**자료구조**

**효율적인 접근 및 수정을 위한 데이터의 조직, 관리, 저장.**

**Ex)데이터 값의 모임, 데이터간의 관계**

**배열**

같은 타입의 변수들로 이루어진 유한 집합.

메모리에 연속적으로 할당하고, 처음에 선언한대로 고정된 크기를 가짐.

배열의 지정된 인덱스를 넘어가는 숫자를 쓰면

인덱스를 이용해서 데이터를 가져올 수 있다. 인덱스를 이용한 데이터의 조회는 매우 빠르게 처리. 하지만 인덱스를 이용해서 데이터를 가져오려면 데이터에 대한 인덱스의 값이 고정되어야 합니다. 자연스럽게 어떤 엘리먼트가 삭제되면 삭제된 상태를 빈 공간으로 남겨둬야 합니다. 이것은 메모리의 낭비를 초래합니다. 또한 배열에 데이터가 있는지 없는지를 체크하는 로직이 필요하다는 의미

**스택 메모리**

**리스트**

**https://opentutorials.org/module/1335/8636**

배열이 가지고 있는 인덱스라는 장점을 버리고 빈틈없는 데이터의 적재라는 장점을 취한 자료구조

삭제하면 빈값이 그대로 자리를 지키고있는 리스트와는 다르게 다음 엘리먼트가 그 자리를 채우게 되니 인덱스가 더 이상 식별자로서 기능할 수 없는것이다.

**연결리스트**

각 노드가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터를 저장하는 자료구조

**이중연결리스트**

**선행노드를 바로 찾기 힘들다는 단순 연결리스트의 단점을 보완한 자료구조로 특정 노드에서 양방향으로 자유롭게 이동할 수 있음, 메모리 공간을 많이 쓴다는 단점이 있음**

**새 노드를 추가할 때 이전 노드가 새 노드에게 포인터를 줘야하기 때문에 이전 노드를 찾아야하는것과는 다르게 이중연결리스트는 찾을필요가 없다는 장점이 있다.**

**원형연결리스트**

**마지막의 노드의 링크가 첫번째 노드를 기르켜 원형으로 연결되어있는 리스트로 하나의 노드를 따라가면 모든 노드를 방문가능하다는 특징이 있다. 마지막 노드가 null인 일반 연결리스트랑 달리 모든 노드의 링크가 null이 아니다.**

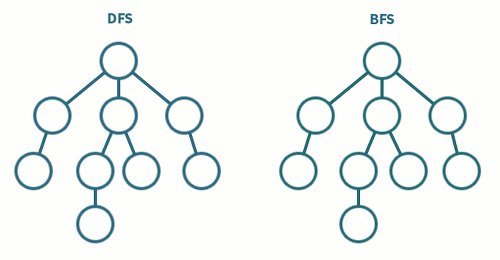
**배열과 리스트의 차이점**

**벡터**

**스택**

**데이터를 쌓아올린 형태의 자료구조, 한곳(top)에서 삽입 삭제가 이루어지는 후입선출형태. Ex)책쌓기**

**깊이 우선탐색 너비우선탐색**

****

**스택 자식기준 가지부터 깊게**

**모든 노드를 방문하려고 할 때 사용**

**너비보다 간단하고 느림**

**큐 부모기준 가지부터 넓게**

지구 상에 존재하는 모든 친구 관계를 그래프로 표현한 후 Ash 와 Vanessa 사이에 존재하는 경로를 찾는 경우

\* 깊이 우선 탐색의 경우 - 모든 친구 관계를 다 살펴봐야 할지도 모름

\* 너비 우선 탐색의 경우 - Ash와 가까운 관계부터 탐색

**큐**

**데이터를 밀어넣는 형태의 자료구조. 한곳에서는 넣고 한곳에서는 삭제하는 선입선출형태 Ex) 매표소**

**트리**

**계층(부모자식)으로 구조화할 때 사용하는 비선형 자료구조**

**이진트리**

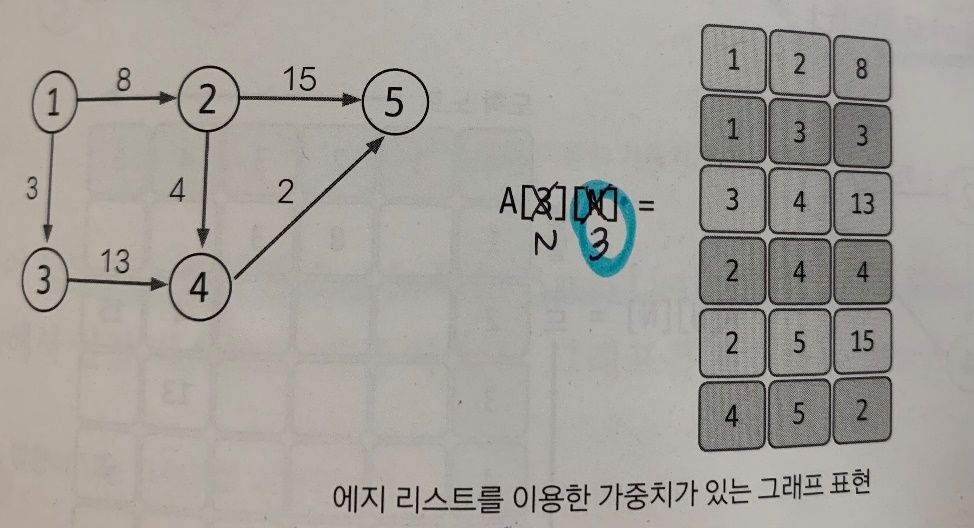
**노드가 자식을 두개까지 가지는 트리**

**그래프**

**트리의 상위개념으로서 노드와 노드들을 연결하는 간선(에지)이 모인것**

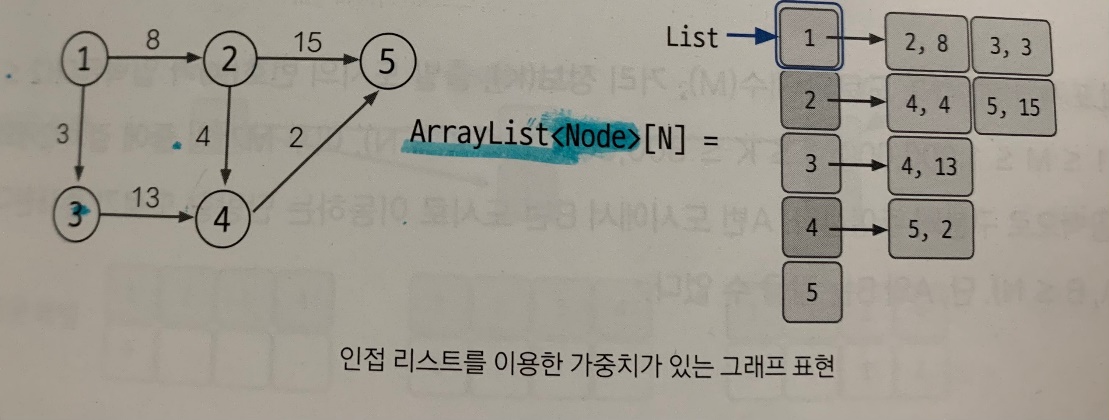
**인접리스트와 인접행렬로서 표현함**

**에지리스트**



**구현하기 쉬우나 특정 노드와 관련된 에지를 탐색하는 것이 어렵다.**

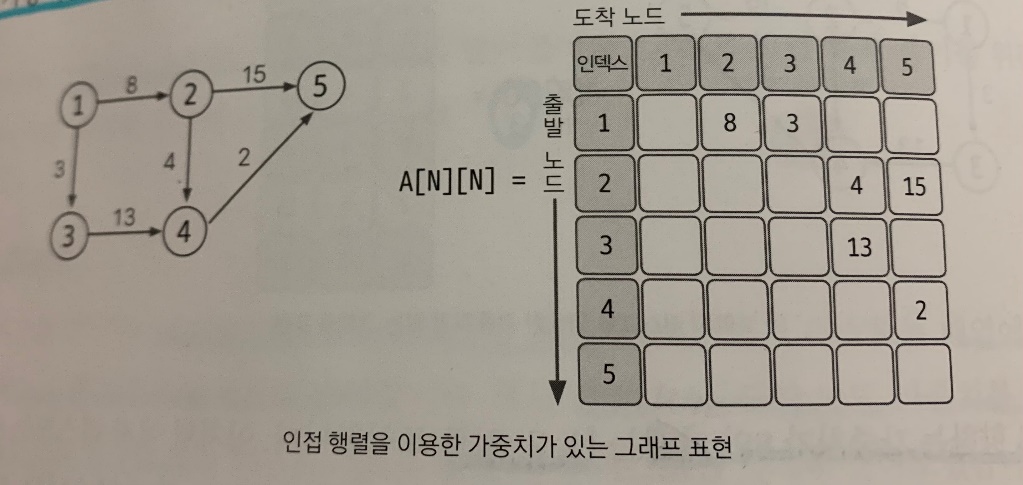
**인접리스트**



복잡하나 노드과 연결되어 있는 에지를 탐색하는 시간은 매우 뛰어나다.

* 노드 개수가 커도 공간 효율이 좋아 메모리 초과 에러도 발생하지 않는다.
* 그냥 좋음

**인접 행렬**



* 두 노드를 연결하는 에지의 여부와 가중치값을 배열에 직접 접근하면 바로 확인할 수 있다.
* 노드와 관련되어있는 에지를 탐색하려면 N번 접근해야하므로 노드 개수에 비해 에지가 적을 때는 공간 효율이 떨어진다.

**투포인터 슬라이딩윈도우 구간합**

**포인터 두개를 사용해 만드는 부분 배열**

**배열 속의 배열을 이용해 값을 찾는 것**

**배열을 하나 더 만드는거니 메모리를 두배로 사용**

**대신 찾고자하는 값을 더 빠르게 구할수 있으니 성능상의 우위를 가가지 되는 것**

**알고리즘 자체가 메모리와 성능의 저울질**

**Sorting 정렬**

**버블 정렬 선택 정렬 삽입 정렬**

**Swap 교환 comparison 비교 를 통해 데이터 이동으로 정렬**

**버블 정렬은 무식하게 첫인덱스부터 끝까지 비교 교환**

**그다음 인덱스부터 끝까지 돌리고**

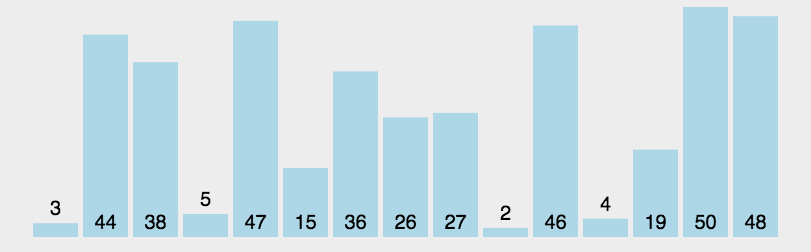
**끝 전 인덱스랑 끝까지 비교할때까지 미친듯이 돌려야해서 구림**

**선택 정렬도 몽땅 비교해야하는 것은 맞지만 교환을 몇번 하지 않는다는것이 장점**

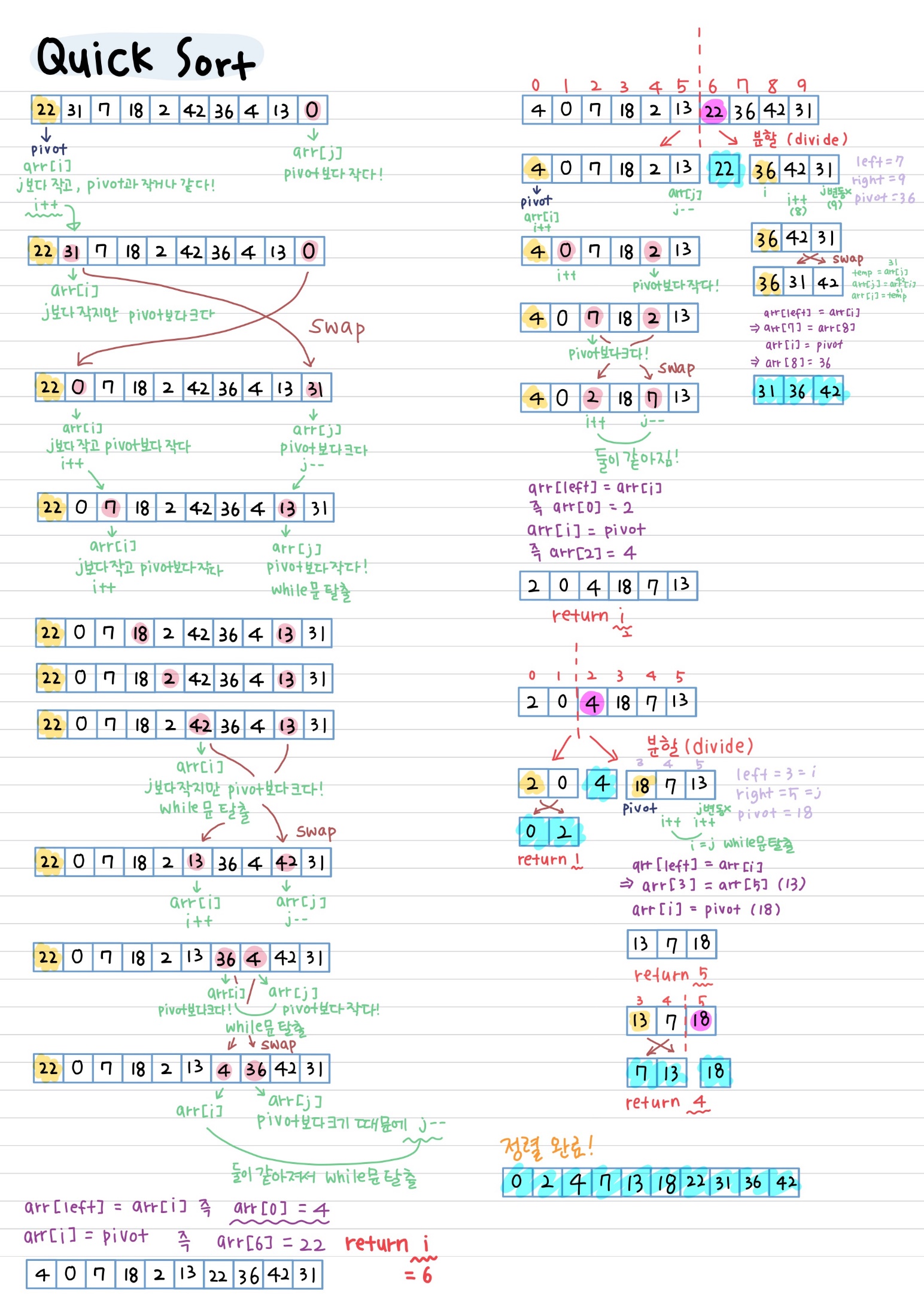
* **버블 정렬**
  + **인접한 원소끼리 비교하여 교환하는 방식**
  + 셋 중에 제일 느리지만 단순함
* **선택 정렬**
  + **최솟값을 찾아서 맨 앞으로 이동하는 방식**
  + 버블 정렬보다 좋음
* **삽입 정렬**
  + **앞에서부터 차례대로 이미 정렬된 부분과 비교하여 교환하는 방식**
  + 셋 중에 제일 빠르지만 배열이 길어질수록 효율성이 떨어짐

**모두 시간 복잡도는 O(n²)**  
선택 정렬과 삽입 정렬은 사용할 메모리가 제한적인 경우에 사용하면 좋음

**퀵 정렬**

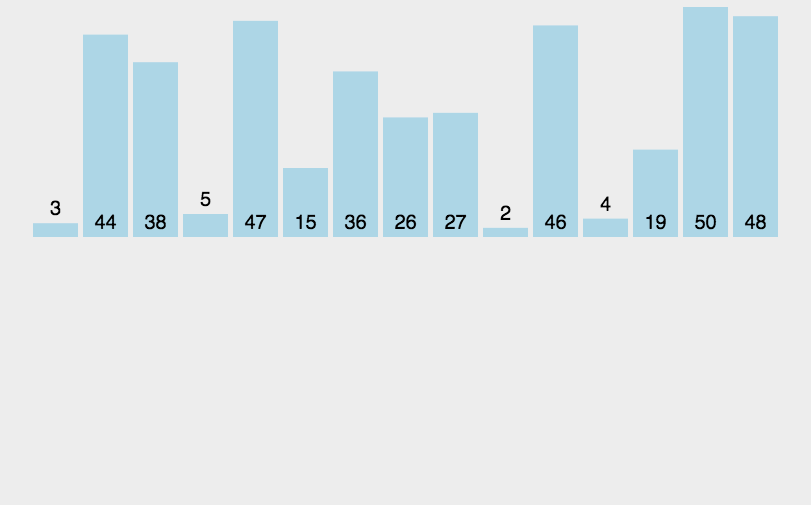
**분할과 재귀**

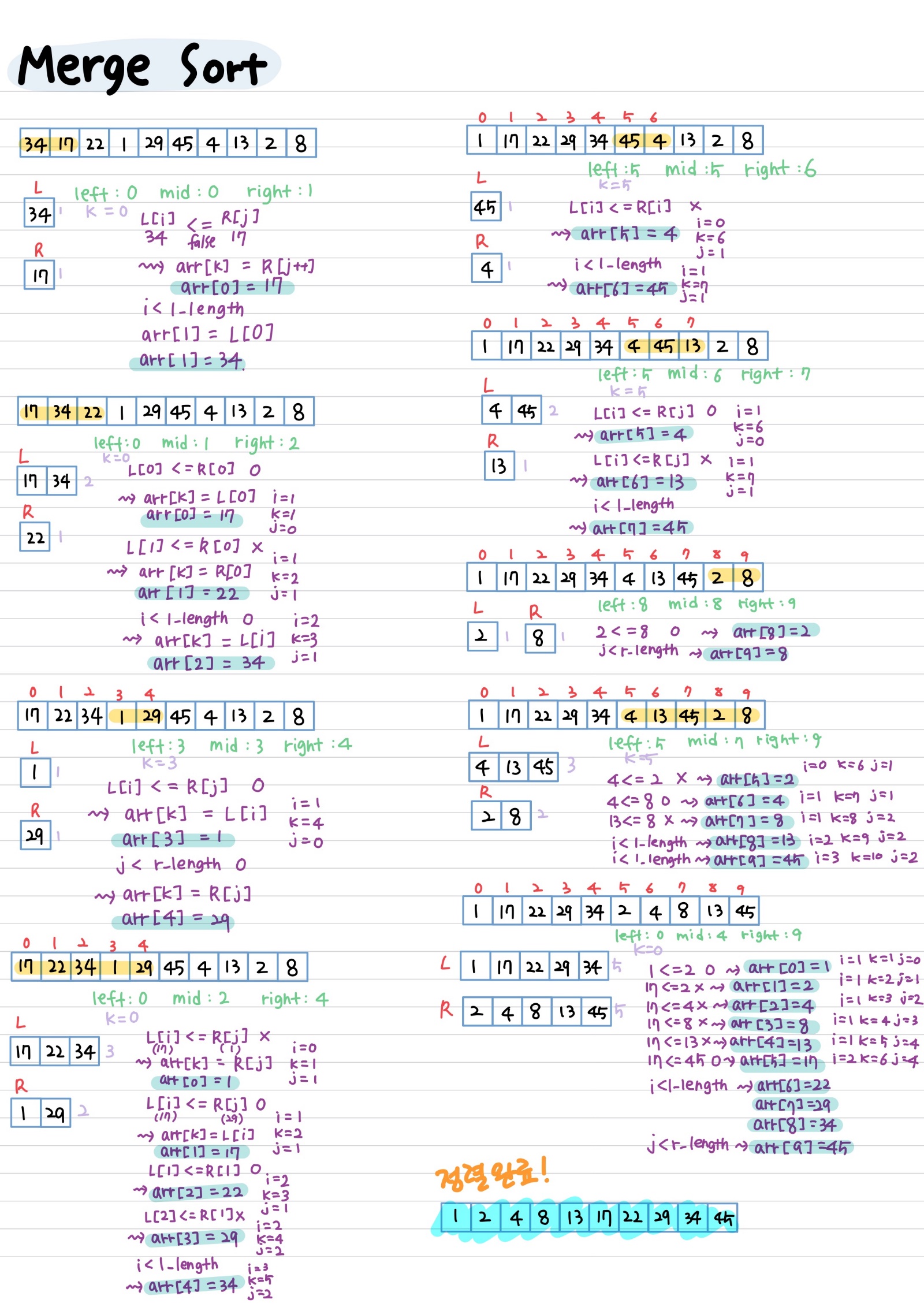
1. 배열을 pivot의 값을 기준으로 값이 작은 배열과 큰 배열로 정렬한다.
2. 부분 배열의 크기가 충분히 작다면 그냥 정렬하고, 아니라면 재귀 호출을 통해 다시 이 과정을 반복한다.

* 매 정렬마다 pivot 하나는 위치가 정해짐이 보장되었으므로 끝남을 보장 가능.
* pivot을 어떻게 정하냐에 따라 최적화 가능.
* 

**병합 정렬**

**일정량 병합과 정렬, 정렬된걸 병합, 그걸 정렬 의 반복**

1. 입력 배열을 2개의 부분 배열로 분할한다.
2. 부분 배열의 크기가 충분히 작다면 그냥 정렬하고, 아니라면 재귀 호출을 통해 다시 이 과정을 반복한다.
3. 정렬된 부분 배열을 하나의 리스트로 병합한다.
4. 



**차이점**

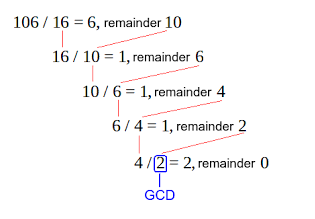
* 퀵소트는 불안정 정렬, 병합 정렬은 안정 정렬이다.
* 퀵 정렬은 최악의 경우(오름차순 정렬이거나 내림차순 정렬일 경우) O(n^2)의 시간 복잡도를 가지지만 병합 정렬은 O(n log n)으로 동일하다.
* 아래 cache의 지역성을 보면 퀵 정렬이 평균의 시간 복잡도에서는 빠를지도 모른다.
* 병합 정렬은 추가 메모리 공간을 필요로 한다.

기수 정렬

정수론

소수 구하기 오일러의 피 유클리드 호제법 확장

유클리드 호제법



작은수를 나머지를 나누기, 0이 될 때 마지막 나눈수가 gcd

오일러의 피

1~n까지의 범위중 n과의 서로소 개수

