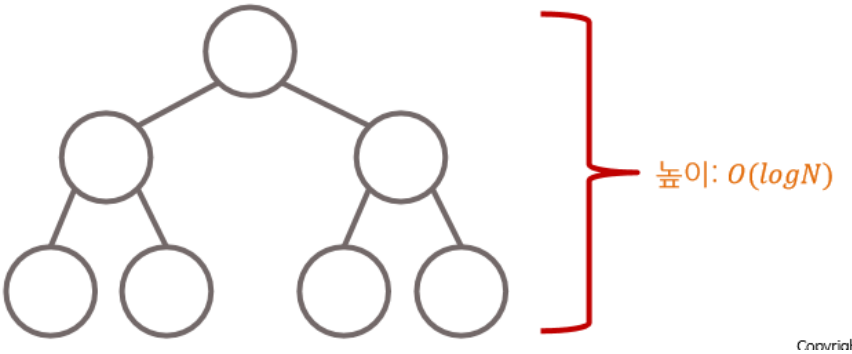
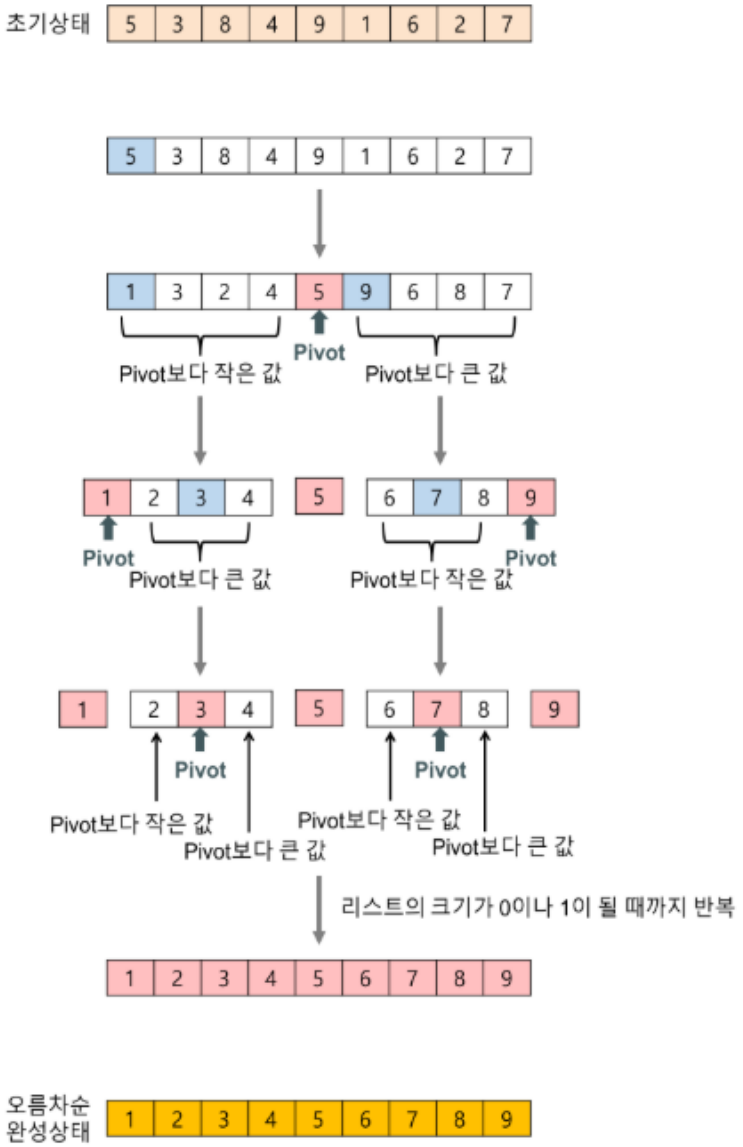
Quick Sort

1. 이론

퀵 정렬이란 피벗을 기준으로 큰 값과 작은 값을 서로 교체하는 정렬 기법입니다. 값을 서로 교체하는 데에 N번, 엇갈린 경우 교체 이후에 원소가 반으로 나누어지므로 전체 원소를 나누는 데에 평균적으로 $logN$번이 소요되므로 평균적으로 $\theta(NlogN)$ 의 시간 복잡도를 가집니다.

원소를 절반씩 나눌 때 $logN$의 시간 복잡도가 나오는 대표적인 예시는 완전 이진 트리입니다. 이러한 완전 이진 트리 형태는 흔히 컴퓨터 공학에서 가장 선호하는 이상적인 형태입니다.



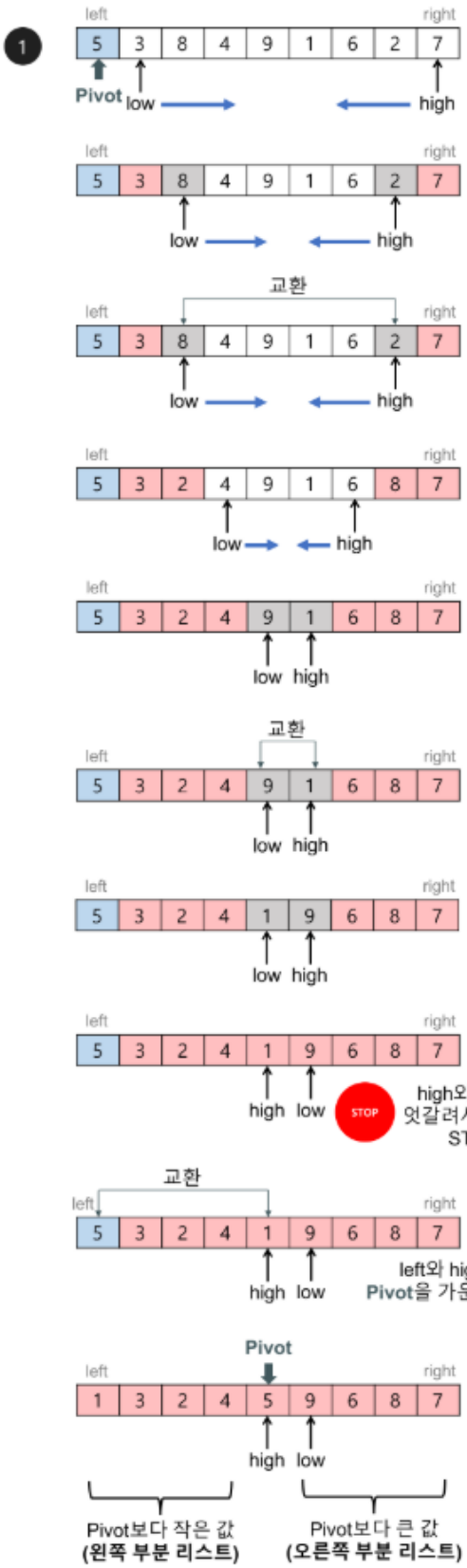
- 하나의 리스트를 피벗(pivot)을 기준으로 두 개의 비균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다.
- 퀵 정렬은 다음의 단계들로 이루어진다.
  - **분할(Divide):** 입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열(피벗을 중심으로 왼쪽: 피벗보다 작은 요소들, 오른쪽: 피벗보다 큰 요소들)로 분할한다.
  - **정복(Conquer):** 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 **순환 호출** 을 이용하여 다시 분할 정복 방법을 적용한다.
  - **결합(Combine):** 정렬된 부분 배열들을 하나의 배열에 합병한다.
  - 순환 호출이 한번 진행될 때마다 최소한 하나의 원소(피벗)는 최종적으로 위치가 정해지므로, 이 알고리즘은 반드시 끝난다는 것을 보장할 수 있다.
- 

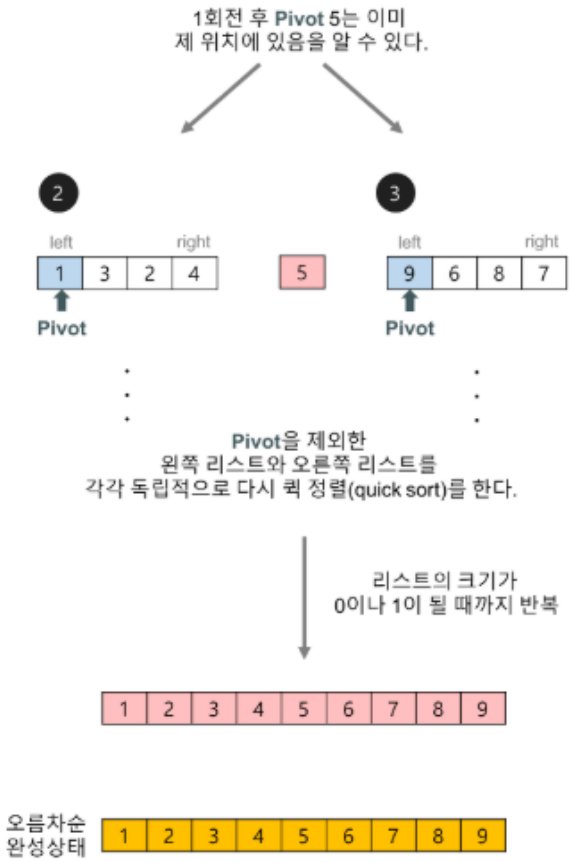
2. 단점

퀵 정렬은 편향된 분할이 발생할 때 연산의 양이 $O(N^2)$입니다. 따라서 실제로 정렬을 함에 있어서는 퀵 정렬을 직접 구현하지 않습니다. 따라서 C++의 Algorithm 라이브러리를 사용합니다. Algorithm 라이브러리의 sort() 함수는 퀵 정렬을 기반으로 하되 $O(NlogN)$을 보장합니다.

3. 과정

- *퀵 정렬에서 피벗을 기준으로 두 개의 리스트로 나누는 과정(c언어 코드의 partition 함수의 내용)*
- 초기상태





- 피벗 값을 입력 리스트의 첫 번째 데이터로 하자. (나는 임의의 값이어도 상관없다.)
- 2개의 인덱스 변수(low, high)를 이용해서 리스트를 두 개의 부분 리스트로 나눈다.
- 1회전: 피벗이 5인 경우,
  i. low는 왼쪽에서 오른쪽으로 탐색해가다가 피벗보다 큰 데이터(8)을 찾으면 멈춘다.
  ii. high는 오른쪽에서 왼쪽으로 탐색해가다가 피벗보다 작은 데이터(2)를 찾으면 멈춘다.
  iii. low와 high가 가리키는 두 데이터를 서로 교환한다.
  iv. 이 탐색-교환 과정은 low와 high가 엇갈릴 때까지 반복한다.
- 2회전: 피벗(1회전의 왼쪽 부분리스트의 첫 번째 데이터)이 1인 경우,
  - 위와 동일한 방법으로 반복한다.
- 3회전: 피벗(1회전의 오른쪽 부분리스트의 첫 번째 데이터)이 9인 경우,
  - 위와 동일한 방법으로 반복한다.

4. Logic

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

void QuickSort(int *&pList, int nStart, int nEnd)
{
        if (nStart >= nEnd)
                return;

        int nPivot = nStart;
        int nLeft = nStart + 1;
        int nRight = nEnd;

        while (nLeft <= nRight)
        {
                while (nLeft <= nEnd && pList[nLeft] <= pList
[nPivot])
                        nLeft++;
```

```
                while (nRight > nStart && pList[nRight] >= pLi
st[nPivot])
                        nRight--;

                if (nLeft > nRight)
                {
                        int nTemp = pList[nPivot];
                        pList[nPivot] = pList[nRight];
                        pList[nRight] = nTemp;
                }
                else
                {
                        int nTemp = pList[nLeft];
                        pList[nLeft] = pList[nRight];
                        pList[nRight] = nTemp;
                }
        }

        QuickSort(pList, nStart, nRight - 1);
        QuickSort(pList, nRight + 1, nEnd);
}

void Show(int *&pList)
{
        int nSize = _msize(pList) / sizeof(int);

        for (int i = 0; i < nSize; ++i)
        {
                printf("[%d] : %d\n", i + 1, pList[i]);
        }
```

```
}

int main(void)
{
	int nSize = 0;
	printf("Size : ");
	scanf("%d", &nSize);

	int *pList = (int*)malloc(sizeof(int) * nSize);
	for (int i = 0; i < nSize; ++i)
	{
		printf("[%d] : ", i + 1);
		scanf("%d", &pList[i]);
	}

	QuickSort(pList, 0, nSize - 1);
	printf("---- After ----\n");
	Show(pList);

	system("pause");
	return 0;
}
```