PEPSI
Protect Every Personal Secret Information

KOOKMIN UNIVERSITY SINCE 1946
KMU
국민대학교

정렬 알고리즘

2022년 02월 05일

PEPSI

발표자: 최건희

목차

목차

1. 정렬 알고리즘

◆ 배열에 들어 있는 수들을 오름차순 또는 내림차순으로 정렬시키는 것!

- 본 내용에서는 오름차순으로 정렬하는 것을 기준으로 삼는다.
  - ✓ 오름차순: 작은 것부터 큰 것으로 가는 순서

- 정렬 알고리즘의 종류는 다음과 같다.
  - ✓ 선택 정렬 (Selection Sort)
  - ✓ 버블 정렬 (bubble Sort)
  - ✓ 삽입 정렬 (Insertion Sort)
  - ✓ Quick 정렬 (Quick Sort)
  - ✓ Merge 정렬 (Merge Sort)

목차

2. 선택 정렬 (Selection Sort)

- 맨 앞자리 부터 작은 수를 채워 넣는 것!
  - 모든 수를 다 탐색해야 하는 비효율적 알고리즘
    - ✓ Index로 사용될 변수 (i , j)

A[i] =A[j]

A =

1	4	2	5

2. 선택 정렬 (Selection Sort)

- 맨 앞자리 부터 작은 수를 채워 넣는 것!
  - 모든 수를 다 탐색해야 하는 비효율적 알고리즘
    - ✓ Index로 사용될 변수 (i , j)

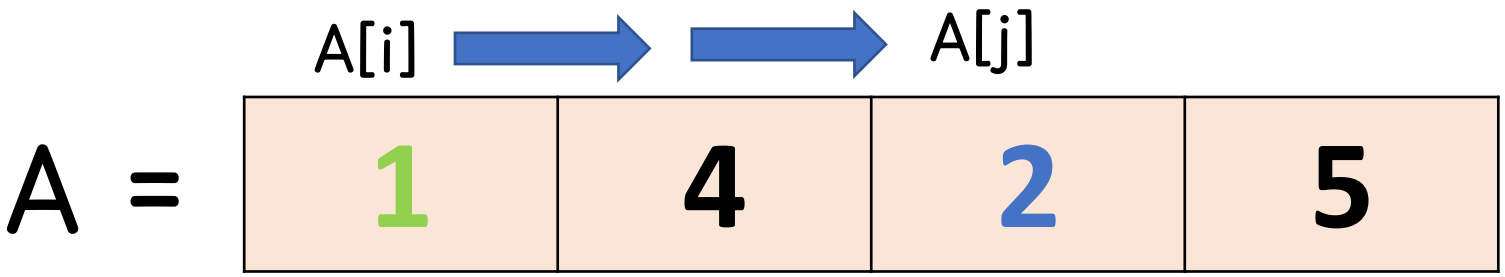
2. 선택 정렬 (Selection Sort)

- 맨 앞자리 부터 작은 수를 채워 넣는 것!
  - 모든 수를 다 탐색해야 하는 비효율적 알고리즘
    - ✓ Index로 사용될 변수 (i , j)

- ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

2. 선택 정렬 (Selection Sort)

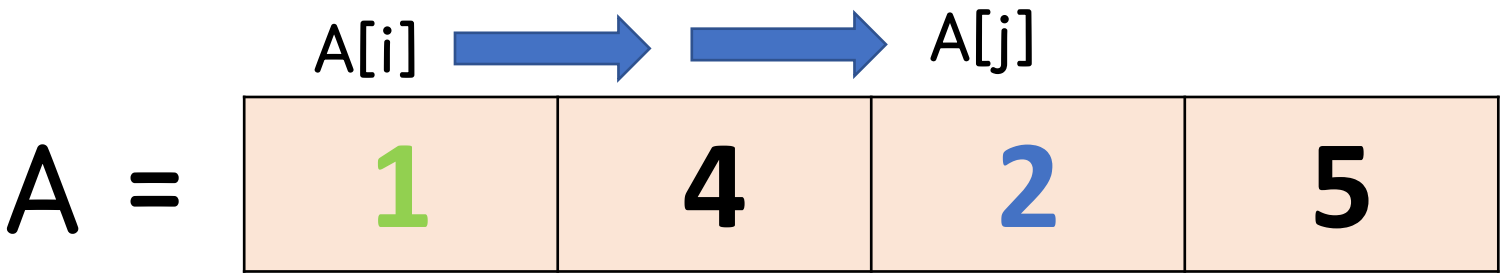
- 맨 앞자리 부터 작은 수를 채워 넣는 것!
  - 모든 수를 다 탐색해야 하는 비효율적 알고리즘
    - ✓ Index로 사용될 변수 (i , j)



- ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

2. 선택 정렬 (Selection Sort)

- 맨 앞자리 부터 작은 수를 채워 넣는 것!
  - 모든 수를 다 탐색해야 하는 비효율적 알고리즘
    - ✓ Index로 사용될 변수 (i , j)



- ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

2. 선택 정렬 (Selection Sort)

◆ CODE

- 2가지의 code. 어떤 코드가 이득일까?

```
void Selection_Sort_v1(int a[], int size)
{
    int i=0,j=0;  //배열의 인덱스

    for(i=0;i<size-1;i++)
    {
        for(int j=i+1;j<size;j++)
        {
            if(a[j]<a[i])
                swap(&a[i],&a[j]);
        }
    }
}
```

```
void Selection_Sort_v2(int a[], int size)
{
    int pivot_index = 0;
    int i=0,j=0;  //배열의 인덱스

    for(i=0;i<size-1;i++)
    {
        pivot_index = i;
        for(int j=i+1;j<size;j++)
        {
            if(a[j]<a[i])
                pivot_index = j;
        }
        swap(&a[i],&a[pivot_index]);
    }
}
```

✓ Swap함수를 살펴보자.

2. 선택 정렬 (Selection Sort)

◆ CODE

● Swap 함수

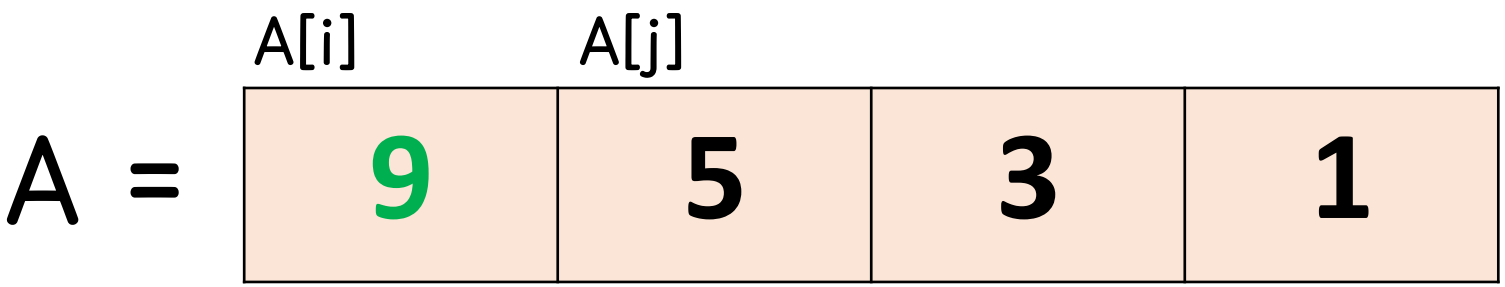
```
void swap(int a[],int b[])
{
    int tmp = 0; //잠시 값을 지니고 있을 변수

    tmp = a[0];

    a[0] = b[0];
    b[0] = tmp;
}
```

목차

3. 버블 정렬 (Bubble Sort)

◆ 큰 수를 맨 뒤로 보내는 것

- 두개 씩 비교하며 큰 수를 뒤에 위치시킨다.
  - ✓ Index로 사용될 변수 (i , j)



  - ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

3. 버블 정렬 (Bubble Sort)

◆ 큰 수를 맨 뒤로 보내는 것

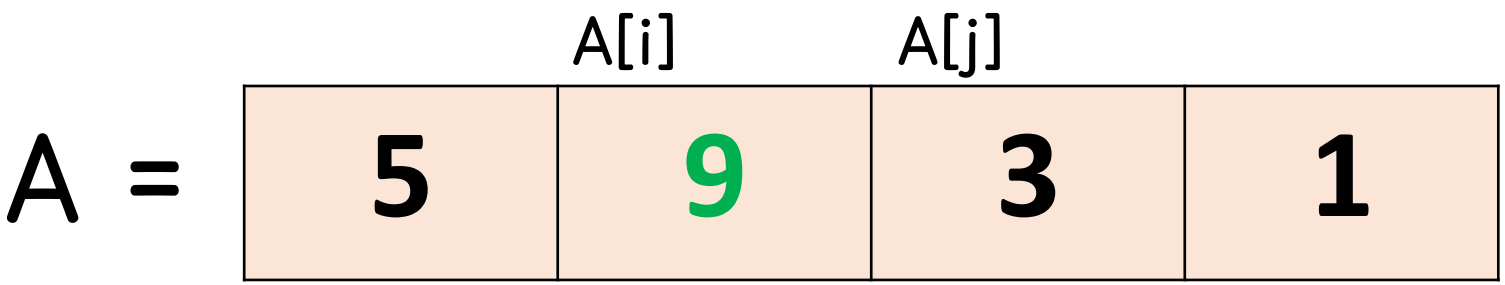
- 두개 씩 비교하며 큰 수를 뒤에 위치시킨다.
  - ✓ Index로 사용될 변수 (i , j)

  - ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

3. 버블 정렬 (Bubble Sort)

◆ 큰 수를 맨 뒤로 보내는 것

- 두개 씩 비교하며 큰 수를 뒤에 위치시킨다.
  - ✓ Index로 사용될 변수 (i , j)

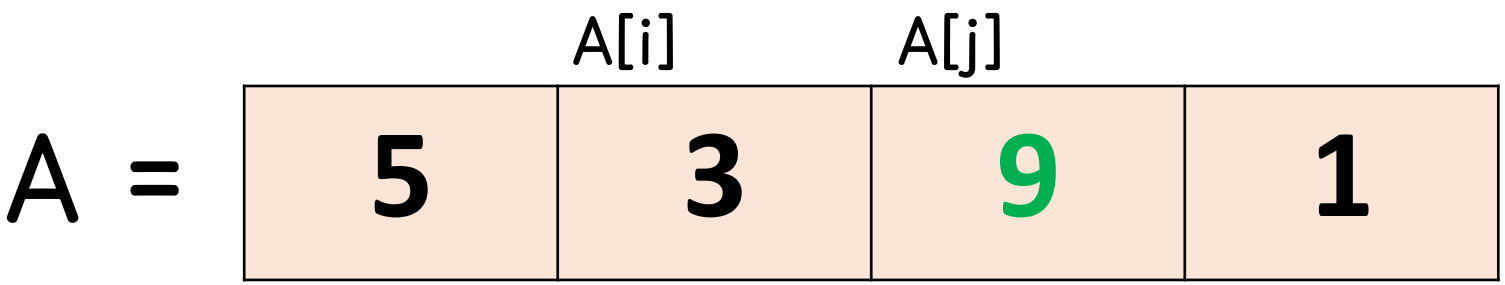


  - ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

3. 버블 정렬 (Bubble Sort)

◆ 큰 수를 맨 뒤로 보내는 것

- 두개 씩 비교하며 큰 수를 뒤에 위치시킨다.
  - ✓ Index로 사용될 변수 (i , j)



  - ✓ IF (A[i] 〉 A[j]) , then Swap(A[i].A[j])

3. 버블 정렬 (Bubble Sort)

◆ CODE

```
void bubbleSort(int a[], int size)
{
    int i=0,j=0;
    for(i=size-1;i>0;i--)
    {
        for(j=0;j<i;j++)
        {       A[i]   A[j]  //우리 예시에서
            if(a[j]>a[j+1])
                swap(&a[j],&a[j+1]);
        }
    }
}
```

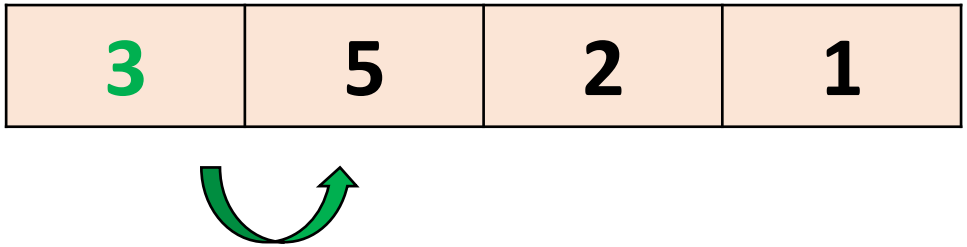
목차

4. Quick 정렬 (Quick Sort)

◆ 맨 앞의 수를 기준으로 배열을 2개로 분할 시키는 것

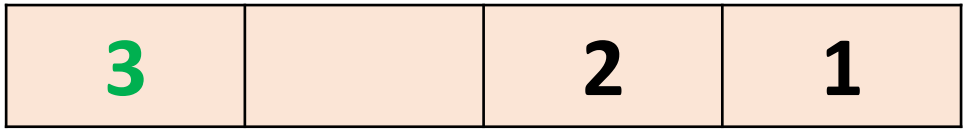
- 맨 앞의 수를 기준으로 둔다. (이를 pivot:피벗이라 한다.)
- 2번째부터 끝까지 pivot과 비교하며 작은 수는 왼쪽 배열에 큰 수는 오른쪽 배열에 넣어준다.
- 피벗은 두 배열 사이에 들어간다.

✓ 감을 잡기 위하여..

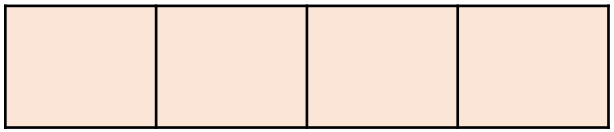


4. Quick 정렬 (Quick Sort)

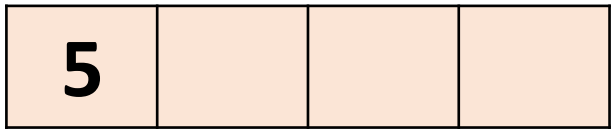
- 맨 앞의 수를 기준으로 배열을 2개로 분할 시키는 것
  - 맨 앞의 수를 기준으로 둔다. (이를 pivot:피벗이라 한다.)
  - 2번째부터 끝까지 pivot과 비교하며 작은 수는 왼쪽 배열에 큰 수는 오른쪽 배열에 넣어준다.
  - 피벗은 두 배열 사이에 들어간다.



Small

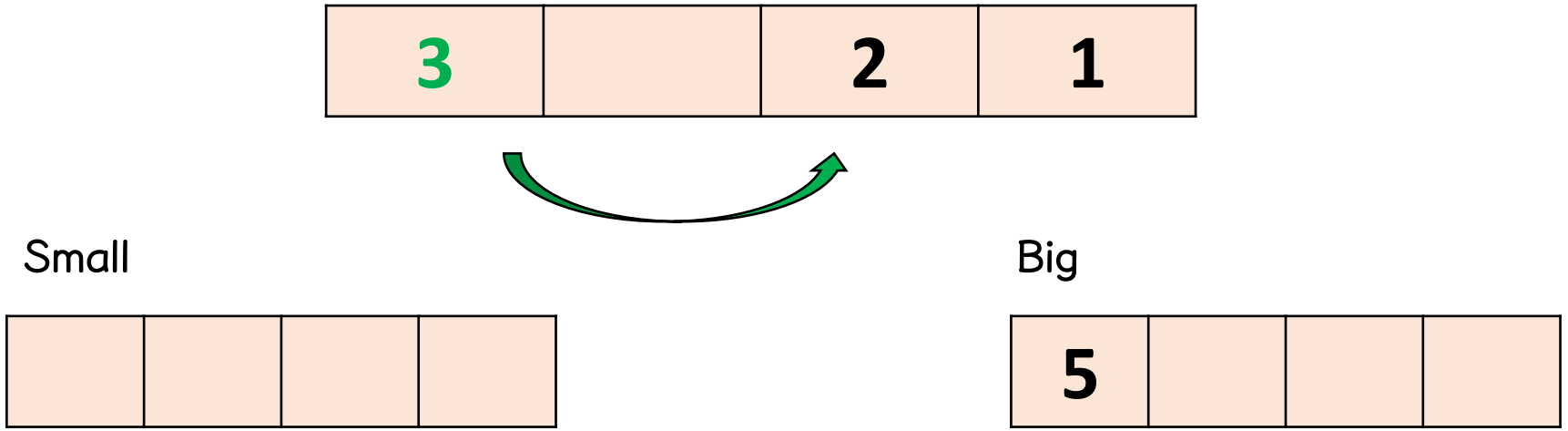


Big



4. Quick 정렬 (Quick Sort)

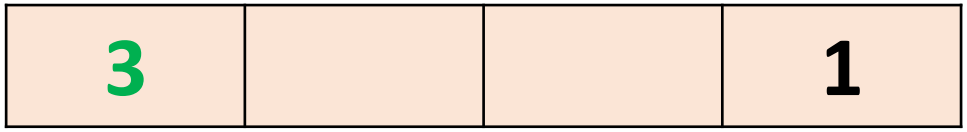
- 맨 앞의 수를 기준으로 배열을 2개로 분할 시키는 것
  - 맨 앞의 수를 기준으로 둔다. (이를 pivot:피벗이라 한다.)
  - 2번째부터 끝까지 pivot과 비교하며 작은 수는 왼쪽 배열에 큰 수는 오른쪽 배열에 넣어준다.
  - 피벗은 두 배열 사이에 들어간다.



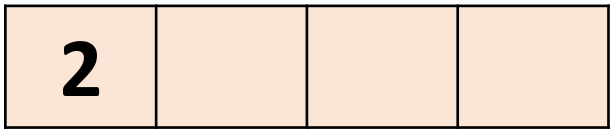
4. Quick 정렬 (Quick Sort)

◆ 맨 앞의 수를 기준으로 배열을 2개로 분할 시키는 것

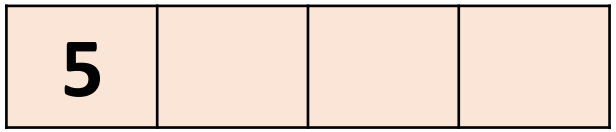
- 맨 앞의 수를 기준으로 둔다. (이를 pivot:피벗이라 한다.)
- 2번째부터 끝까지 pivot과 비교하며 작은 수는 왼쪽 배열에 큰 수는 오른쪽 배열에 넣어준다.
- 피벗은 두 배열 사이에 들어간다.



Small



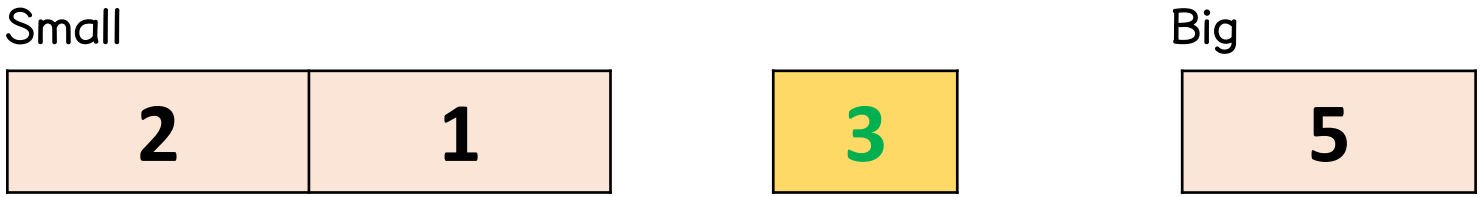
Big



4. Quick 정렬 (Quick Sort)

◆ 맨 앞의 수를 기준으로 배열을 2개로 분할 시키는 것

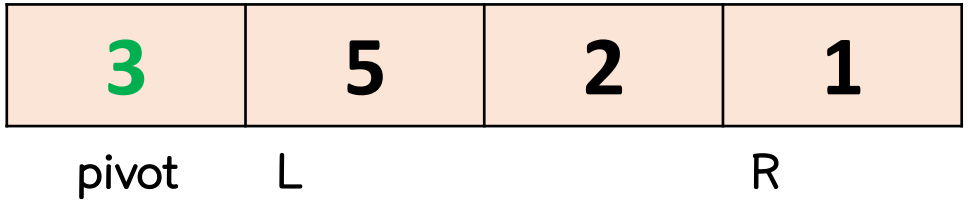
- 맨 앞의 수를 기준으로 둔다. (이를 pivot:피벗이라 한다.)
- 2번째부터 끝까지 pivot과 비교하며 작은 수는 왼쪽 배열에 큰 수는 오른쪽 배열에 넣어준다.
- 피벗은 두 배열 사이에 들어간다.
- 더 이상 정렬할 수 없을 때까지 반복한다.



4. Quick 정렬 (Quick Sort)

◆ 실제 로직

- 1. 피벗 설정 및 L과 R설정.



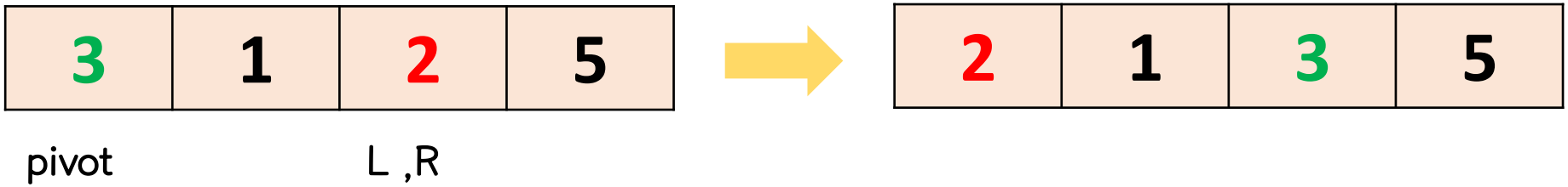
- 2. L은 ⟶ 로 진행 and R은 ⟵ 로 진행하며

  pivot보다 작은 수는 왼쪽에 pivot보다 큰 수는 오른쪽으로 정렬한다.

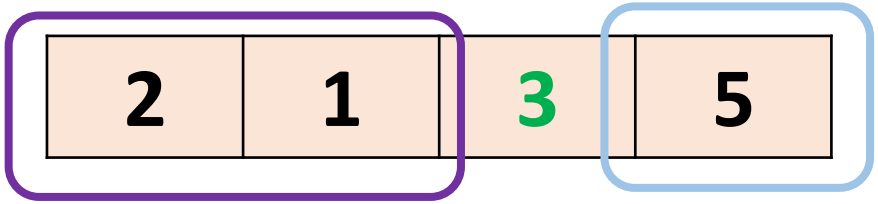
4. Quick 정렬 (Quick Sort)

◆ 실제 로직

- 3. L과 R이 만나면 그 수와 pivot을 비교 후 정렬한다.



- 4. pivot을 기준으로 왼쪽 배열과 오른쪽 배열을 동일한 과정을 반복 수행한다.



4. Quick 정렬 (Quick Sort)

◆ CODE

```
void QuickSort(int arr[], int left, int right)
{
    if (left <= right)
    {
        int pivot = Partition(arr, left, right); // 둘로 나누어서
        QuickSort(arr, left, pivot - 1); // 왼쪽 영역을 정렬한다.
        QuickSort(arr, pivot + 1, right); // 오른쪽 영역을 정렬한다.
    }
}
```

4. Quick 정렬 (Quick Sort)

◆ CODE

```
int Partition(int arr[], int left, int right)
{
    int pivot = arr[left]; // 피벗의 위치는 가장 왼쪽에서 시작
    int low = left + 1;
    int high = right;

    while (low <= high) // 교차되기 전까지 반복한다
    {
        while (low <= right && pivot >= arr[low]) // 피벗보다 큰 값을 찾는 과정
        {
            low++; // low를 오른쪽으로 이동
        }
        while (high >= (left+1)  && pivot <= arr[high]) // 피벗보다 작은 값을 찾는 과정
        {
            high--; // high를 왼쪽으로 이동
        }
        if (low <= high)// 교차되지 않은 상태이면 스왑 과정 실행
        {
            Swap_quick(arr, low, high); //low와 high를 스왑
        }
    }
    Swap_quick(arr, left, high); // 피벗과 high가 가리키는 대상을 교환
    return high;  // 옮겨진 피벗의 위치정보를 반환

}
```

4. Quick 정렬 (Quick Sort)

◆ CODE

```
void QuickSort_recursive(int arr[], int left, int right) {
  int L = left, R = right;
  int temp;
  int pivot = arr[(left + right) / 2]; //피봇 위치(중앙)의 값을 받음.

  //아래의 while문을 통하여 pivot 기준으로 좌, 우 크고 작은 값 나열. = Partition
  while (L <= R)
  {
  //pivot이 중간 값이고, 비교 대상 arr[L], arr[R]은 pivot과 비교하니 중간 지점을 넘어가면 종료로 볼 수 있음.
    while (arr[L] < pivot) //left부터 증가하며 pivot 이상의 값을 찾음.
      L++;
    while (arr[R] > pivot) //right부터 감소하며 pivot 이하 값을 찾음.
      R--;
    //L, R 모두 최대 pivot 위치까지 이동.

    if (L <= R) { //현재 L이 R이하면. (이유 : L>R 부분은 이미 정리가 된 상태임).
      if (L != R) { //같지 않은 경우만.
        Swap_quick(arr,L,R);
      } //L과 R이 같다면 교환(SWAP)은 필요 없고 한 칸씩 진행만 해주면 됨.
      L++; R--; //그리고 L,R 한 칸 더 진행.
      }

    }

  if (left < R)
    QuickSort_recursive(arr, left, R);
  if (L < right)
    QuickSort_recursive(arr, L, right);
}
```

4. Quick 정렬 (Quick Sort)

◆ CODE

```
void Swap_quick(int arr[], int a, int b) // a,b 스왑 함수
{
    int temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}
```

목차

5. Merge 정렬 (Merge Sort)

◆ 하나의 배열을 두개의 균등한 크기로 분할하고, 분할된 리스트를 정렬해서 다시 합하는 방식!

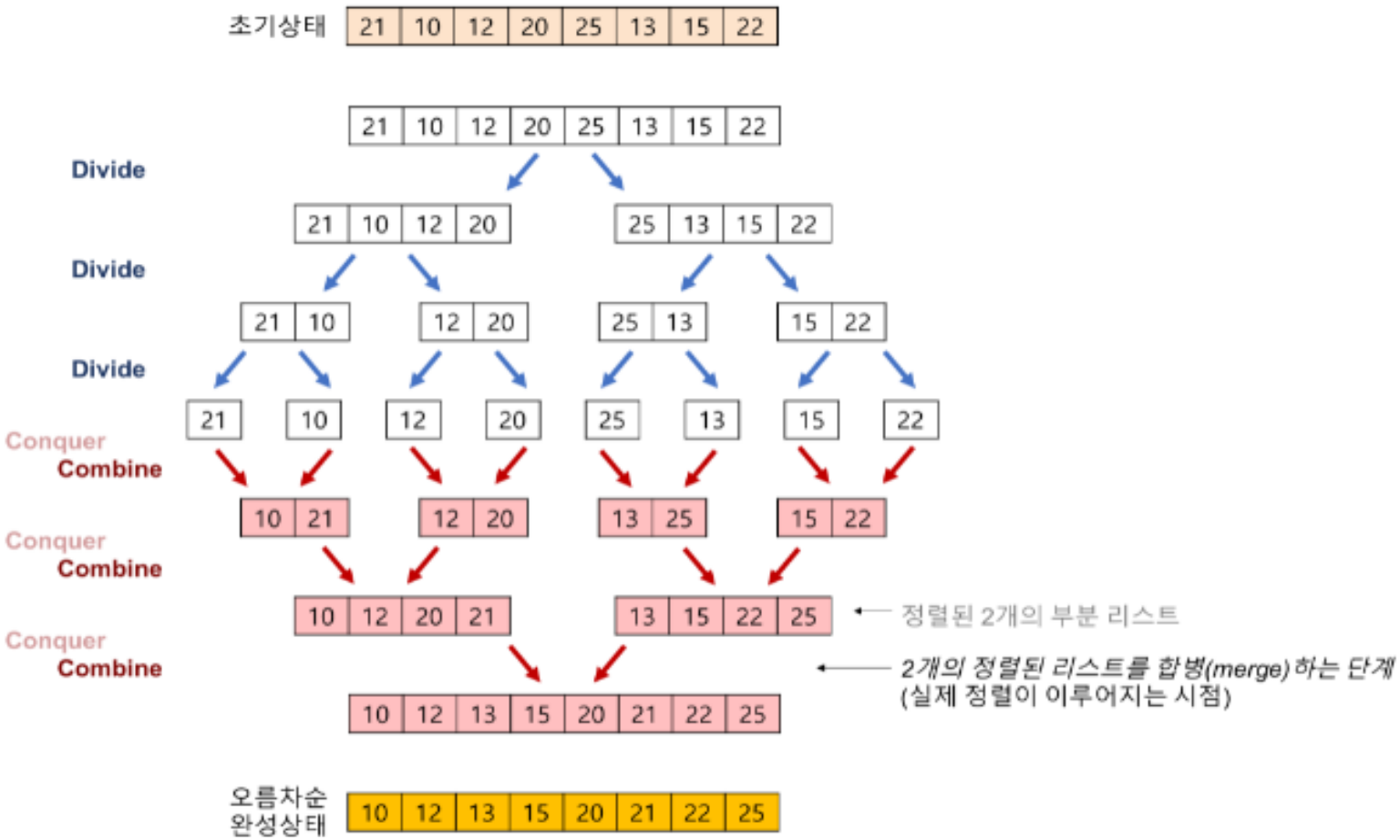
✓ 분할 (Divide) -〉 정복(Conquer) -〉 결합(Combine) 순으로 이해

✓ 분할: 입력 받은 배열을 같은 크기의 2개의 부분 배열로 분할하는 과정.

✓ 정복: 부분 배열을 정렬하는 과정. 이때 배열의 크기가 충분히 작지 않으면 분할을 계속 진행해 준다.

✓ 결합: 정렬된 부분 배열들을 하나의 배열로 합병하는 과정.

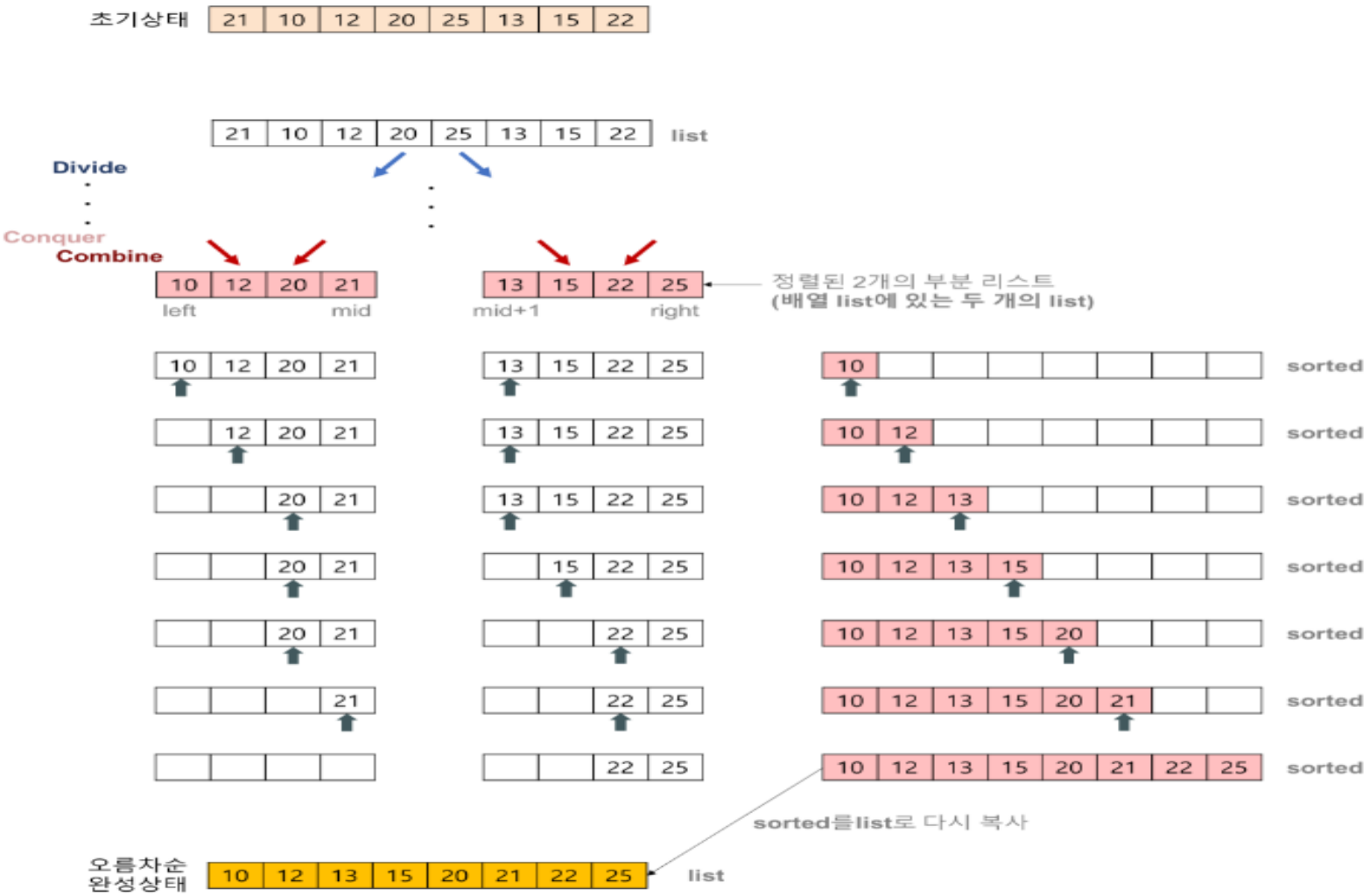
5. Merge 정렬 (Merge Sort)



합병 정렬(merge sort) 이란 - https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

5. Merge 정렬 (Merge Sort)

◆ 예시)



합병 정렬(merge sort) 이란 - https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

5. Merge 정렬 (Merge Sort)

◆ CODE

```
void MergeSort(int arr[], int start, int end)
{
    int middle = 0;
    if(start < end)
    {
        middle = (start + end) / 2;
        MergeSort(arr, start, middle);
        MergeSort(arr, middle+1, end);
        Merge(arr, start, end, middle);
    }
}

int main()
{

    int a[MAX_SIZE] = {1,3,4,10};
    MergeSort(a,0,3);
    for(int i=0;i<4;i++)
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```

합병 정렬(merge sort) 이란 - https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

5. Merge 정렬 (Merge Sort)

◆ CODE

```
# define MAX_SIZE 4
int tmp[MAX_SIZE];

void Merge(int arr[], int start, int end, int middle)
{
    int i=0, j=0, k=0;

    i = start;
    j = middle + 1;
    k = start;



    while(i <= middle && j <= end)
    {
        if(arr[i] <= arr[j]) tmp[k++] = arr[i++];

        else tmp[k++] = arr[j++];
    }

   if(i>middle)
   {
       for(int l = j; l <= end;l++)
       {
           tmp[k++] = arr[l];
       }
   }

   else
   {
       for(int l=i; l <= middle; l++)
        {
            tmp[k++] = arr[l];
        }
   }

   for(int l = start; l <= end; l++)
   {
       arr[l] = tmp[l];
   }
}
```

합병 정렬(merge sort) 이란 - https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

Question?