배열 (Array)

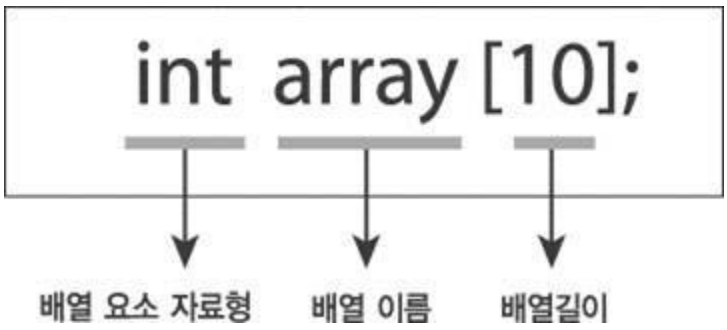
배열을 사용하면 한 번에 여러 개의 변수를 생성하고 저장할 수 있다
배열을 사용하면 대량의 같은 종류 데이터를 효율적이고 간편하게 처리할 수 있다.
배열을 구성하는 각각의 항목을 배열 요소(array element) 또는 배열 원소라고 한다. 배열 요소에는 번호가 붙어 있는데 이것을 인덱스(index) 또는 첨자(subscript)라고 부른다.

▶ 배열의 특징

1. 배열은 메모리의 연속적인 공간에 저장된다.
2. 배열의 가장 큰 장점은 서로 관련된 데이터를 차례로 접근하여서 처리할 수 있다는 점이다.

배열은 근본적으로 데이터들에게 하나하나 이름을 붙이지 않고 전체 집단에 하나의 이름을 부여한 다음, 각각의 데이터에 숫자로 된 번호를 붙여서 접근하는 방법이다.

▶ 배열의 선언



※ 배열의 크기를 나타날 때는 항상 상수를 사용하여야 한다.

▶ 배열 요소 접근

배열 요소는 변수와 100% 동일하다. 배열 요소에 값을 저장할 수 있고 배열 요소에 저장된 값을 꺼낼 수도 있다.

▶ 배열과 반복문

배열의 가장 큰 장점은 반복문을 사용하여서 배열의 요소를 간편하게 처리할 수 있다는 점이다.

▶ 인덱스의 범위

배열을 사용할 때 아주 조심해야 하는 부분이 배열 인덱스의 범위이다. 인덱스가 배열의 크

기를 벗어나게 되면 프로그램에 치명적인 오류를 발생시킨다.

▶ 배열의 초기화

모든 배열 요소에 초기값을 부여하는 것이 원칙이다. 그러나 만약 초기값의 개수가 요소들의 개수보다 적은 경우에는 앞에 있는 요소들만 초기화된다. 나머지 배열 요소들은 모두 0으로 초기화 된다.
초기화만 하고 배열의 크기를 비워놓으면 컴파일러가 자동으로 초기값들의 개수만큼의 배열 크기를 잡는다.

▶ 배열 요소의 개수를 계산하는 방법

int size = sizeof(배열이름) / sizeof(배열이름[0]);
sizeof(배열이름)은 전체 배열의 크기이고 sizeof(배열이름[0])은 배열 요소의 크기이다. 따라서 전체 배열의 크기를 요소의 크기로 나누어주면 배열의 크기를 얻을 수 있다.

▶ 배열의 복사

하나의 배열을 다른 배열로 복사하려면 반복 구조를 사용하여서 배열의 요소를 하나씩 복사해주어야 한다.

▶ 2차원 배열

2개의 대괄호를 이용하여 2차원 배열을 선언할 수 있다. 첫 번째 대괄호 안에는 행의 개수, 두 번째 대괄호 안에는 열의 개수를 적는다.

int arr[2][3];

	0열	1열	2열
0행	arr[0][0]	arr[0][1]	arr[0][2]
1행	arr[1][0]	arr[1][1]	arr[1][2]

▶ 2차원 배열에서 요소 참조

2차원 배열에서 하나의 요소를 참조하려면 2개의 인덱스가 필요하다. 일반적으로 s[i][j]는 배열 s의 I번째 행과 j번째 열의 요소이다. 여기서 첫 번째 인덱스를 행번호라고 하고 두 번째 인덱스를 열 번호라고 한다.

▶ 2차원 배열의 초기화

2차원 배열도 1차원 배열과 마찬가지로 선언과 동시에 초기화 할 수 있다. 다만 같은 행에 속하는 초기값들을 중괄호 { }로 따로 묶어주어야 한다.

```
#include <stdio.h>

void main()
{
    //1.
    int arr1[3][3] = {1,2,3,4,5,6,7,8,9};

    //2. 계단식구조
    int arr2[3][3] = {
        {1,2,3},
        {4,5,6},
        {7,8,9}
                    };
```

정렬

정렬은 컴퓨터 공학에서 가장 기본적이고 중요한 알고리즘 중의 하나로 일상생활에서 많이 사용된다. 또한 정렬은 자료 탐색에 있어서 필수적이다.
일반적으로 정렬시켜야 될 대상은 레코드(record)라고 불린다.
레코드는 필드(field)라는 단위로 나누어진다.
여러 필드 중에서 특별히 레코드와 레코드를 식별해주는 역할을 하는 필드를 키(key)라고 한다.
정렬이란 결국 레코드들을 키 값의 순서로 재배열 하는 것이다.
대개 정렬 알고리즘을 평가하는 효율성의 기준으로는 정렬을 위해 필요한 비교 연산의 횟수와 이동 연산의 횟수이다.
이 횟수들을 정확하게 구하기는 힘들기 때문에 이 횟수들을 빅오 표기법을 이용하여 근사적으로 표현한다.
정렬 알고리즘은 내부 정렬(internal sorting)과 외부정렬(external sorting)로 구분할 수 있다.
내부 정렬은 정렬하기 전에 모든 데이터가 메인 메모리에 올라와 있는 정렬을 의미한다.
외부정렬은 외부 기억 장치에 대부분의 데이터가 있고 일부만 메모리에 올려놓은 상태에서 정렬을 하는 방법이다.
정렬 알고리즘을 안정성(stability)의 측면으로 분류할 수도 있다.
안정성이란 입력 데이터에 동일한 킷값을 갖는 레코드가 여러 개 존재할 경우, 이 레코드들의 상대적인 위치가 정렬 후에도 바뀌지 않음을 뜻한다.

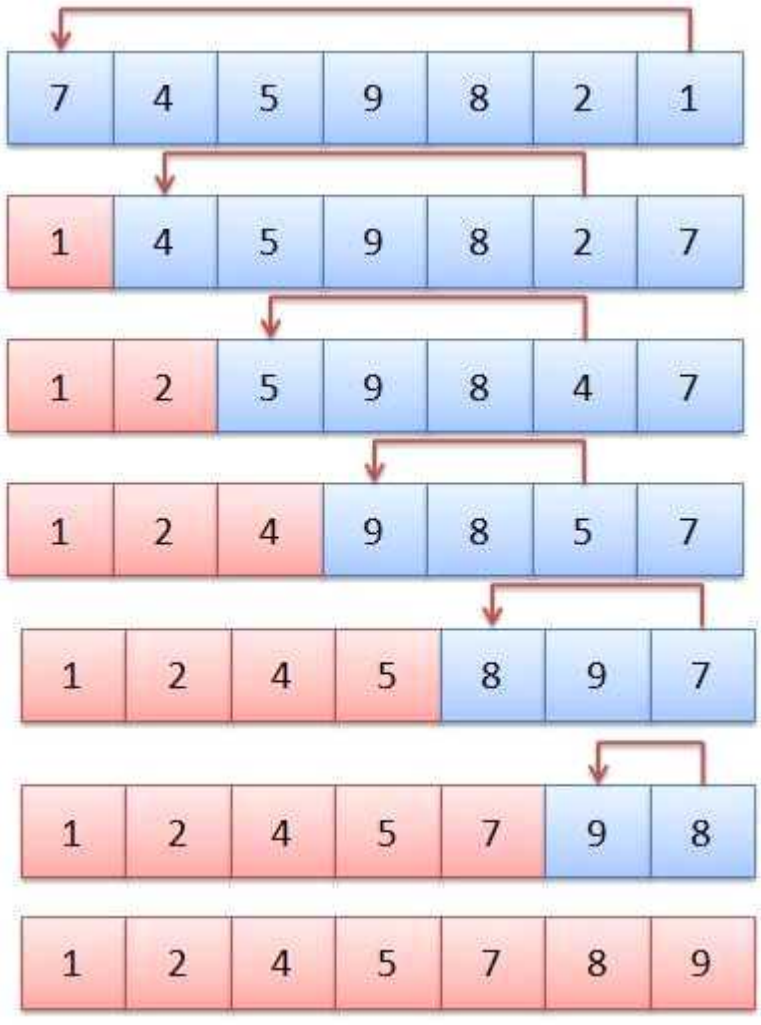
▶ 선택 정렬 (selection sort)

선택 정렬은 가장 이해하기가 쉬운 정렬 방법이다.
초기 상태에서 왼쪽 리스트는 비어 있고 정렬되어야 할 숫자들은 모두 오른쪽 리스트에 들어 있다. 선택 정렬은 오른쪽 리스트에서 가장 작은 숫자를 선택하여 왼쪽 리스트로 이동하는 작업을 되풀이한다.

왼쪽 리스트	오른쪽 리스트	설명
( )	(5,3,8,1,2,7)	초기상태
(1)	(5,3,8,2,7)	1선택
(1,2)	(5,3,8,7)	2선택
(1,2,3)	(5,8,7)	3선택
(1,2,3,5)	(8,7)	5선택
(1,2,3,5,7)	(8)	7선택
(1,2,3,5,7,8)	()	8선택

[그림 12-8] 선택 정렬의 원리

메모리를 절약하기 위하여 입력 배열 외에 추가적인 공간을 사용하지 않는 선택 정렬 알고리즘을 제자리 정렬(in-place sorting)이라고 한다.



입력 배열에서 최소값을 발견한 다음, 이 최소값을 배열의 첫 번째 요소와 교환한다. 다음에는 첫 번째 요소를 제외한 나머지 요소들 중에서 가장 작은 값을 선택하고 이를 두 번째 요

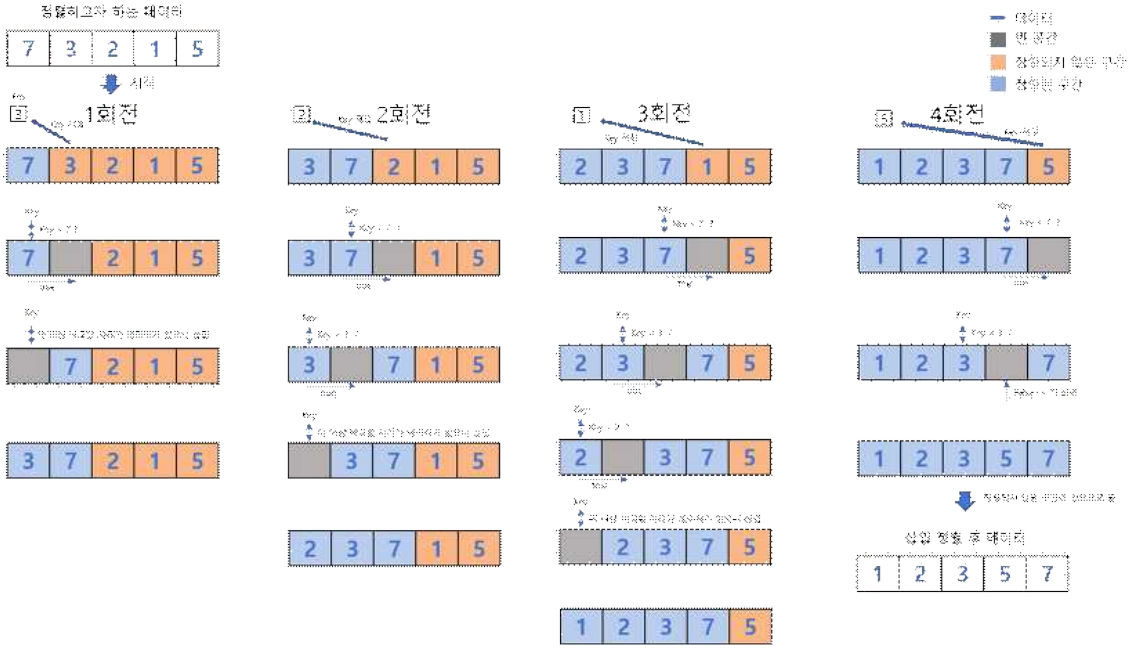
소와 교환한다. 이 절차를 (숫자 개수-1)만큼 되풀이한다.

▶ 선택 정렬의 분석

비교 횟수
외부루프는 n-1번 실행될 것이고 내부루프는 0에서 n-2까지 변하는 I에 대하여 (n-1) - I번 반복될 것이다. 키 값들의 내부 루프 안에서 이루어지므로 전체 비교횟수는 $O(n^2)$이다.
교환 횟수
외부 루프의 실행 횟수와 같으며 한번 교환하기 위하여 3번의 이동이 필요하므로 전체 이동 횟수는 3(n-1)이 된다.
선택 정렬의 장점은 자료 이동 횟수가 미리 결정된다는 점이다.
선택 정렬의 문제점은 안정성을 만족하지는 않는다는 점이다.

▶ 삽입 정렬 (insertion sort)

삽입 정렬은 정렬되어있는 리스트에 새로운 레코드를 적절한 위치에 삽입하는 과정을 반복한다.



▶ 삽입 정렬의 복잡도 분석

삽입 정렬의 복잡도는 입력 자료의 구성에 따라서 달라진다.
이미 정렬되어 있는 경우는 가장 빠르다.
총 비교횟수는 n-1번, 총 이동횟수는 2(n-1)번이 되어 알고리즘 시간 복잡도는 O(n)이다.
최악의 복잡도는 입력 자룍 역순일 경우이다.
총 비교횟수는 $O(n^2)$이며, 총 이동횟수는 $O(n^2)$이다.

삽입정 렬은 비교적 많은 레코드들의 이동을 포함한다.
결과적으로 삽입 정렬은 레코드 양이 많고 레코드 크기가 클 경우에는 적합하지 않다.
반면에 삽입 정렬은 안전한 정렬 방법이다.

▶ 버블 정렬 (bubble sort)

버블 정렬은 인접한 2개의 레코드를 비교하여 크기가 순서대로 되어 있지 않으면 서로 교환하는 비교-교환 과정을 리스트의 왼쪽 끝에서 시작하여 오른쪽 끝까지 진행한다.

▶ 버블 정렬의 시간 복잡도

버블 정렬의 비교횟수는 최선, 평균, 최악의 어떠한 경우에도 항상 일정하게 $O(n^2)$이다.
버블 정렬의 가장 큰 문제점은 순서에 맞지 않은 요소를 인접한 요소와 교환한다는 것이다.

▶ 쉘 정렬 (shell sor)

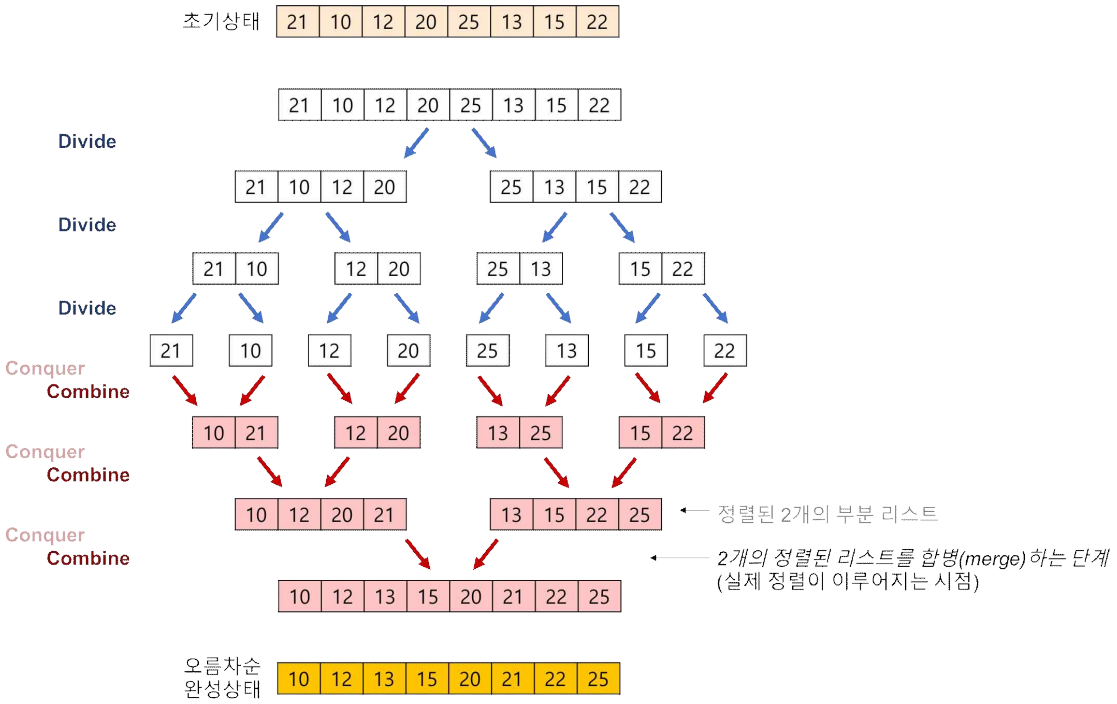
쉘 정렬은 삽입 정렬이 어느 정도 정렬된 배열에 대해서는 대단히 빠른 것에 착안한 방법이다.
삽입 정렬과는 다르게 쉘 정렬은 전체의 리스트를 한 번에 정렬하지 않는다. 대신에 먼저 정렬해야할 리스트를 일정한 기준에 따라 분류하여 연속적이지 않은 여러 개의 부분 리스트를 만들고, 각 부분 리스트를 삽입 정렬을 이용하여 정렬한다. 모든 부분 리스트가 정렬되면 쉘 정렬은 다시 전체 리스트를 더 적은 개수의 부분 리스트로 만든 후에 알고리즘을 되풀이한다. 이 과정은 부분 리스트의 개수가 1이 될 때까지 되풀이된다.
부분 리스트들이 만들어지는 것은 아니고 일정한 간격으로 삽입 정렬을 수행하는 것뿐이다. 따라서 추가적인 공간은 필요 없다.

▶ 쉘 정렬의 분석

삽입 정렬에 비하여 쉘 정렬은 2가지의 장점이 있다.
(1) 연속적이지 않은 부분 리스트에서 자료의 교환이 일어나면 더 큰 거리를 이동한다. 반면 삽입 정렬에서는 한 번에 한 칸씩만 이동된다. 따라서 교환되는 아이템들이 삽입 정렬보다는 최종위치에 더 가까이 있을 가능성이 높아진다.
(2) 부분 리스트는 어느 정도 정렬이 된 상태이기 때문에 쉘 정렬은 기본적으로 삽입 정렬을 수행하는 것이지만 빠르게 수행된다.
쉘 정렬의 시간 복잡도는 대략 최악의 경우에는 $O(n^2)$이지만 평균적인 경우에는 $O(n^{1.5})$로 나타난다.

▶ 합병 정렬 (merge sort)

합병 정렬은 하나의 리스트를 두 개의 균등한 크기로 분할하고 분할된 부분 리스트를 정려한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트를 얻고자 하는 것이다.
합병 정렬은 분할 정복(divide and conquer) 기법에 바탕을 두고 있다. 분할 정복 기법은 문제를 작은 2개의 문제로 분리하고 각각을 해결한 다음 결과를 모아서 원래의 문제를 해결하는 전략이다.
1. 분할(Divide): 입력 배열을 같은 크기의 2개의 부분 배열로 분할한다.
2. 정복(Conquer): 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 기법을 적용한다.
3. 결합(Combine): 정렬된 부분 배열들을 하나의 배열에 통합한다.



▶ 합병 정렬의 복잡도 분석

합병 정렬은 순환 호출 구조로 되어 있다.

$n=2^3$인 경우 순환 호출의 깊이가 3이다. $n=2^k$라고 한다면 순환 호출의 깊이가 k가 될 것임을 쉽게 알 수 있다. 즉 $k = \log_2 n$이 된다.

배열이 부분 배열로 나누어지는 단계에서는 비교 연산이나 이동 연산은 수행되지 않는다. 부분 배열이 합쳐지는 merge 함수에서 비교 연산과 이동 연산이 수행되는 것이다. 순환 호출의 깊이만큼의 합병 단계가 필요하다.

하나의 합병단계에서는 최대 n번의 비교 연산이 필요하다. 그러한 합병 단계가 $k = \log_2 n$번 만큼 있으므로 총 비교 연산은 최대 $n\log_2 n$번 필요하다.

이동 연산의 경우 하나의 합병 단계에서 2n개가 필요하다. 따라서 $\log_2 n$개의 합병 단계가 필요하므로 총 $2n\log_2 n$개의 이동 연산이 필요하다.

결론적으로 합병정렬은 비교 연산과 이동 연산의 경우 $O(n\log_2 n)$의 복잡도를 가지는 알고리즘이다.

합병 정렬의 다른 장점은 안정적인 정렬 방법이며 데이터의 분포에 영향을 덜 받는다. 즉 입력 데이터가 무엇이든간에 정렬되는 시간은 동일하다.

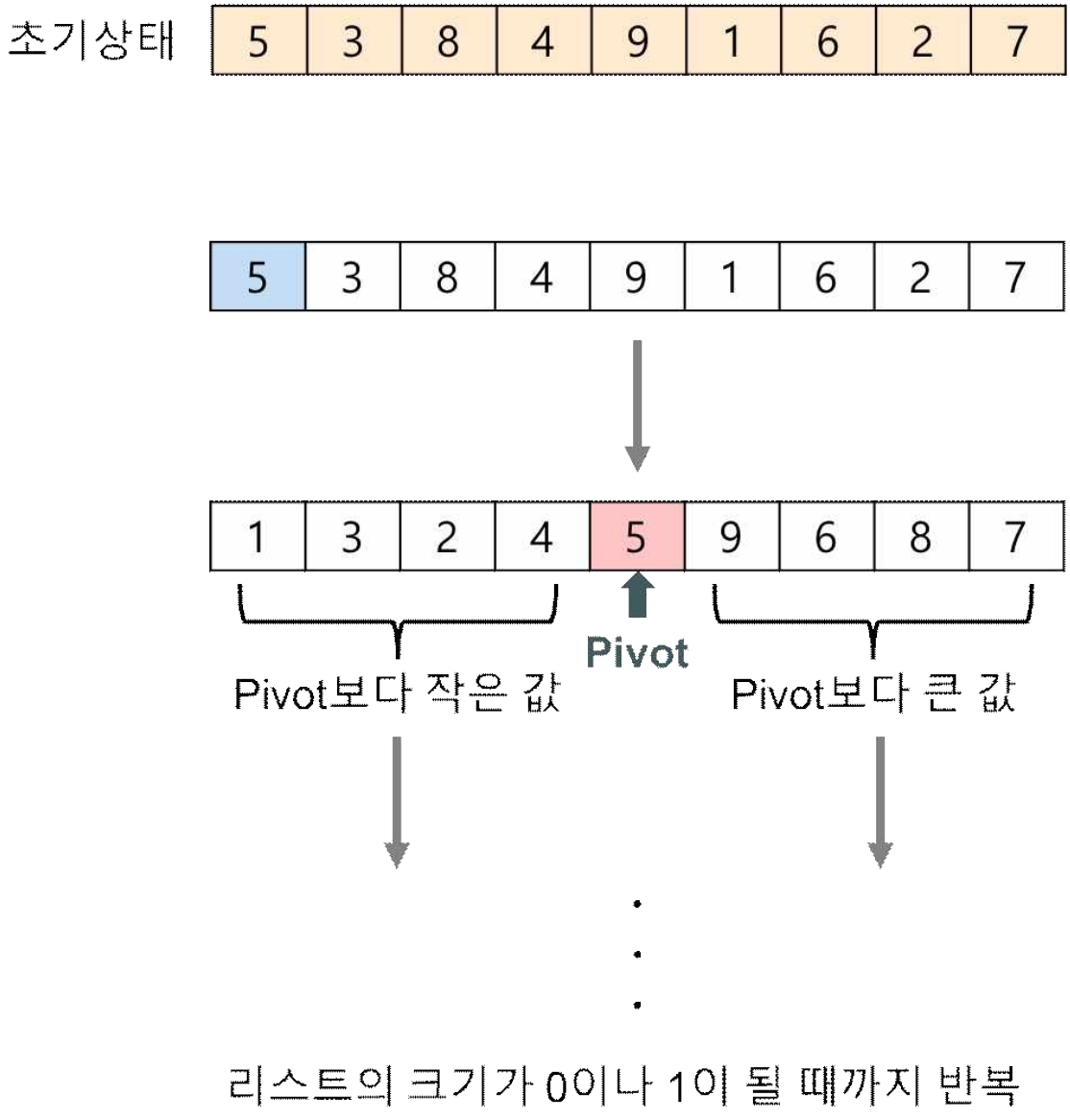
즉 최악, 평균, 최선의 경우가 다같이 $O(n\log_2 n)$인 정렬 방법이다.

합병 정렬의 단점은 임시 배열이 필요하다는 것과 만약 레코드들의 크기가 큰 경우에는 이동 횟수가 많아지므로 합병 정렬은 매우 큰 시간적 낭비를 초래할 수 있다. 그러나 만약 레코드를 연결리스트로 구성하여 합병 정렬할 경우, 링크 인덱스만 변경되므로 데이터의 이동은 무시할 수 있을 정도로 작아진다. 따라서 크기가 큰 레코드를 정렬할 경우, 만약 연결 리스트를 사용한다면, 합병 정렬은 퀵 정렬을 포함한 다른 어떤 정렬 방법보다 효율적일 수

있다.

▶ 퀵 정렬 (quick sort)

퀵 정렬은 평균적으로 매우 빠른 수행 속도를 자랑하는 정렬 방법이다. 퀵정렬도 분할-정복 법에 근거한다.
합병 정렬과는 달리 퀵정렬은 피벗(pivot)을 사용하여 리스트를 비균등하게 분할한다.
결과적으로 피벗을 중심으로 왼쪽을 피벗보다 작은 요소들로 구성되고, 오른쪽은 피벗보다 큰 요소들로 구성된다.



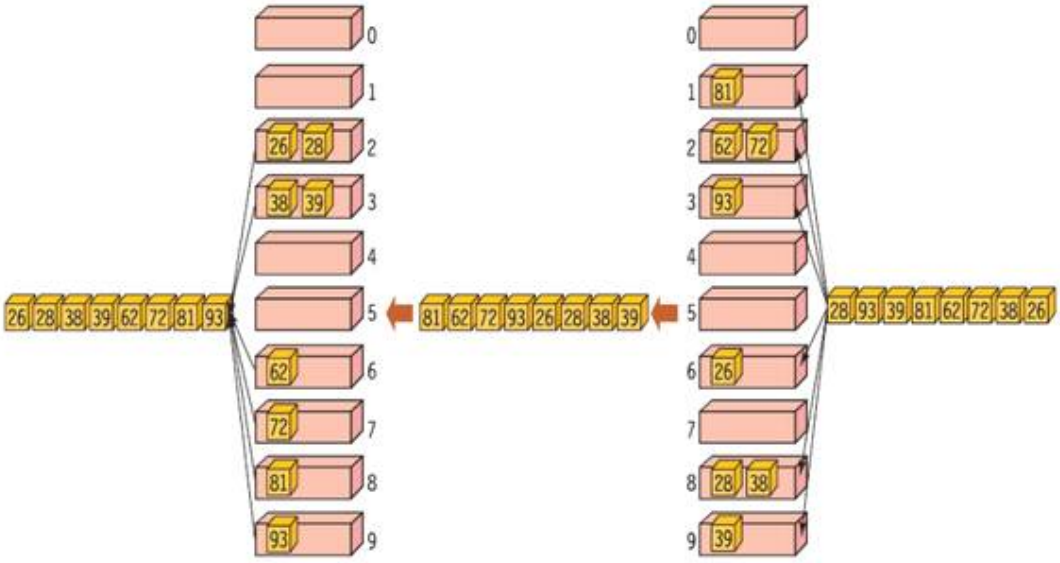
▶ 퀵 정렬의 복잡도 분석

퀵 정렬은 비교 연산을 총 $n\log_2 n$번 실행하게 되어 $O(n\log_2 n)$의 복잡도를 가지는 알고리즘이다. 레코드의 이동 횟수는 비교 횟수보다 적으므로 무시할 수 있다.

퀵 정렬은 최악의 경우, 이미 정렬된 리스트를 정렬하는 경우 $O(n^2)$의 시간 복잡도를 가지게 된다.
퀵정렬이 불필요한 데이터의 이동을 줄이고 먼 거리의 데이터를 교환할 뿐만 아니라, 한번 결정된 피벗들이 추후 연산에서 제외되는 특성 등에 기인한다고 보인다.
퀵 정렬은 속도가 빠르고 추가 메모리 공간을 필요로 하지 않는 등의 장점이 있는 반면에 정렬된 리스트에 대해서는 오히려 수행시간이 더 많이 걸리는 등의 단점도 가진다. 이러한 불균형 분할을 방지하기 위하여 피벗을 선택할 때 단순히 리스트의 왼쪽 데이터를 사용하는 대신에 보다 리스트의 중앙 부분을 분할할 수 있는 데이터를 선택한다. 많이 사용되는 방법은 리스트 내의 몇 개의 데이터 중에서 중간값(median)을 피벗으로 선택하는 것이다.
리스트의 왼쪽, 오른쪽, 중간의 3개의 데이터 중에서 중간 값을 선택하는 방법 (median of three)이 많이 사용 된다.

▶ 기수 정렬 radix sort

기수 정렬은 레코드를 비교하지 않고도 정렬하는 방법이다. 기수 정렬은 입력 데이터에 대해서 어떤 비교 연산도 실행하지 않고 데이터를 정렬할 수 있는 색다른 정렬 기법이다. 기수 정렬은 O(kn)의 시간 복잡도를 가지는데 대부분 k<4이하이다. 다만 문제는 기수 정렬이 추가적인 메모리를 필요로 한다는 것이다.
기수 정렬의 단점은 정렬할 수 있는 레코드의 타입이 한정된다는 점이다. 즉 기수 정렬을 사용하려면 레코드들의 키들이 동일한 길이를 가지는 숫자나 문자열로 구성되어 있어야 한다.
기수란 숫자의 자리수이다. 기수 정렬은 다단계 정렬이다. 단계의 수는 데이터의 자리수의 개수와 일치한다.
버킷(bucket)을 만들어서 입력 데이터를 각 자리수의 값에 따라 넣는다. 각 왼쪽상자부터 순차적으로 버킷 안에 들어 있는 숫자를 순차적으로 읽는다.
2자리수 이상의 정수를 정렬 할 때는 낮은 자리수로 정렬한 다음 차츰 높은 자리수로 정렬해야 한다.



▶ 기수 정렬의 분석

입력 리스트가 n개의 정수를 가지고 있다면 알고리즘 내부 루프는 n번 반복될 것이다. 만약 각 정수가 d개의 자리수를 가지고 있다면 외부 루프는 d번 반복된다. 따라서 기수 정렬은 O(d * n)의 시간 복잡도를 가진다. 일반적으로 d는 n에 비하여 아주 작은 수가 되므로 기수 정렬은 O(n)이라고 하여도 무리가 없다.
따라서 기수 정렬은 다른 정렬 방법에 비하여 비교적 빠른 수행 시간안에 정렬을 마칠 수 있다. 그러나 문제점은 기수 정렬은 정렬에 사용되는 킷값이 숫자로 표현되어야만이 적용이 가능하다.

▶ 정렬 알고리즘의 비교

정렬 종류	시간 복잡도			공간 복잡도
	평균(Average)	최선(Best)	최악(Worst)	
선택 정렬	O(n2)	O(n2)	O(n2)	O(n2)
버블 정렬	O(n2)	O(n2)	O(n2)	O(n)
삽입 정렬	O(n2)	O(n)	O(n2)	O(n2)
합병 정렬	O(n×log n)	O(n×log n)	O(n×log n)	O(n×log n)
퀵 정렬	O(n×log n)	O(n×log n)	O(n2)	O(n×log n)
힙 정렬	O(n×log n)	O(n×log n)	O(n×log n)	O(n×log n)
쉘 정렬	O(N^1.25)	O(N^1.25)	O(N^1.25)	O(n)
기수 정렬	O(dn)	O(dn)	O(dn)	

(히프 정렬 큐를 사용하여 완전 이진 트리로 구현하는 방법으로 윗 내용에서 제외 했습니다.)

▶ 히프 정렬 heap sort

히프는 우선순위 큐를 완전 이진 트리로 구현하는 방법으로 히프는 최대값이나 최소값을 쉽게 추출할 수 있는 자료 구조이다. 히프에는 최소 히프와 최대 히프가 있다.
최소 히프는 부모 노드의 값이 자식 노드의 값보다 작다.
최소 히프의 이러한 특성을 이용하여 정렬할 배열을 먼저 최소 히프로 변환한 다음 가장 작은 원소부터 차례대로 추출하여 정렬하는 방법을 히프 정렬이라 한다.