# 정렬 알고리즘

# 목차

# 버블 정렬(bubble sort)

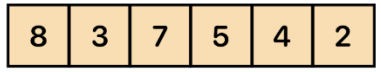
- 정렬 과정에서 원소의 이동이 마치 거품이 수면 위로 올라오는 것과 같다고 해서 붙여진 이름

방법

1. 현재 원소와 바로 다음 원소의 크기를 비교한다.
2. 현재 원소가 다음 원소의 크기보다 크면 자리 교환
3. 다음 원소로 이동하여 또 비교 후 자리 교환

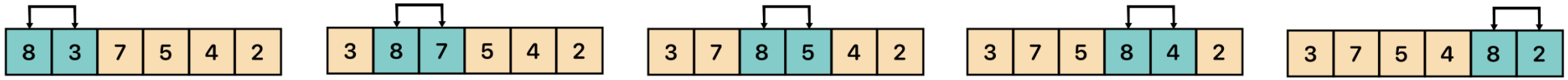
# 버블 정렬

초기 배열

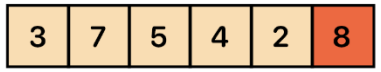



round 1

비교 및 교환

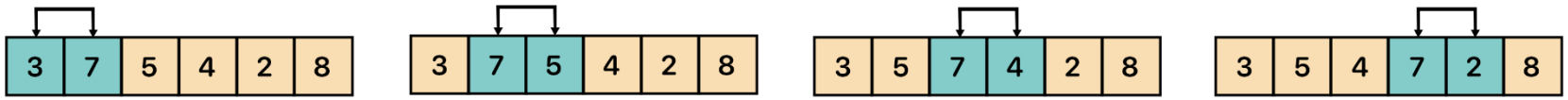



1회전 결과




round 2

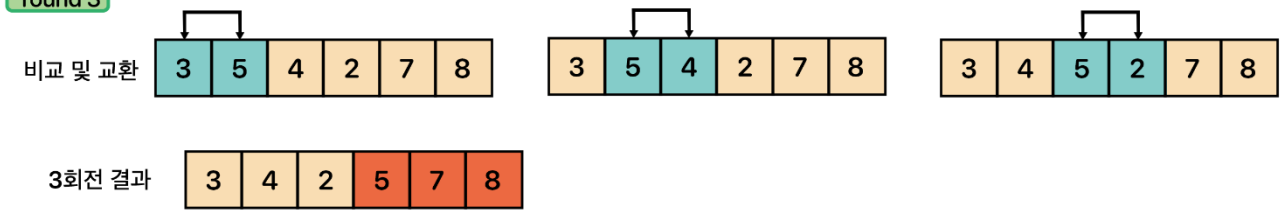
비교 및 교환



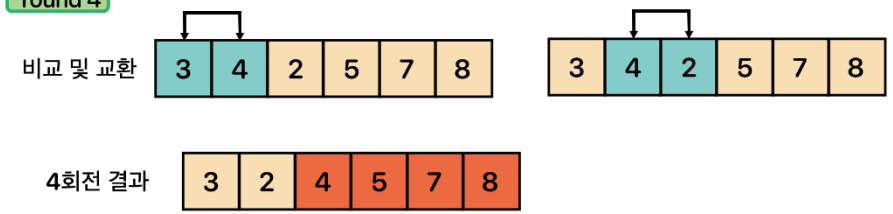

2회전 결과

# 버블 정렬

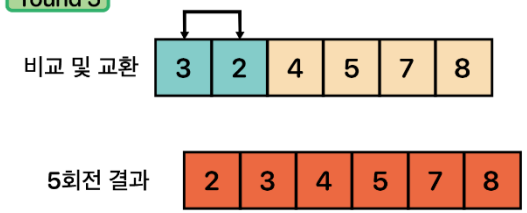
round 3




round 4




round 5



# 버블 정렬 코드

```java
public static void bubble(int[] arr){
    for(int i=0; i<arr.length; i++){
        for(int j=0; j<arr.length-i-1; j++){
            if(arr[j]>arr[j+1]){
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
        }
    }
}
```

# 버블 정렬 시간 복잡도

(n-1) + (n-2) + … + 3 + 2 + 1

$= \sum_{i=1}^{(n-1)} i$

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

$$\frac{n(n-1)}{2} \leq \frac{n^2}{2}$$

따라서 시간 복잡도는 $O(n^2)$

# 선택 정렬(select sort)

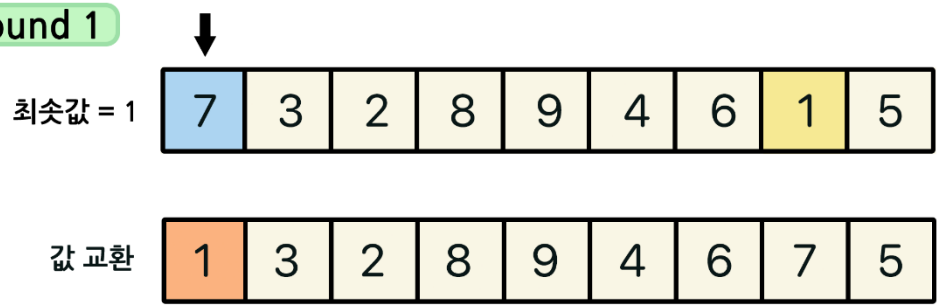
방법

1. 주어진 배열의 최소값을 찾는다.
2. 최소값을 첫번째 자리와 교환
3. 다음 원소도 똑같이 반복

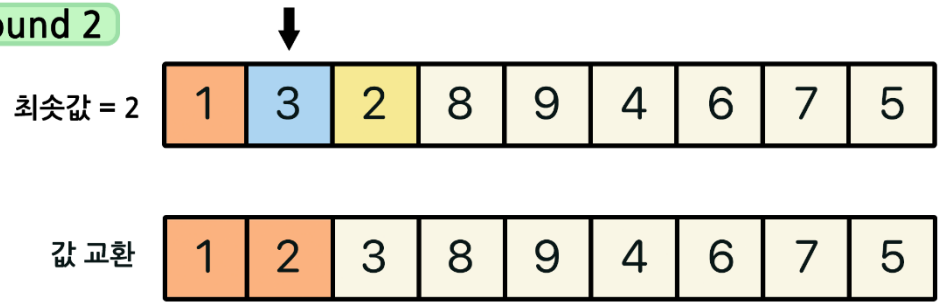
# 선택 정렬

초기 배열

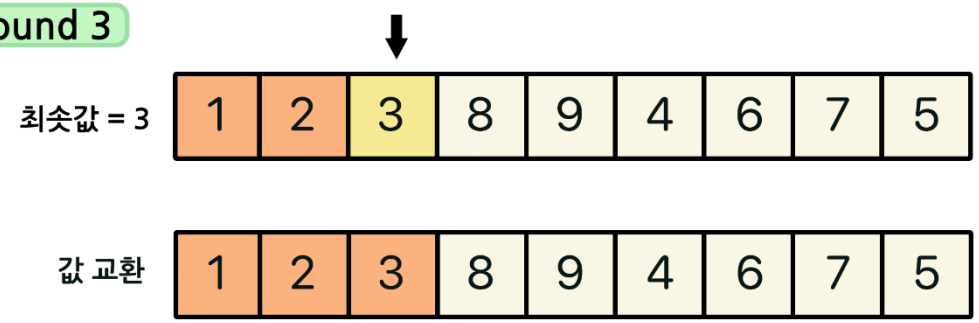

round 1



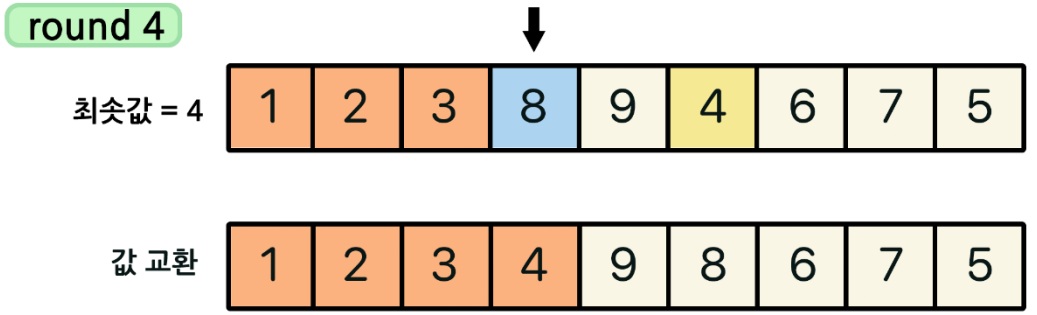

round 2



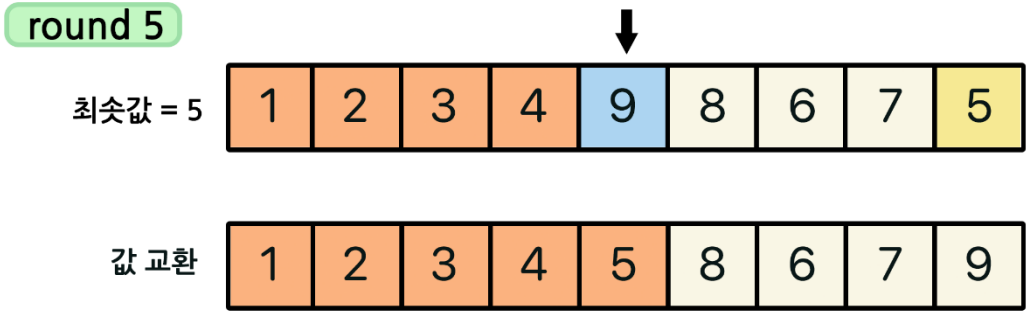

round 3




round 4

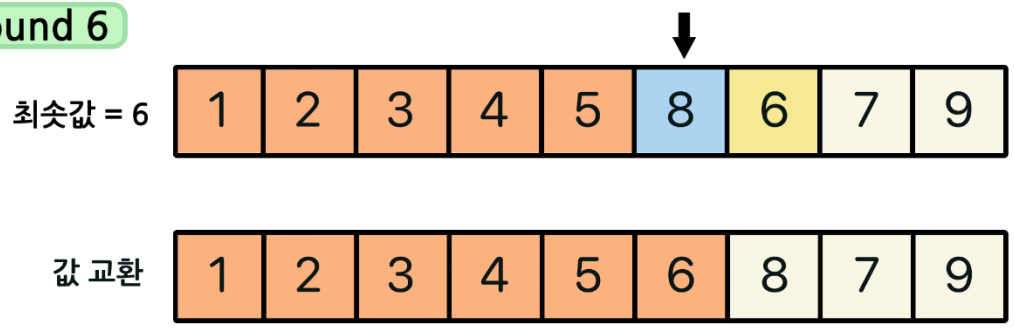



round 5

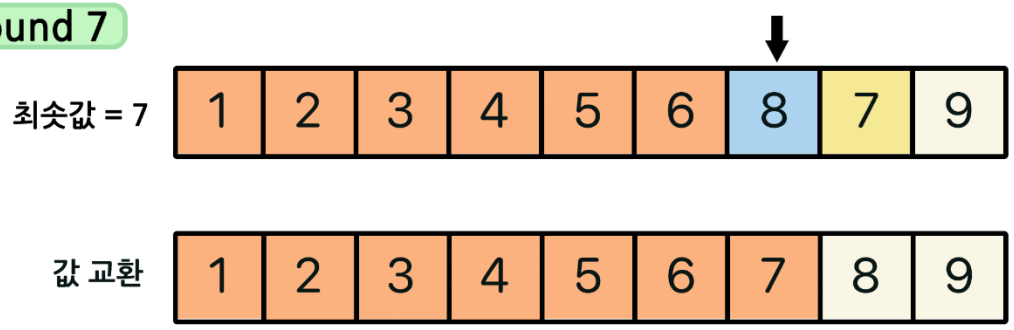


# 선택 정렬

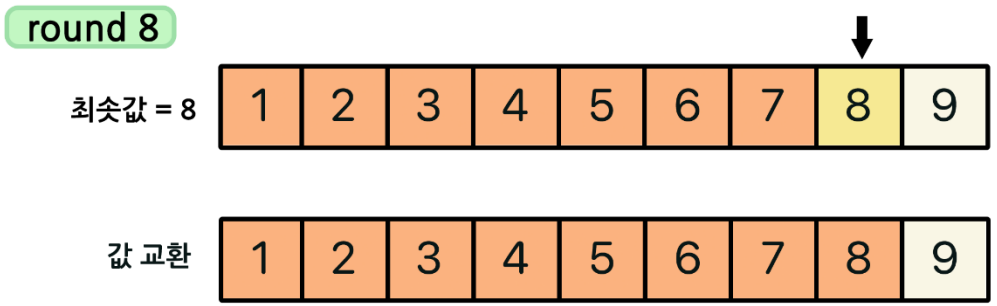
round 6




round 7




round 8




# 선택 정렬 코드

```java
public static void select(int[] arr){
    for(int i=0; i<arr.length; i++){
        int min_index = i;
        for(int j=i+1; j<arr.length; j++){
            if(arr[j]<arr[min_index]){
                min_index = j;
            }
        }
        int tmp = arr[min_index];
        arr[min_index] = arr[i];
        arr[i] = tmp;
    }
}
```

# 선택 정렬 시간 복잡도

(n-1) + (n-2) + … + 3 + 2 + 1

$= \sum_{i=1}^{(n-1)} i$

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

$$\frac{n(n-1)}{2} \leq \frac{n^2}{2}$$

따라서 시간 복잡도는 $O(n^2)$

# 삽입 정렬(insertion sort)

방법

1. 타겟이 되는 원소를 앞 원소와 비교한다.
2. 타겟을 옳은 자리에 삽입한다.
3. 다음 타겟도 똑같이 반복

# 삽입 정렬

초기 배열


round 1

target

7과 3 비교


값 교환


round 2

target

7과 2 비교


값 교환


round 3

target

7과 8 비교


round 4

target

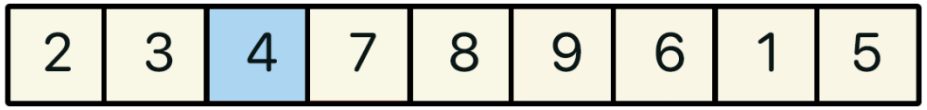
8과 9 비교


round 5

target

4와 9 비교


값 교환



# 삽입 정렬

round 6

target

9와 6 비교

| 2 | 3 | 4 | 7 | 8 | 9 | 6 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|

값 교환

| 2 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|

round 7

target

9와 1 비교

| 2 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|

값 교환

| 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 5 |
|---|---|---|---|---|---|---|---|---|

round 8

target

9와 5 비교

| 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 5 |
|---|---|---|---|---|---|---|---|---|

값 교환

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|

정렬 결과

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|

# 삽입 정렬 코드

```java
public static void insertion(int[] arr){
    for(int i=1; i<arr.length ; i++){
        int target = arr[i];
        int j = i-1;
        while(j>=0 && arr[j]>target){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = target;
    }
}
```

# 삽입 정렬 시간 복잡도

1 + 2 + 3 + … + (n-2) + (n-1)

$= \sum_{i=1}^{(n-1)} i$

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

$$\frac{n(n-1)}{2} \leq \frac{n^2}{2}$$

따라서 시간 복잡도는 $O(n^2)$

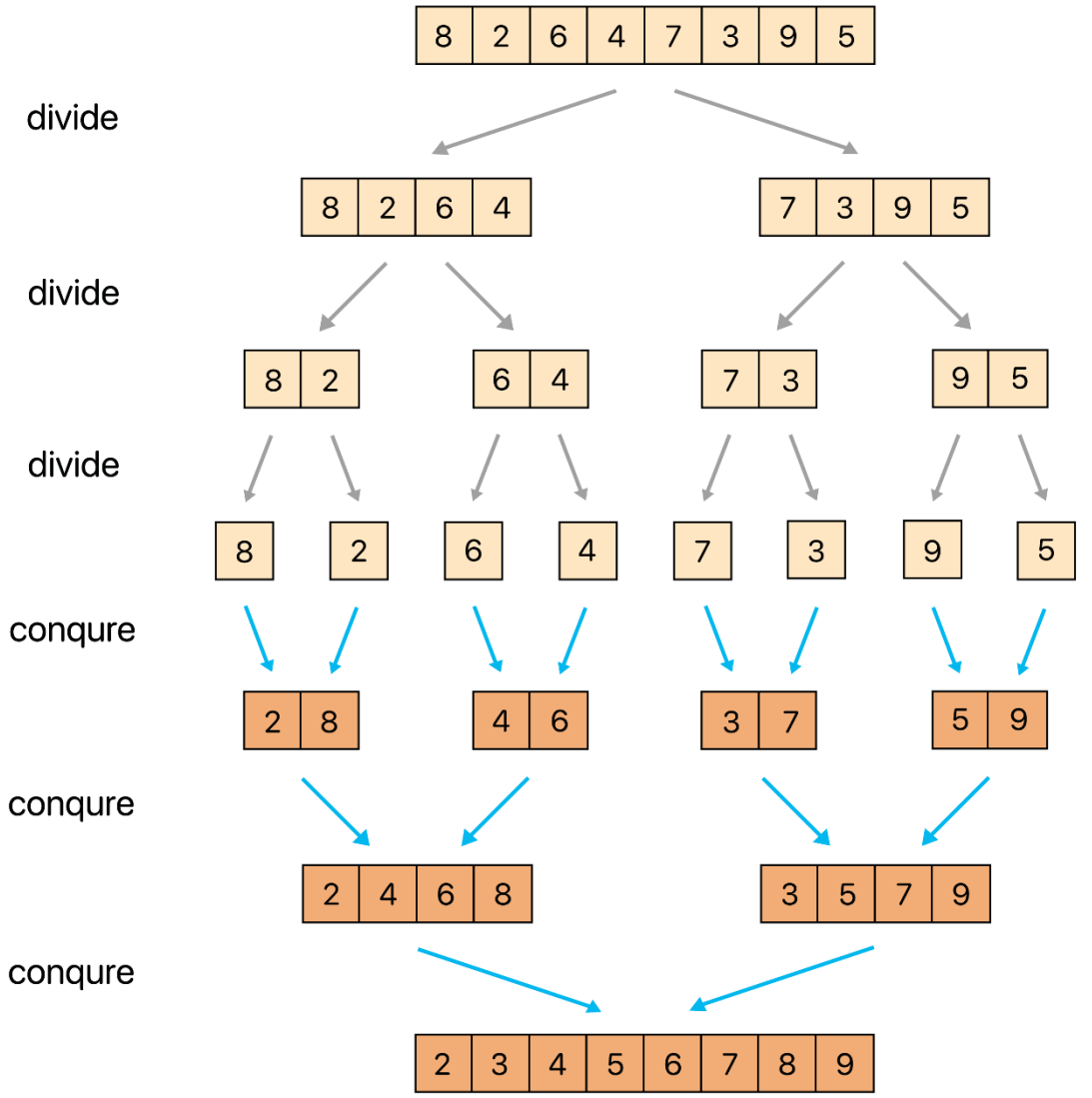
# 병합 정렬(merge sort)

방법

1. 주어진 배열을 최소 단위로 분할한다.
2. 분할 된 배열을 정렬한다.
3. 합치며 다시 정렬한다.

※ 병합 정렬을 합병 정렬이라고도 함.

# 병합 정렬


8 2 6 4 7 3 9 5
divide
8 2 6 4
7 3 9 5
divide
8 2
6 4
7 3
9 5
divide
8
2
6
4
7
3
9
5
conqure
2 8
4 6
3 7
5 9
conqure
2 4 6 8
3 5 7 9
conqure
2 3 4 5 6 7 8 9

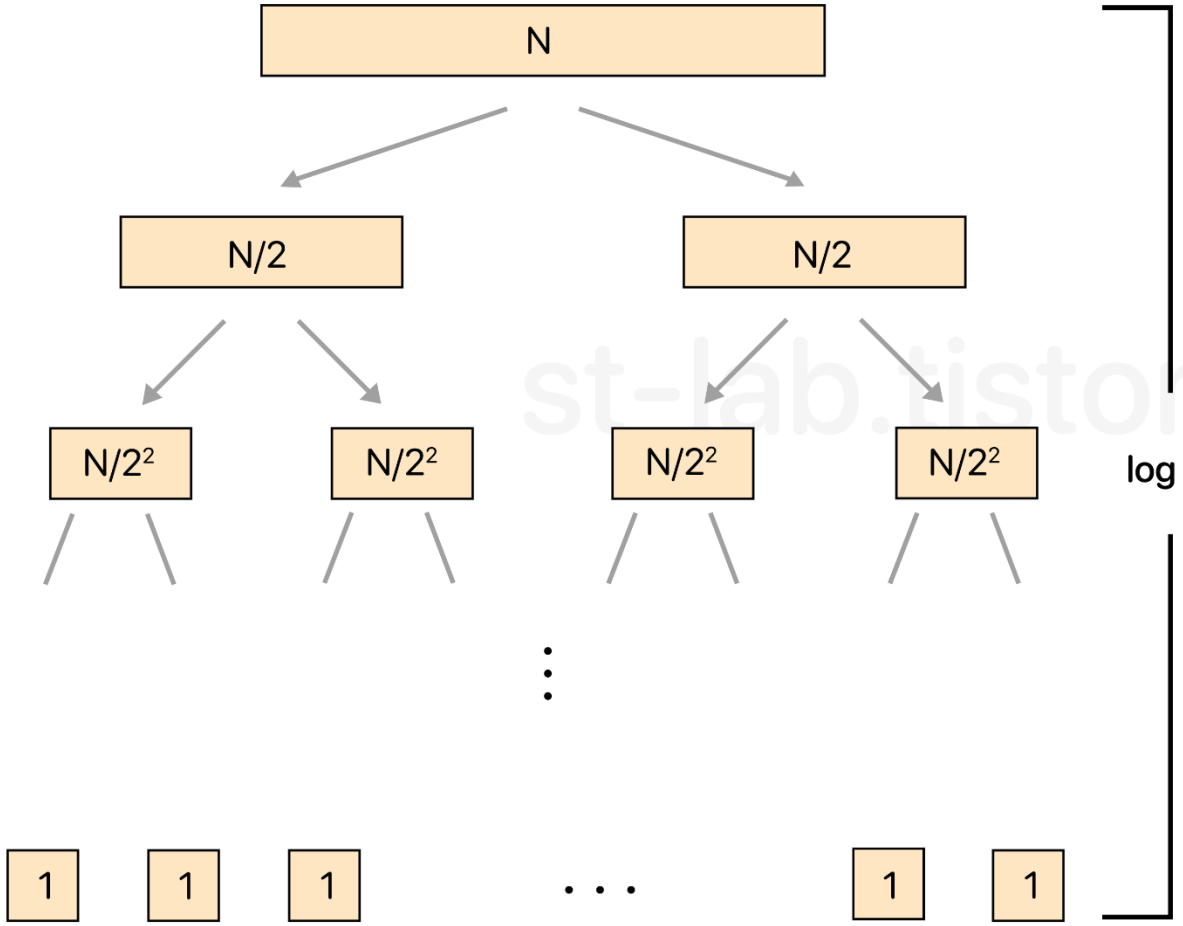
# 병합 정렬 코드

```java
public static void merge(int[] arr, int start, int end){
    if(start<end){
        int mid = (start+end)/2;
        merge(arr, start, mid);
        merge(arr, start: mid+1, end);
        merge1(arr, start, mid, end);
    }
}
```

```java
public static void merge1(int[] arr, int start, int mid, int end){
    int[] tmp = new int[arr.length];
    for(int i=0; i<tmp.length; i++){
        tmp[i] = arr[i];
    }

    int part1 = start;
    int part2 = mid+1;
    int index = 0;
    while (part1 <= mid && part2 <= end){
        if(tmp[part1]<=tmp[part2]){
            arr[index] = tmp[part1];
            part1++;
        }else{
            arr[index] = tmp[part2];
            part2++;
        }
        index++;
    }
    for(int i=0; i<mid-part1; i++){
        arr[index+i] = tmp[part1+i];
    }
}
```

# 병합 정렬 시간 복잡도




노드의 개수 : 1
노드의 크기 : N

노드의 개수 : 2
노드의 크기 : $N/2^1 = N/2$

비교횟수 = N

노드의 개수 : $2^2 = 4$
노드의 크기 : $N/2^2 = N/4$

비교횟수 = N

노드의 개수 : $2^i$
노드의 크기 : $N/2^i$

비교횟수 = $2^i * (N / 2^i)$
= N

노드의 개수 : N
노드의 크기 : 1

비교횟수 = N

# 병합 정렬 시간 복잡도

## 노드의 개수

$$N = \sum_{k=1}^{d} a_k = 1 + 2 + 4 + \cdots + 2^{d-1}$$

$$= 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{d-1}$$

$$= \frac{1(2^d - 1)}{2 - 1}$$

$$= 2^d - 1$$

## 트리의 높이

$$N = 2^d - 1$$

$$N + 1 = 2^d$$

$$\log_2(N + 1) = \log_2 2^d$$

$$\log_2(N + 1) = d$$

$$h = d - 1 = \log_2(N + 1) - 1$$

$$h = \log_2(N)$$

# 병합 정렬 시간 복잡도

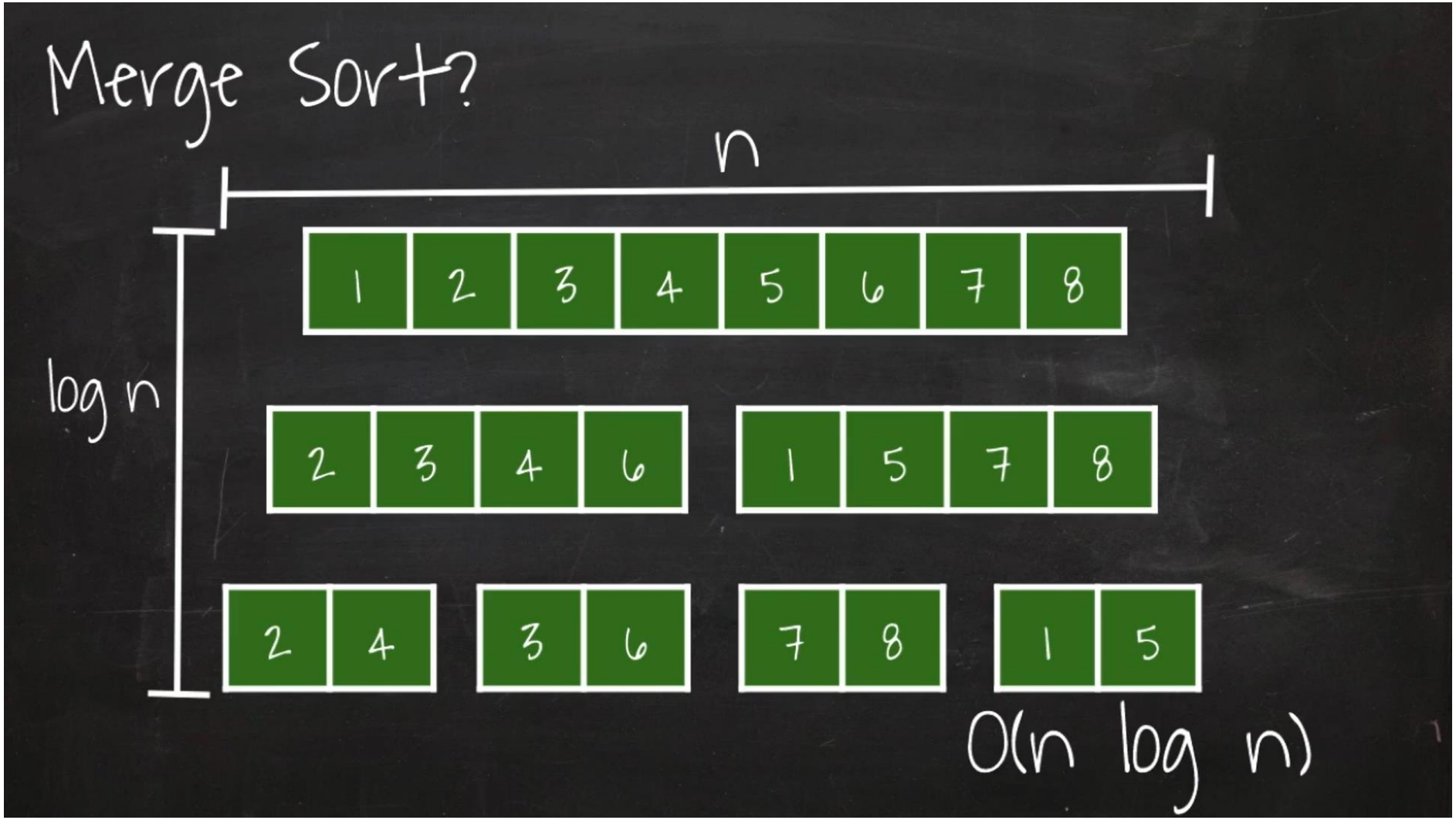
한 레벨에서 비교 작업에 대한 시간 복잡도

$$O\left(2^n * \frac{n}{2^n}\right) = O(n)$$

트리의 높이
$O(\log n)$

따라서 시간 복잡도는 $O(n) * O(\log n)$

# 병합 정렬 시간 복잡도

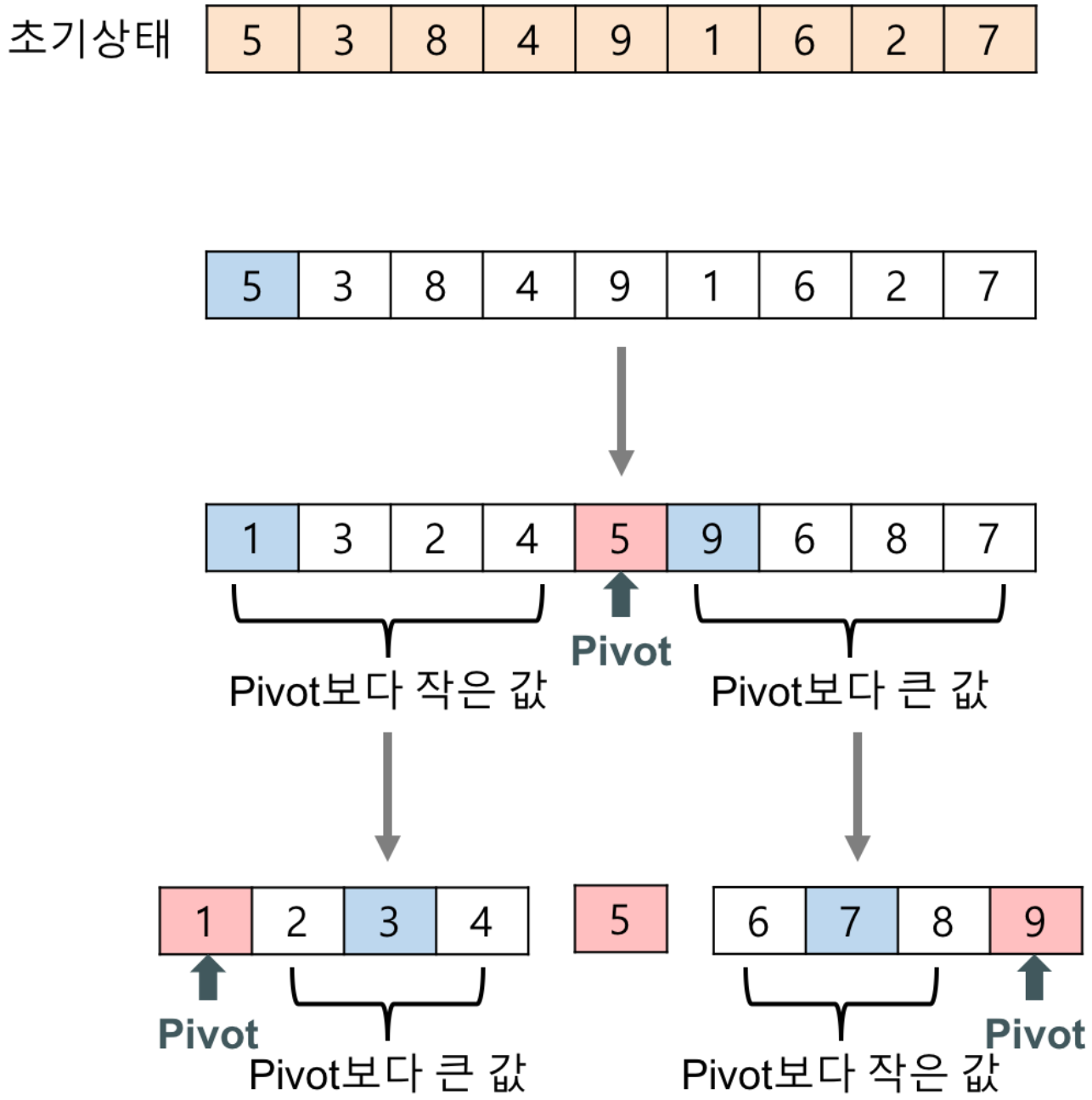


# 퀵 정렬(quick sort)

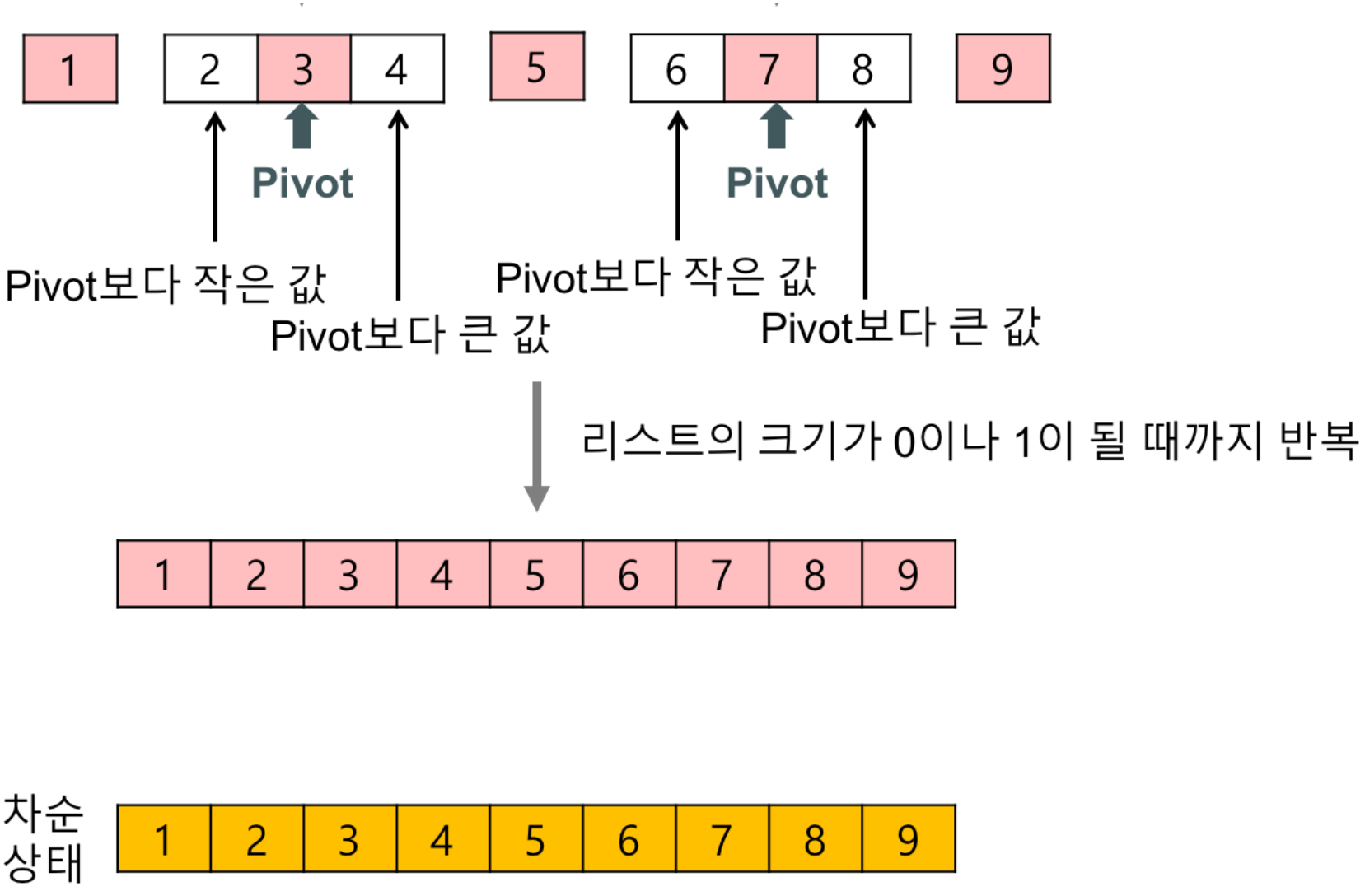
방법

1. 피봇을 고른 후 피봇을 기준으로 피봇의 왼쪽은 피봇보다 작은 값, 피봇의 오른쪽은 피봇보다 큰 값이 되도록 만든다.
2. 엇갈린 기점으로 배열을 나누어 앞 과정을 반복한다.

# 퀵 정렬


초기상태
5 3 8 4 9 1 6 2 7
5 3 8 4 9 1 6 2 7
1 3 2 4 5 9 6 8 7
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
1 2 3 4
5
6 7 8 9
Pivot
Pivot보다 큰 값
Pivot보다 작은 값
Pivot

# 퀵 정렬

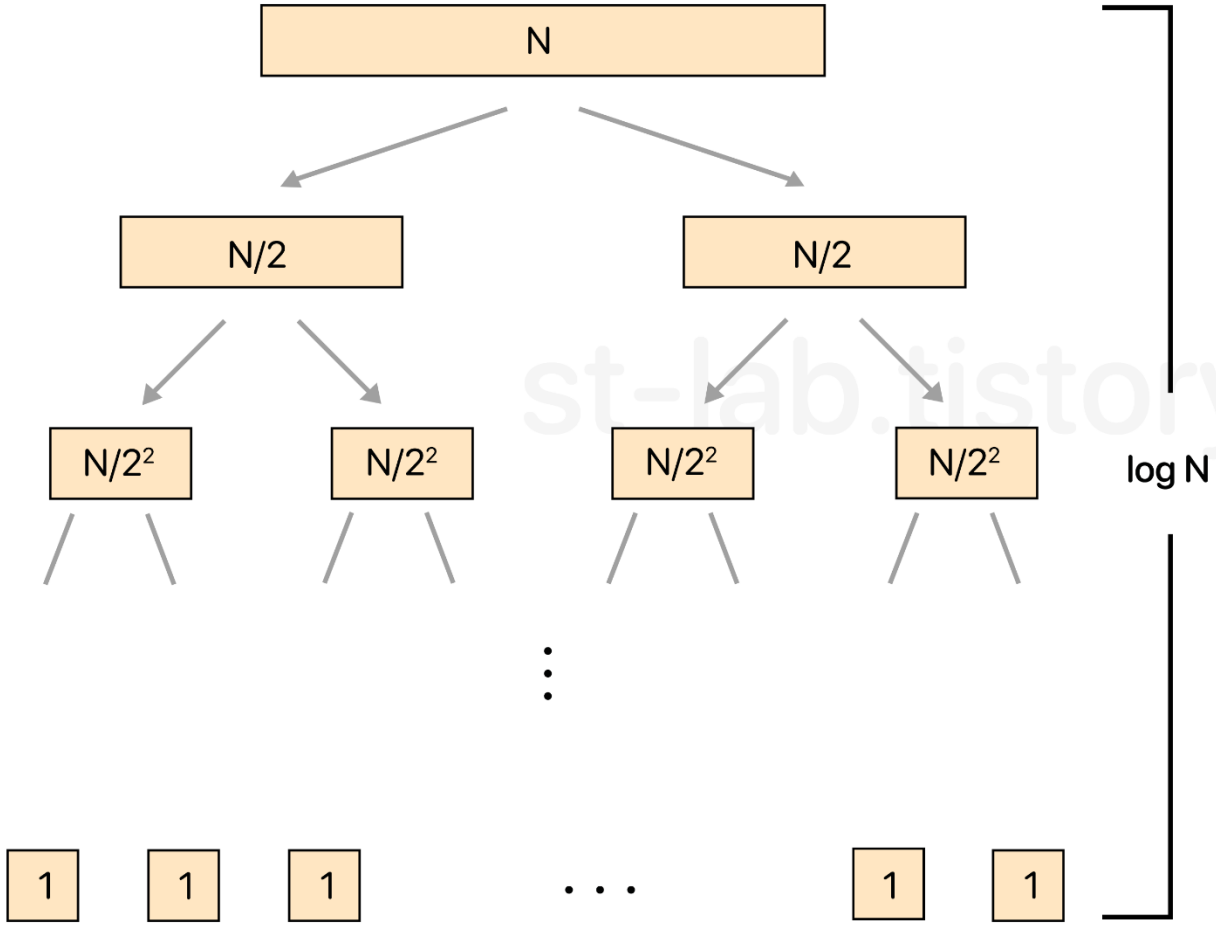

1
2
3
4
5
6
7
8
9
Pivot
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
Pivot보다 작은 값
Pivot보다 큰 값
리스트의 크기가 0이나 1이 될 때까지 반복
1 2 3 4 5 6 7 8 9
오름차순
완성상태
1 2 3 4 5 6 7 8 9

# 퀵 정렬 코드

```java
public static int divide(int[] arr, int start, int end){
    int pivot = arr[(start+end)/2];
    while(start<=end){
        while(pivot>arr[start]) start++;
        while(pivot<arr[end]) end--;
        if(start<=end){
            int tmp = arr[start];
            arr[start] = arr[end];
            arr[end] = tmp;
            start++;
            end--;
        }
    }
    return start;
}

public static void quick(int[] arr, int start, int end){
    int point = divide(arr, start, end);
    if(start<point-1) quick(arr, start,  end: point-1);
    if(end>point) quick(arr, point, end);
}
```

# 퀵 정렬 시간 복잡도




노드의 개수 : 1
노드의 크기 : N

노드의 개수 : 2
노드의 크기 : $N/2^1 = N/2$

비교횟수 = N

노드의 개수 : $2^2 = 4$
노드의 크기 : $N/2^2 = N/4$

비교횟수 = N

노드의 개수 : $2^i$
노드의 크기 : $N/2^i$

비교횟수 = $2^i * (N / 2^i)$
= N

노드의 개수 : N
노드의 크기 : 1

비교횟수 = N

# 퀵 정렬 시간 복잡도

## 노드의 개수

$$N = \sum_{k=1}^{d} a_k = 1 + 2 + 4 + \cdots + 2^{d-1}$$

$$= 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{d-1}$$

$$= \frac{1(2^d - 1)}{2 - 1}$$

$$= 2^d - 1$$

## 트리의 높이

$$N = 2^d - 1$$

$$N + 1 = 2^d$$

$$\log_2(N + 1) = \log_2 2^d$$

$$\log_2(N + 1) = d$$

$$h = d - 1 = \log_2(N + 1) - 1$$

$$h = \log_2(N)$$

# 퀵 정렬 시간 복잡도

한 레벨에서 비교 작업에 대한 시간 복잡도

$$O\left(2^n * \frac{n}{2^n}\right) = O(n)$$

트리의 높이
$O(\log n)$

따라서 시간 복잡도는 $O(n) * O(\log n)$