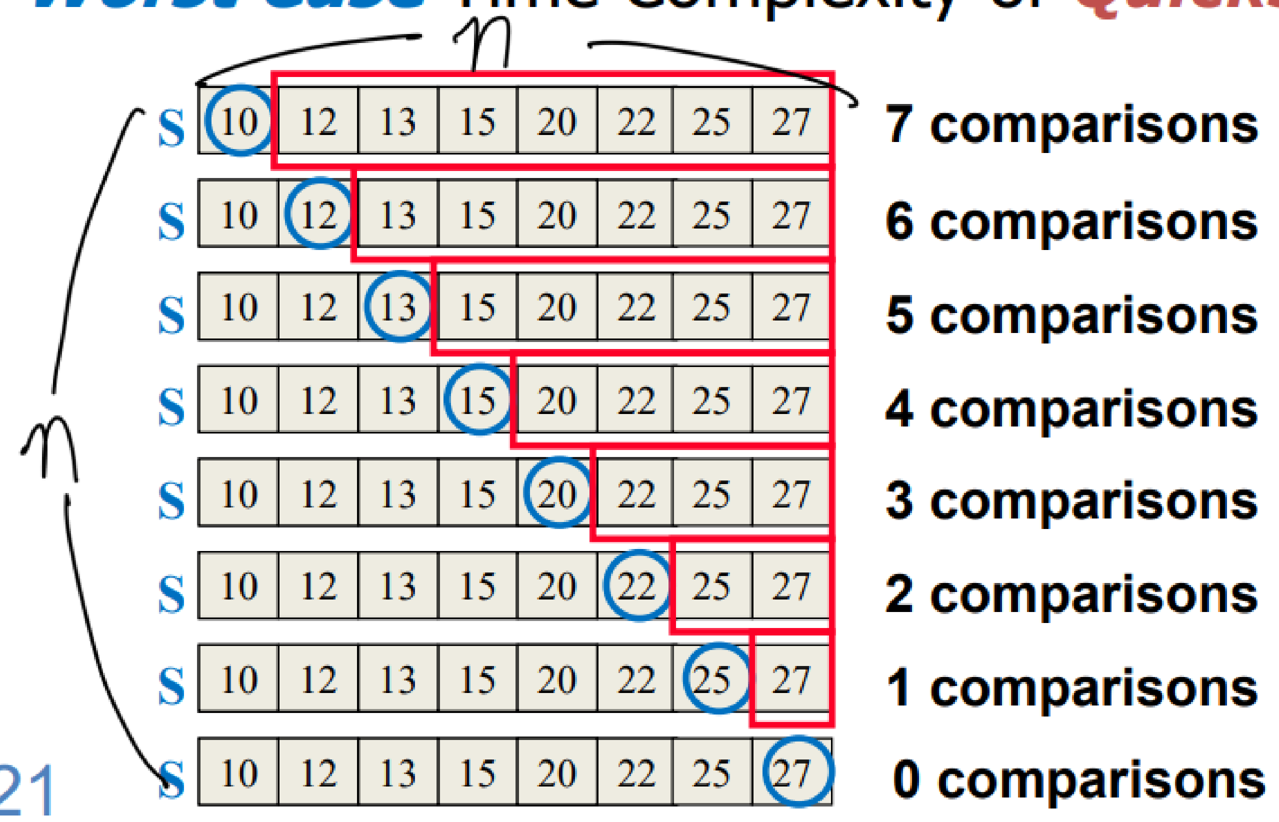
```
public static index partition(index low, index high) {
    index i, j, pivotPoint ;   keytype pivotItem ;
    pivotItem = S[low] ;
    j = low ;
    for ( i = low + 1; i <= high; i++ )        → T(high-low-1)
        if (S[i] < pivotItem ) {   피봇보다 작은값 나오면 S[++j]와 교환
            exchange S[i] and S[++j] ;
        }                          ↑ 피봇보다 작은 값
    pivotPoint = j ;   정해진 피봇값 → J
    exchange S[ low ] and S[pivotPoint] ;
    return pivotPoint;  ↑ 기준의 피봇   ↑ 정해진 피봇
```

### Average Case, Worst Case

#### Worst Case

이미 정렬되어있을 경우

## ❑ *Worst Case* Time Complexity of *QuickSort*



→ 이미 정렬되어 있어 분할된 배열의 크기가 (1, n-1)로 나누어질 때

## ❑ Worst Case Time Complexity of QuickSort

When the array is already sorted in non-decreasing order:

$$T(n) = \underbrace{n-1}_{\text{Time to partition}} + \underbrace{T(0)}_{\text{Time to sort the left subarray}} + \underbrace{T(n-1)}_{\text{Time to sort the right subarray}}$$

← 이미 정렬되어있으면 0

$$\rightarrow T(n) = T(n-1) + n-1$$
$$= T(n-2) + n-2 + n-1$$
$$= T(n-3) + n-3 + n-2 + n-1$$
$$\cdots$$
$$= T(0) + 0 + 1 + 2 + \ldots + n-2 + n-1 = \boxed{\frac{n(n-1)}{2}} = n^2$$

3/25

Average Case 세타 (n lg n)

## ❑ Average Case Time Complexity of QuickSort

| S | 34 | … | … | 15 | … | … | … | 53 |
|---|---|---|---|---|---|---|---|---|
| | 1 | | | i | | | | n |

$\underbrace{\quad}_{i-1}$ $\underbrace{\quad}_{n-i}$

$$\rightarrow A(n) = \sum_{i=1}^{n} p_i [\underbrace{A(i-1)}_{\text{Average time to sort the left subarray when pivotPoint is } i} + \underbrace{A(n-i)}_{\text{Average time to sort the right subarray when pivotPoint is } i} + \underbrace{(n-1)}_{\text{Time to partition}}]$$

## 임계값 결정

교환 정렬이 작은 n에 대해서는 병합 정렬보다 빠르므로 어느 정도 분할 되면 교환 정렬을 쓴다.

## ❑ A Modified MergeSort using ExchangeSort

- the ExchangeSort is called when the subarray size becomes less than some threshold $t$

* Suppose that it takes $\boxed{\alpha n\ \mu s}$ to take care of the overheads.

$$W(n) = \begin{cases} n(n-1)/2\ \mu s & n < t \\ W(n/2) + W(n/2) + \alpha n\ \mu s & n \ge t \end{cases}$$

➔ The threshold t is obtained when the two expressions are the same.

## ❑ A Modified MergeSort using ExchangeSort

$t(t-1)/2 = W(t/2) + W(t/2) + \alpha t$ ···> t를 구하기 위해

$\rightarrow t(t-1)/2 = 2W(t/2) + \alpha t$

Since $t/2 < t$, we have

$t(t-1)/2 = \underline{2 \cdot (t/2) \cdot (t/2 - 1)/2} + \alpha t$

$\rightarrow t(t-1)/2 = t(t-2)/4 + \alpha t$

$\boxed{\rightarrow t = 4\alpha}$

∴ That is , in MergeSort, call ExchangeSort when n becomes smaller than 4 α .

---

# 분할 정복을 사용하면 안되는 경우

1. N을 쪼개서N N과 비슷한 2개 이상의 케이스가 나올 경우 (피보나치)
2. N을 나눴을 때 그 N의 크기만큼 나눠질 때