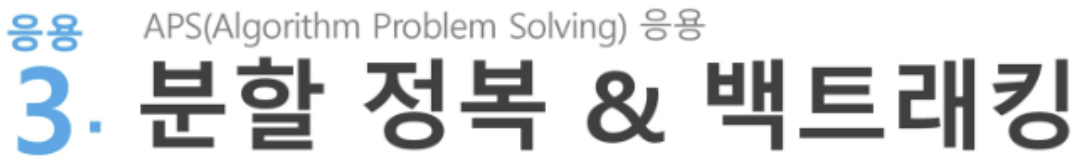
머지소트 발표 - 복사본

◆ 머지 소트란 무엇인가?

- 싸피 알고리즘 수업의 분할정복 & 백트래킹 배울 때 배웠던 개념



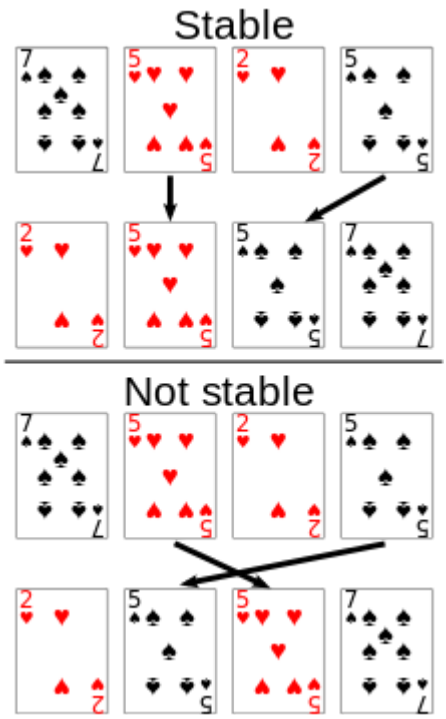
- 분할정복
- 퀵 정렬
- 이진 검색
- 백트래킹
- 트리

- 문제를 분할해서 해결하는 분할 정복 기법 중 대표적인 알고리즘으로 퀵 정렬과 병합 정렬이 있다.

◆ 병합 정렬 (Merge Sort)

- **병합정렬, 합병정렬, 머지 소트**
- **존 폰 노이만**씨가 1945년에 개발한 **분할 정복 알고리즘** 중 하나
- **비교 기반 정렬 알고리즘** : 말 그대로 비교를 통해 sorting이 이루어진다.

- **안정 정렬** : 반복 혹은 중복되는 요소가 있을 때(ex. 값) **입력과 동일한 순서로 정렬**이 되는 것



여러 개의 정렬된 자료의 집합을 병합하여 한 개의 정렬된 집합으로 만드는 방식

분할 정복 알고리즘 활용

- 자료를 최소 단위의 문제까지 나눈 후에 차례대로 정렬하여 최종 결과를 얻어냄.
- top-down 방식

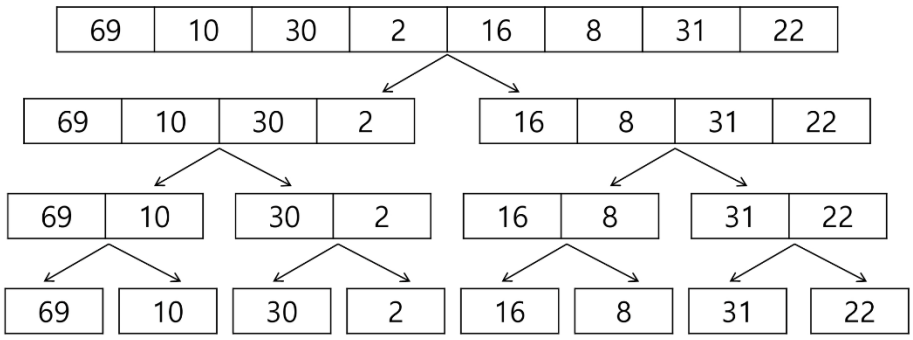
시간 복잡도

- O(n log n)

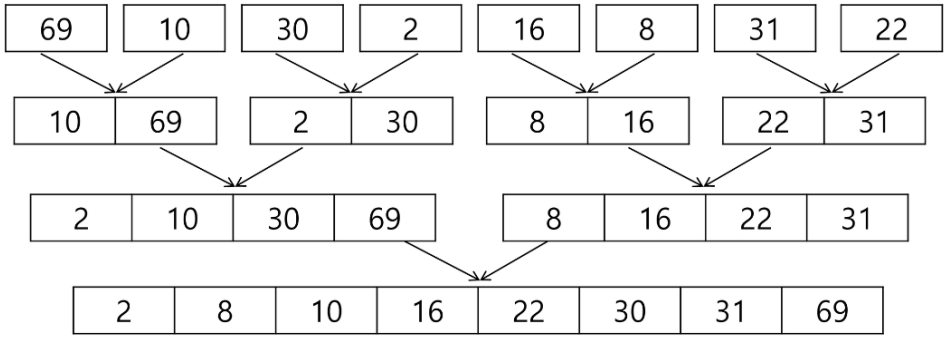
- N개의 원소가 있다면 N개의 원소에 대해 계속 둘로 나눈 후, 최소 단위가 되었을 때 다시 합치면서 정렬해나가는 정렬 방법이다.
- 병합 정렬 과정은 크게 2가지의 과정, 분할 단계와 병합 단계를 거친다.
- (사실상 분할 - 정렬 - 병합 3단계)
- 병합 단계에서 부분 집합들을 크기 순으로 정렬하면서 합쳐준다.

병합 정렬 과정

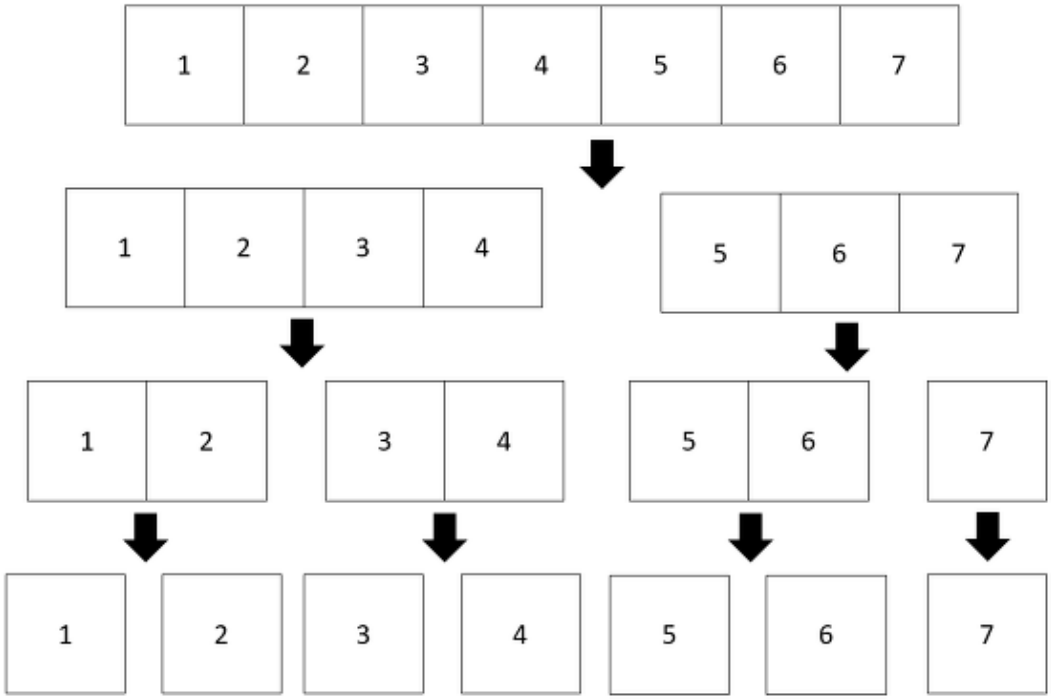
- {69, 10, 30, 2, 16, 8, 31, 22}를 병합 정렬하는 과정
- 분할 단계 : 전체 자료 집합에 대하여, 최소 크기의 부분집합이 될 때까지 분할 작업을 계속한다.



- 병합 단계 : 2개의 부분집합을 정렬하면서 하나의 집합으로 병합
- 8개의 부분집합이 1개로 병합될 때까지 반복함



- 물론 총 개수가 홀수인 경우에도 가능


1 2 3 4 5 6 7
1 2 3 4
5 6 7
1 2
3 4
5 6
7
1
2
3
4
5
6
7


4 1 7 5 3 2 6

알고리즘 : 분할 과정

```
merge_sort(LIST m)
    IF length(m) == 1 : RETURN m

    LIST left, right
    middle ← length(m) / 2
    FOR x in m before middle
         add x to left
    FOR x in m after or equal middle
         add x to right

    left ← merge_sort(left)
    right ← merge_sort(right)

    RETURN merge(left, right)
```

- 분할하는 과정은 재귀함수를 통해 분할을 한다.
- size가 n (위에서는 length(m))인 배열이 있으면, [1 ~ n/2]의 배열과, [n/2 + 1 ~ n]으로 나누는 과정을 재귀를 통해 size가 1이 될 때까지 반복

알고리즘 : 병합 과정

```
merge(LIST left, LIST right)
    LIST result

    WHILE length(left) > 0 OR length(right) > 0
        IF length(left) > 0 AND length(right) > 0
            IF first(left) <= first(right)
                append popfirst(left) to result
            ELSE
                append popfirst(right) to result
        ELIF length(left) > 0
            append posfirst(left) to result
        ELIF length(right) > 0
            append popfirst(right) to result
    RETURN result
```

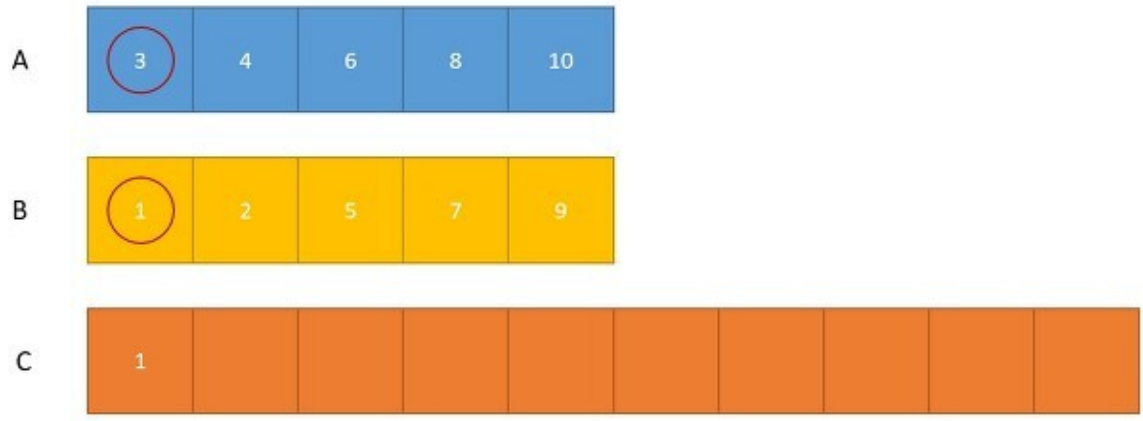
- 이때 if문을 통해 왼쪽보다 오른쪽이 더 크거나 같은지 확인한다.
- if left ≤ right
- 왼쪽 오른쪽 부분의 원소를 크기 순으로 구분하고 병합해주도록 한다.

- 이 때 한쪽의 원소가 모두 정렬이 된다면 남은 한쪽의 원소를 전부 넣어준다.
- 이러한 과정이 재귀를 통해 이루어지고, 모두 수행하고 나면 정렬이 완료된다.

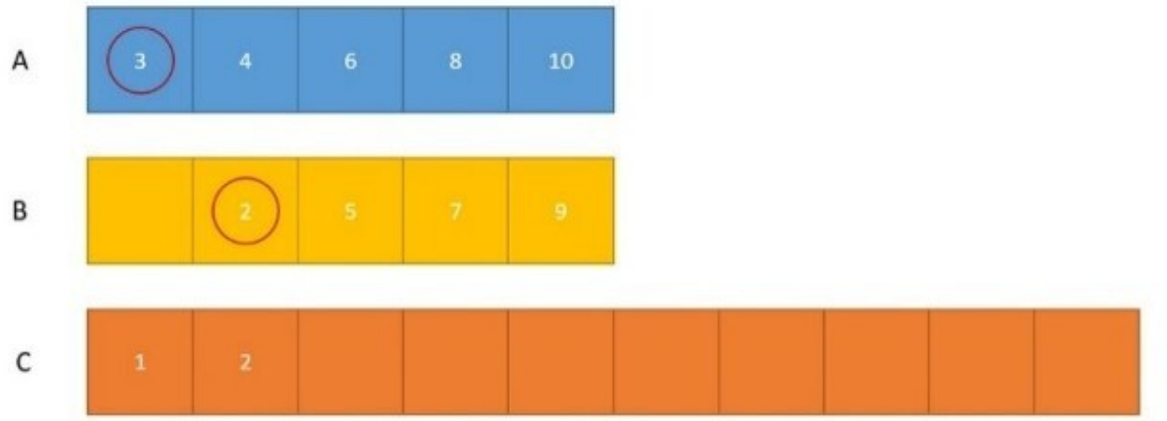
◆ 병합정렬 (merge sort)의 동작 방식

- 각 단계에서 두 개의 배열을 정렬을 하며 결합을 한다.
- 두 개의 배열을 결합할 때는 각각의 배열은 전 단계에서 이미 정렬이 된 상태이기 때문에 맨 앞의 원소들만 비교를 하여 병합을 한다.
- 각각의 배열을 결합을 하며 정렬을 하는 과정은 아래와 같다.

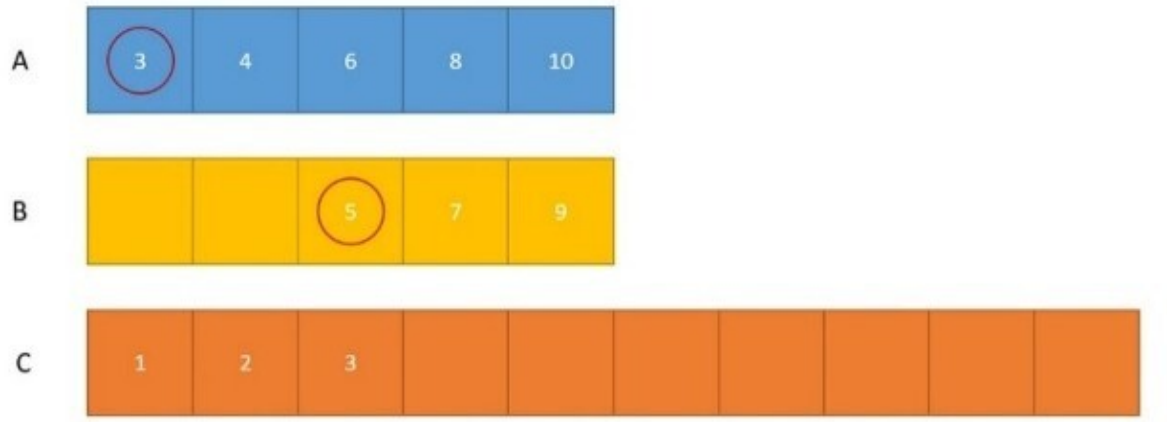
1. A의 첫 번째 원소인 3과, B의 첫 번째 원소인 1을 비교한다. 1이 더 작기 때문에 C배열에 1을 넣는다.



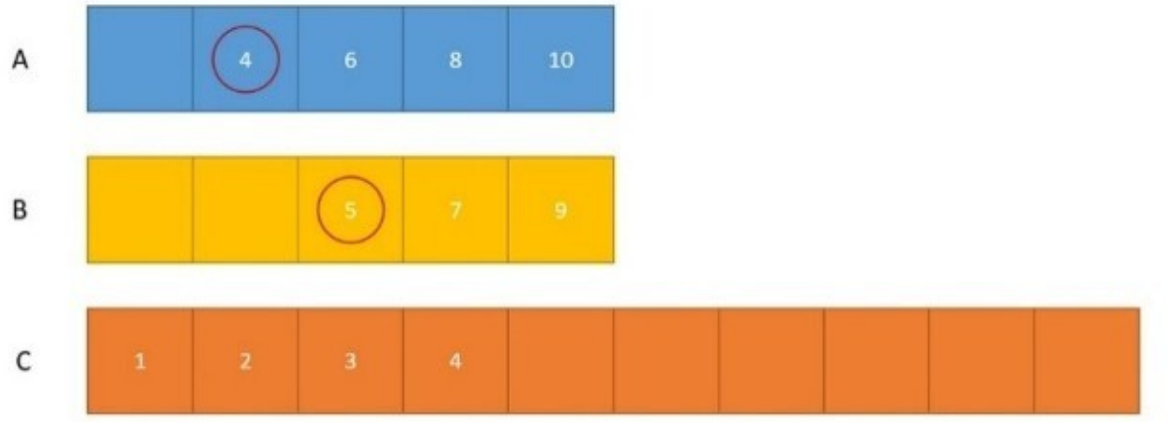
2. A의 첫 번째 원소인 3과, B의 두 번째 원소인 2을 비교한다. 2이 더 작기 때문에 C배열에 2을 넣는다.



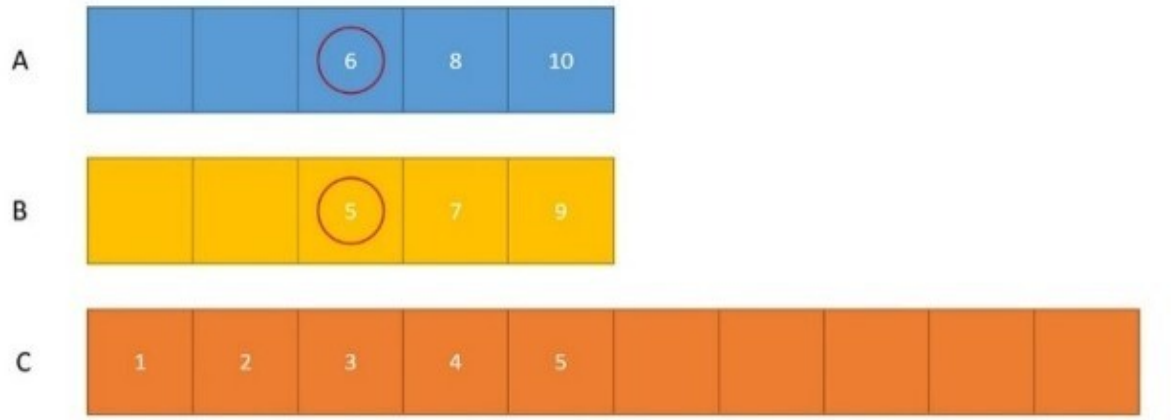
3. A의 첫 번째 원소인 3과, B의 세 번째 원소인 5을 비교한다. 3이 더 작기 때문에 C배열에 3을 넣는다.



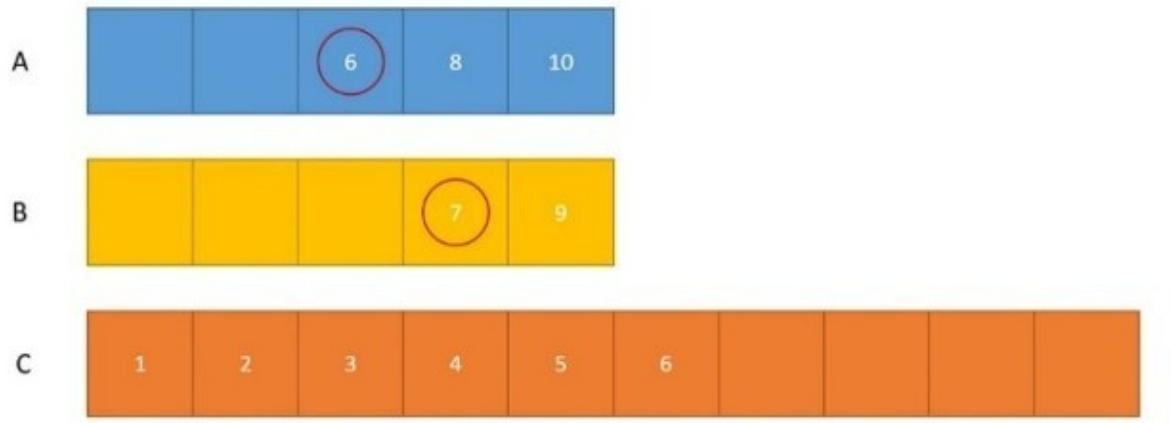
4. A의 두 번째 원소인 4과, B의 세 번째 원소인 5을 비교한다. 4이 더 작기 때문에 C배열에 4을 넣는다.



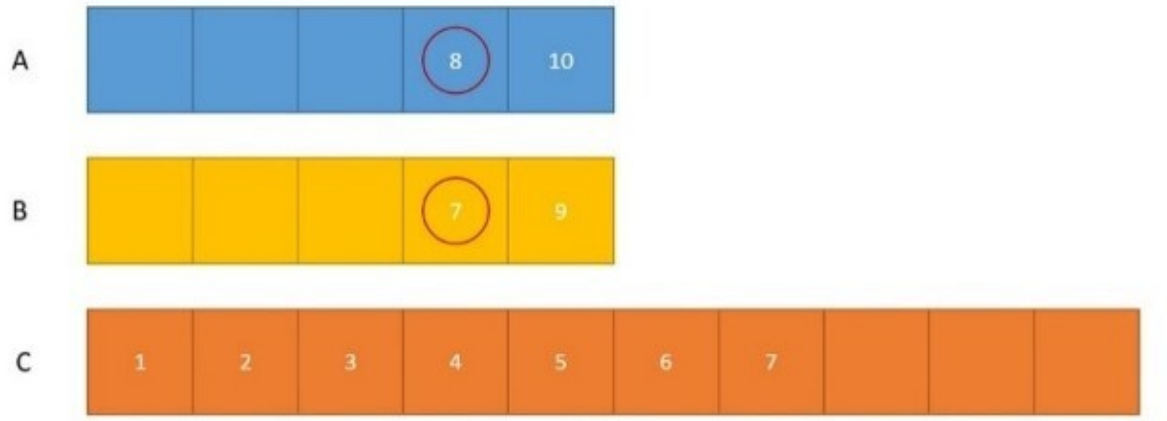
5. A의 세 번째 원소인 6과, B의 세 번째 원소인 5을 비교한다. 5이 더 작기 때문에 C배열에 5을 넣는다



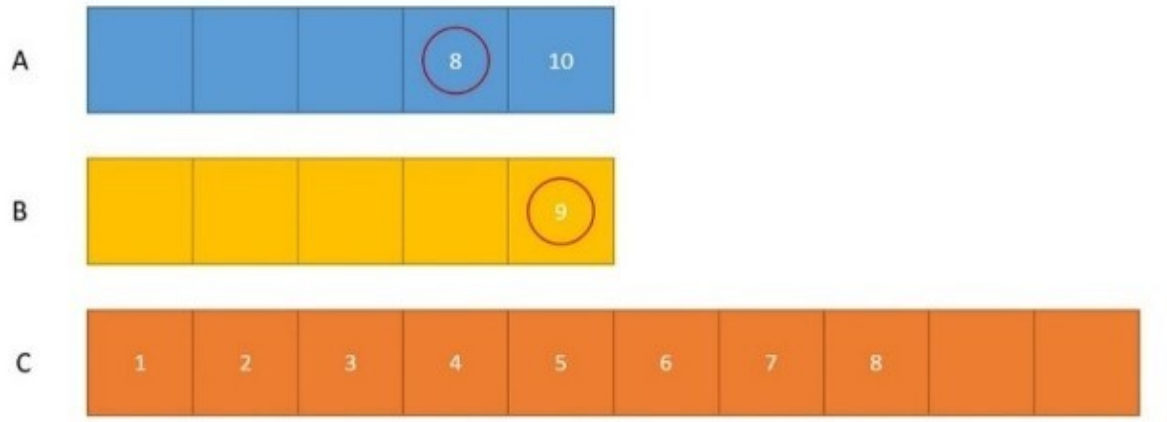
6. A의 세 번째 원소인 6과, B의 네 번째 원소인 7을 비교한다. 6이 더 작기 때문에 C배열에 6을 넣는다.



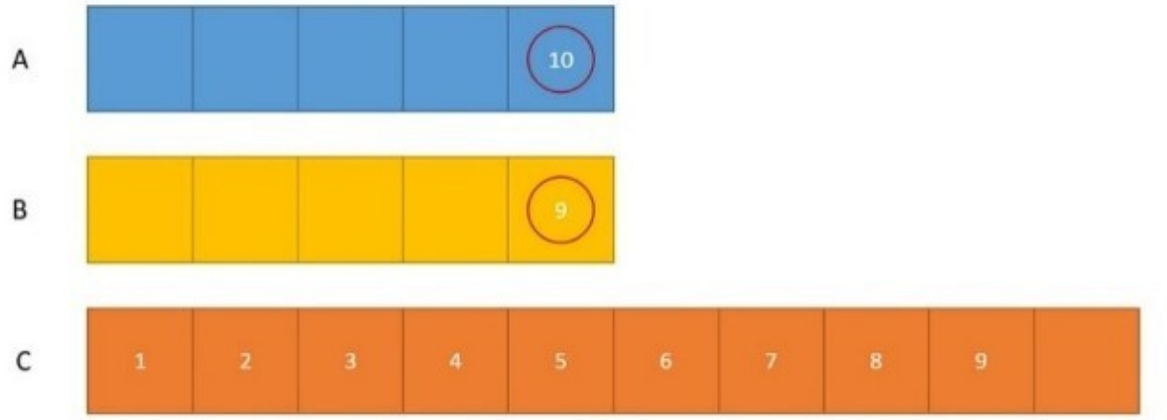
7. A의 네 번째 원소인 8과, B의 네 번째 원소인 7을 비교한다. 7이 더 작기 때문에 C배열에 7을 넣는다.



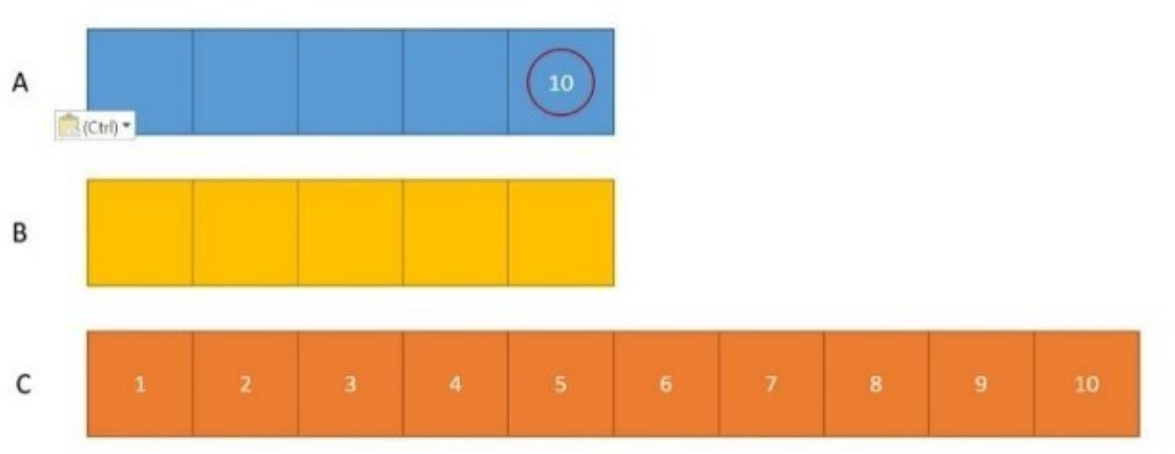
8. A의 네 번째 원소인 8과, B의 다섯 번째 원소인 9을 비교한다. 8이 더 작기 때문에 C배열에 8을 넣는다.



9. A의 다섯 번째 원소인 10과, B의 다섯 번째 원소인 9을 비교한다. 9이 더 작기 때문에 C배열에 9을 넣는다.



10. B 배열의 더 이상 원소가 없기 때문에 A배열의 남은 원소를 C 배열에 넣는다.



◆ Merge Sort의 시간 복잡도

- Merge sort의 time complexity를 계산하기 분석하기 위해 worst-case의 경우를 생각한다.
- Size가 n인 배열을 merge sort를 통하여 정렬을 할 경우, 분할을 한 후, 결합을 하는 과정에서 총 $log_2 n$의 결합 과정을 거친다.
- 또한 각각의 결합 과정은 최대 n번의 비교와 정렬 과정을 거친다.
- 그래서 merge sort의 time complexity는 θ ($nlog_2 n$)이 된다. Merge sort의 경우는 insertion sort와 다르게 어떠한 경우에도 n 의 연산 횟수를 거치게 된다.
- 1초에 약 2천만 번의 연산 횟수
- merge sort이 경우 input의 정렬 정도가 러닝 타임에는 크게 영향을 미치지 않는다.
- 왜냐하면 어쨌든 앞 과정에서 정렬 정도가 낮든 높든 똑같은 연산 횟수를 하기 때문이다.
- 따라서 최악, 평균, 최선 모두 θ ($nlog_2 n$)의 시간 복잡도를 가지게 된다.

◆ 장점 및 단점, 사용 예시

- **장점** : 합병 정렬의 가장 큰 장점은 **안정 정렬 알고리즘**이란 점이다. 입력 배열에 따른 결과가 항상 똑같기 때문에 프로그램의 안정성 면에서 우수하다 할 수 있다.

- **단점** : 정렬과 결합 과정을 보면 알아차리겠지만, Sorting된 값을 넣을 임시 배열이 하나 필요하다. 이런 이유로 레코드가 매우 큰 경우에 메모리 소모도 심하고, 임시 배열로 이동하는 횟수도 많아져 시간적으로도 비효율적이다.
- **해결법** : 위의 단점을 해결하는 방법은 배열이 아닌 **연결 리스트(Linked List)**를 사용하는 것이다. node가 가르키는 구조체의 주소가 바뀌는 과정만 반복되므로 메모리 소모도 없으며 데이터의 이동도 무시할 수준이 된다.

- **연결 리스트 ?**

→ 다른 추상 자료형(Abstract Data Type)을 구현할 때 기반이 되는 기초 선형 자료구조이다.

각 노드가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터를 저장한다.

- 혹시 연결 리스트가 뭔지 더 알아보고 싶으면?

→ https://daimhada.tistory.com/72

- 그러면 우리는 어떨 때 머지소트, 즉 병합정렬을 사용하면 좋을까?

→ 연결 리스트는 순차적인 접근과 같은 합병정렬 방식이 유용

- 여러 정렬 알고리즘 중 하나
- 최소, 최대 값을 구할 때 힙을 쓰는 것처럼 뚜렷한 목적으로 사용하거나 이득을 보는 방식은 아닌듯

◆ 문제풀이 (백준 2751_수 정렬하기2)

https://www.acmicpc.net/problem/2751

```
# 머지소트로 풀어보자
def merge_sort(array):
    # 배열의 크기가 1이라면 그대로 나옴
    if len(array) <= 1:
        return array
    # 그게 아니라면 왼쪽과 오른쪽으로 2개로 나누기 (재귀함수)
    mid = len(array) // 2
    left = merge_sort(array[:mid])
    right = merge_sort(array[mid:])

    # i는 왼쪽 분할 배열 인덱스, j는 오른쪽 분할 배열 인덱스, k는 해당 배열의 인덱스
    i,j,k = 0,0,0
```

```
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            array[k] = left[i]
            i += 1
        else:
            array[k] = right[j]
            j += 1
        k+=1

    # 둘 중 한쪽은 이미 다 채워졌을 때 나머지 넣기
    if i == len(left):
        while j < len(right):
            array[k] = right[j]
            j += 1
            k += 1
    elif j==len(right):
        while i < len(left):
            array[k] = left[i]
            i += 1
            k += 1
    return array

# 데이터 입력
n = int(input())
num = []

for _ in range(n):
    num.append(int(input()))

num = merge_sort(num)

for i in num:
    print(i)
```

→ 사실 이 문제는 n의 범위가 100만이기 때문에 pypy로 풀어야 한다.

- **백준 1151번 문제 단어 정렬의 경우**
- 길이가 짧은 것부터 오도록 하고, 만약 같을 경우 단어를 비교하는데
- 문자열의 경우 비교 연산자를 사용할 때, 아스키 코드를 기준으로 하므로, 사전 순으로 정렬하려면 작은 순으로 정렬해야 한다. (‘a’ < ‘z’)

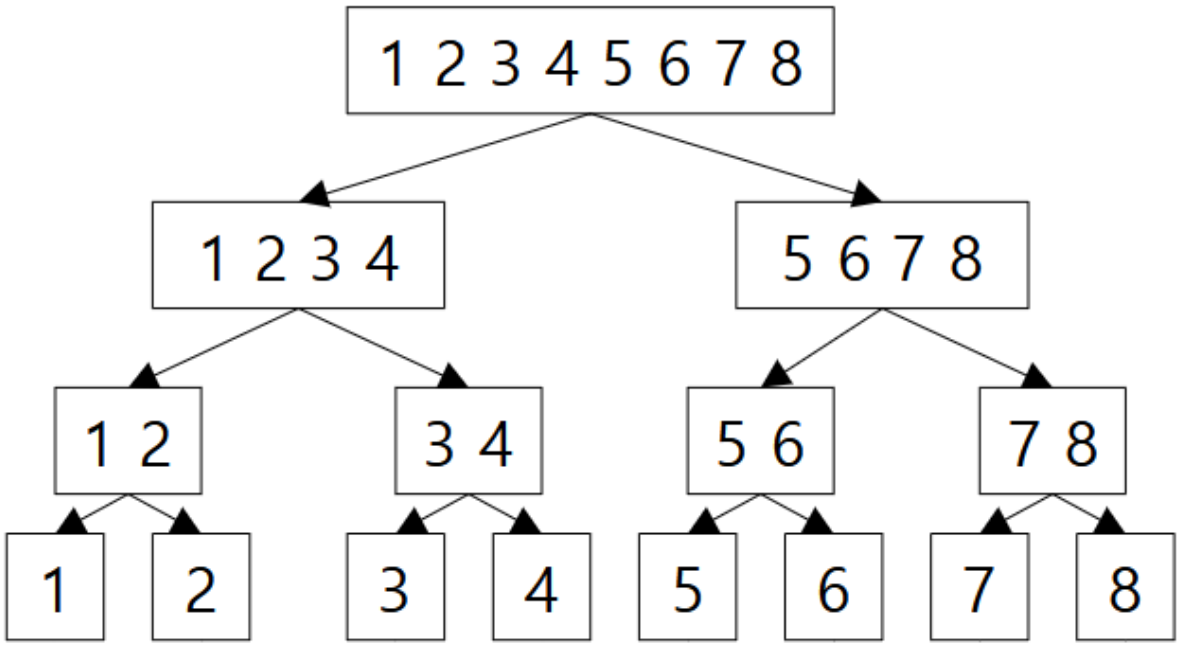
```
while i<len(left) and j<len(right):
        if len(left[i])<len(right[j]):#한쪽이 더길면
            arr.append(left[i])
            i += 1
        elif len(left[i])==len(right[j]):#길이가 같을때
            for c in range(len(left[i])):#길이만큼 반복함
                if left[i][c]<right[j][c]:#각 단어비교
                    arr.append(left[i])
                    i += 1
```

```
                    break
                elif left[i][c]>right[j][c]:
                    arr.append(right[j])
                    j += 1
                    break
                if c == len(left[i])-1:  # 끝까지 비교해 두단어가 같다면
                    arr.append(left[i])
                    i += 1
                    break

        else:
            arr.append(right[j])
            j += 1
```

◆ 머지소트 트리

- 머지소트를 자세히 보면 트리와 방식이 유사하다.



- 머지소트 트리는 세그먼트 트리의 파생형으로 위 그림처럼 각 node가 정렬된 구간을 보유하는 형태로 되어있다.
- 먼저 리프 노드에 원소를 삽입한 후 부모에 두 자식 노드를 merge 한 결과를 넣는 것으로 각 노드가 정렬된 구간을 보유할 수 있게 된다.
- 분할 정복을 이용하는 합병 정렬(Merge Sort)을 메모이제이션하는 Merge Sort Tree

◎ **메모이제이션**

- **컴퓨터 프로그램이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술**이다. 동적 계획법의 핵심이 되는 기술

◎ **세그먼트 트리(Segment Tree)란?**

- **여러 개의 데이터가 존재할 때 특정 구간의 합(최솟값, 최댓값, 곱 등)을 구하는 데 사용하는 자료구조**이다.
- **트리 종류 중에 하나로 이진 트리의 형태이며, 특정 구간의 합을 가장 빠르게 구할 수 있다**는 장점이 있다. (`O(logN)`)
- https://velog.io/@kimdukbae/자료구조-세그먼트-트리-Segment-Tree

- 쉽게 말해서 합병정렬을 좀 더 빠르게 처리하는 세그먼트 트리를 변형하여 만든 구조
- 머지소트 시에 일어나는 각 배열들의 중간 상태를 저장하는 자료 구조
- 배열 [4, 3, 7, 1, 2, 1, 2, 5]가 있다고 가정

<table>
<tr><td colspan="8">[1, 1, 2, 2, 3, 4, 5, 7]</td></tr>
<tr><td colspan="4">[1, 3, 4, 7]</td><td colspan="4">[1, 2, 2, 5]</td></tr>
<tr><td colspan="2">[3, 4]</td><td colspan="2">[1, 7]</td><td colspan="2">[1, 2]</td><td colspan="2">[2, 5]</td></tr>
<tr><td>[4]</td><td>[3]</td><td>[7]</td><td>[1]</td><td>[2]</td><td>[1]</td><td>[2]</td><td>[5]</td></tr>
</table>

- 머지소트 트리는 각 노드가 한 값이 아닌 리스트를 들고 있는 세그먼트 트리이다.
- 이미 자신이 의미하는 구간에 들어있는 모든 값들을 정렬된 형태로 들고 있음. (중복 값 포함)
- 각 노드에 저장된 배열들을 모두 정렬되어 있음이 보장
- 값의 개수가 N일 때, 이 트리의 시간 복잡도는 머지 소트와 동일한 **O(NlogN)**이며, 공간 복잡도 또한 **O(NlogN)**입니다.
- 어떤 노드가 쿼리의 구간에 완전히 포함된다면, 그 자리에서 자신이 가진 리스트에 대해 이분 탐색을 하면 k보다 큰 값의 개수를 아주 쉽게 알 수 있습니다. 따라서 각 쿼리 처리에 **O((logN)^2)**
  의 시간이 걸립니다.

- 머지 소트 트리는 부모 노드에 각 자식 노드를 머지한 결과 값을 저장한 **배열을 저장**
- 일반적인 세그먼트 트리와 다르게 하나의 노드에 값이 아닌 배열을 저장
- 모든 i에 대해 트리의 i번째 리프노드에 i번째 원소를 넣어준 뒤, bottom-up 방식으로 두 노드를 합쳐주면 된다.
- **머지소트**는 top-bottom 방식인 반면에, **머지소트트리**는 bottom-up 방식으로 리프노드부터 위로 거슬러 올라간다.

◆ 그 밖의 또다른 분할 정복 기법의 대표적인 정렬 방식 - 퀵 정렬

- 1+1으로 퀵정렬 하나를 더 배워보자
- 퀵정렬은 찰스 앤터니 리처드 호어가 개발한 정렬 알고리즘

주어진 배열을 두 개로 분할하고, 각각을 정렬한다.

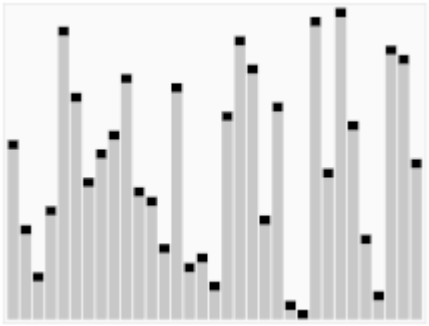
- 병합 정렬과 동일?

다른 점 1 : 병합 정렬은 그냥 두 부분으로 나누는 반면에, 퀵 정렬은 분할할 때, 기준 아이템(pivot item) 중심으로, 이보다 작은 것은 왼편, 큰 것은 오른편에 위치시킨다.

다른 점 2 : 각 부분 정렬이 끝난 후, 병합정렬은 "병합"이란 후처리 작업이 필요하나, 퀵 정렬은 필요로 하지 않는다.

알고리즘

```
quickSort(A[], l, r)
      if l < r
         s ← partition(a, l, r)
         quickSort(A[], l, s - 1)
         quickSort(A[], s + 1, r)
```

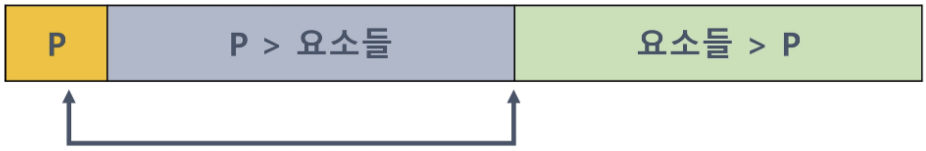


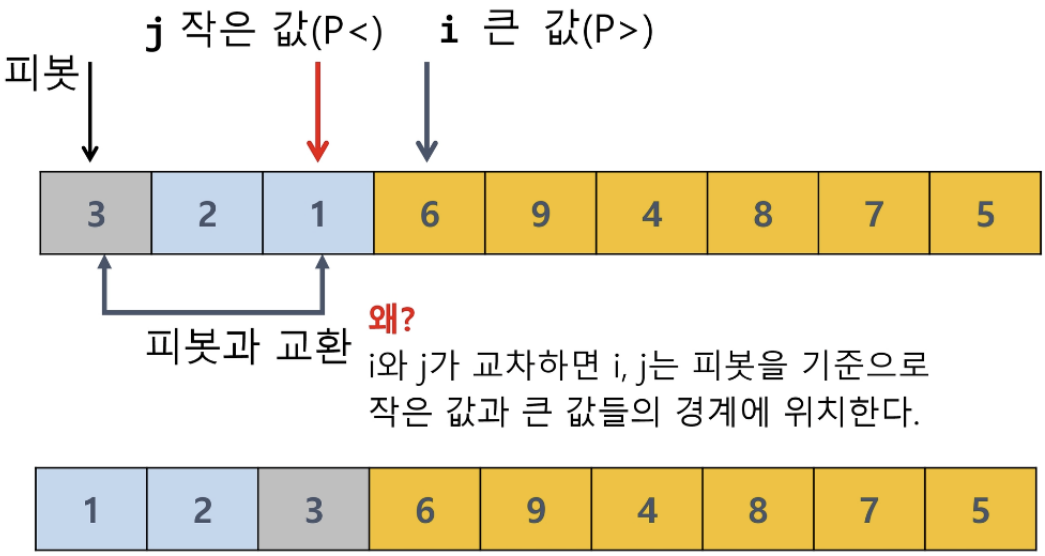
1. 리스트 가운데서 하나의 원소를 고른다. 이렇게 고른 원소를 **피벗**이라고 한다.
2. 피벗 앞에는 피벗보다 값이 작은 모든 원소들이 오고, 피벗 뒤에는 피벗보다 값이 큰 모든 원소들이 오도록 피벗을 기준으로 리스트를 둘로 나눈다. 이렇게 리스트를 둘로 나누는 것을 **분할**이라고 한다. 분할을 마친 뒤에 피벗은 더 이상 움직이지 않는다.
3. 분할된 두 개의 작은 리스트에 대해 재귀(Recursion)적으로 이 과정을 반복한다. 재귀는 리스트의 크기가 0이나 1이 될 때까지 반복된다.

아이디어

- P(피봇)값들 보다 큰 값은 오른쪽, 작은 값들은 왼쪽 집합에 위치하도록 한다.

- 피봇을 두 집합의 가운데에 위치시킨다.





- 퀵소트는 시간복잡도가 최악의 경우 O(n^2)라는 매우 느린 값을 가지면서도 빠른 정렬 알고리즘이라고 알려져 있다.
- 사전에 어떠한 설계를 더해줌으로써 위와 같은 최악의 경우가 나오지 않게 개선하는 것이 가능하다.
- 퀵소트는 피벗이라는 기준점이 존재하고, 피벗을 선택하는 방식에 따라서도 정렬 속도가 달라질 수 있다. (ex- 첫번째 인덱스, 중간 인덱스, 마지막 인덱스 등)
- 하지만 평균적으로 매우 빠른 속도를 자랑하는 정렬 방식이다.
- 또한 병합정렬과 다르게 불안정 정렬이다.

- **퀵정렬 쉽게 설명한 예시**

→ https://spacebike.tistory.com/29

- 과정이 끝나면 피벗이 없어진 게 아니라 이미 피벗을 기준으로 앞뒤정렬이 필요 없기 때문에 더 이상 움직이지 않는 것

◆ 퀵정렬이 빠른 이유?

- **매 단계에서 적어도 1개의 원소가 자기 자리를 찾게 되므로 이후 정렬할 개수가 줄어든다.**
- 예를 들어 처음 피벗 선정할 때 그 피벗 기준으로 한쪽만 값이 있을 때 (피벗이 전체의 최소나 최대일 경우)에도 어쨌든 피벗은 맨 뒤나 맨 앞이라는 게 정해지므로..

◆ 병합정렬과 퀵정렬의 비교

정렬 종류	장점	단점	안정성	평균시간복잡도
버블정렬	간단하다	느리다	안정	$O(n^2)$
선택정렬	간단하다	느리다	불안정	$O(n^2)$
삽입정렬	간단하다	느리다	안정	$O(n^2)$
병합정렬	빠르다	복잡, 메모리 필요	안정	O(nlogn)
퀵정렬	빠르다	복잡하다.	불안정	O(nlogn)

퀵정렬 VS 병합 정렬 비교

구분	퀵 정렬	병합 정렬
부분 배열의 구획	나뉘어진 배열은 여러 비율로 나뉜다.	배열은 항상 반으로 나뉜다.
최악의 경우 시간복잡도	O(n^2)	O(nlogn)
사용 용도	작은 크기의 배열에서 잘 동작	어떤 크기의 Dataset에서도 적절히 동작
효율성	작은 크기 Dataset에서는 병합 정렬보다 빠르다.	큰 Dataset에서는 퀵 정렬보다 빠르다.
정렬 방식	내부 정렬	외부 정렬
별도 저장 공간	불 필요	필요
참조 지역성	좋음	퀵 정렬 대비 나쁨
Stable	X(그러나 구현 방식에 따라 가능)	O

- **병합 정렬은 정렬 시, 순차 탐색을 하며 퀵 정렬은 Random 탐색을 하는 경우가 많아, 연결리스트는 퀵 정렬이 불리**

- **퀵정렬 파이썬 구현**

```
# 파이썬 퀵정렬 구현
def qsort(data):
    if len(data) <= 1:
        return data

    pivot,tail = data[0], data[1:]

    left = [ x for item in tail if pivot > x ]
```

```
    right = [ x for item in tail if pivot <= x ]

    return qsort(left) + [pivot] + qsort(right)
```

- **병합정렬 파이썬 구현**

```
# 분할힘수
def split_func(data):
    medium = int(len(data) / 2)
    print (medium)
    left = data[:medium]
    right = data[medium:]
    print (left, right)

# 머지함수
def merge(left, right):
    merged = list()
    left_point, right_point = 0, 0 #인덱스번호

    # case1 - left/right 둘다 있을때(데이터가 있을때)
    while len(left) > left_point and len(right) > right_point:
        if left[left_point] > right[right_point]:
            merged.append(right[right_point])
            right_point += 1
        else:
            merged.append(left[left_point])
            left_point += 1

    # case2 - left 데이터가 없을 때
    while len(left) > left_point:
        merged.append(left[left_point])
        left_point += 1

    # case3 - right 데이터가 없을 때
    while len(right) > right_point:
        merged.append(right[right_point])
        right_point += 1

    return merged
```

◆◆

◆◆