정렬

정렬

- 정렬(sorting)
  - 자료 모음에 순서를 부여하는 작업
  - 예
    - 9, 1, 3, 7, 3, 4, 2, 1, 5 → 1, 1, 2, 3, 3, 4, 5, 7, 9

정렬 알고리즘

- 정렬 알고리즘
  - 비교 기반 정렬 (comparison sorting)
    - 거품 정렬 (bubble sort)
    - 선택 정렬 (selection sort)
    - 삽입 정렬 (insertion sort)
    - 퀵 정렬 (quick sort)
    - 합병 정렬 (merge sort)
    - 힙 정렬 (heap sort)
  - 정수 정렬 (integer sorting)
    - 계수 정렬 (counting sort)
    - 기수 정렬 (radix sort)

정렬알고리즘 시간복잡도

- 비교 기반 정렬 알고리즘 시간복잡도

정렬 알고리즘	시간복잡도	
	평균	최악
퀵정렬 (quick sort)	O(n log n)	$O(n^2)$
합병정렬 (merge sort)	O(n log n)	O(n log n)
힙정렬 (heap sort)	O(n log n)	O(n log n)
삽입정렬 (insertion sort)	$O(n^2)$	$O(n^2)$
선택정렬 (selection sort)	$O(n^2)$	$O(n^2)$
버블정렬 (bubble sort)	$O(n^2)$	$O(n^2)$

거품정렬


```
public class Test {
    public static void main(String[] args) {
        int   v[]={5,8,1,9,3,5,1,5};
        bubbleSort(v);
        System.out.println(Arrays.toString(v));
    }
    private static void bubbleSort(int[] v) {
        for (int i = 0; i < v.length; i++) {
            for (int j = 1; j < v.length; j++) {
                if(v[j-1]>v[j]) {
                    int   temp=v[j];
                    v[j]=v[j-1];
                    v[j-1]=temp;
                }
            }
        }
    }
}
```

거품 정렬: 리스트 전체에 대해 인접 원소들을 비교하여 순서가 맞지 않는 경우 교환하는 작업을 반복하는 작업을 리스트 전체가 정렬될 때까지 반복함으로써 정렬을 수행한다.

실습: i번째 반복인 경우 이미 i개 값들의 최종 위치가 결정되었으므로 매번 전체 자료에 대한 검사는 비효율적.
i=1인 경우 이미 가장 큰 1개 자료 값 9는 최종 위치가 결정되었으로, [5,1,8,3,5,1,5]에 대해서만 내부 반복문 수행하면 됨.

5	1	8	3	5	1	5	9

실습: 특정 내부 반복문 수행 시 교환이 전혀 발생하지 않았다면 자료는 이미 정렬된 상태이므로 정렬 종료하면 됨

1	1	3	5	5	5	8	9

거품정렬

					오름차순 정렬 가정
5	4	3	2	1	최초 상태
5	4	3	2	1	5,4 비교 후 교환
4	5	3	2	1	5,3 비교
4	3	5	2	1	5,2 비교 후 교환
4	3	2	5	1	5,1 비교 후 교환
4	3	2	1	**5**	n=5인 경우, 총 4번 비교 후 최대값(5) 위치 결정
3	2	1	**4**	**5**	총 3회 비교 후 두번째 큰 값(4) 위치 결정
2	1	**3**	**4**	**5**	총 2회 비교 후 세번째 큰 값(3) 위치 결정
1	**2**	**3**	**4**	**5**	총 1회 비교 후 네번째 큰 값(2) 위치 결정

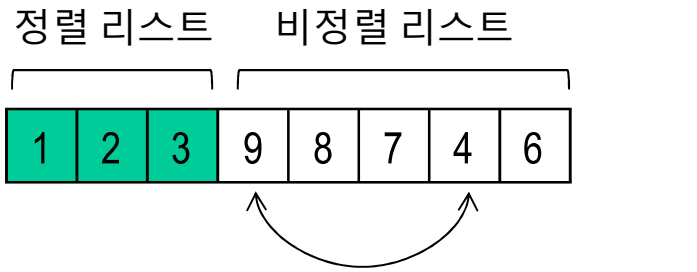
총 비교 횟수(n=5): 4+3+2+1
총 비교 횟수(n=1000): 999+998+ … +2+1

$$1 + 2 + \cdots (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

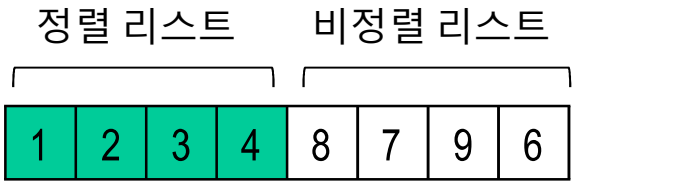
선택정렬


```
public class Test {
    public static void main(String[] args) {
        int   v[]={1,2,3,9,8,7,4,6};
        selectionSort(v);
        System.out.println(Arrays.toString(v));
    }
    private static void selectionSort(int[] v) {
        for (int i = 0; i < v.length-1; i++) {
            int   minPos=i;
            for (int j = i+1; j < v.length; j++) {
                if(v[minPos]>v[j]) minPos=j;
            }
            int   x=v[i];
            v[i]=v[minPos];
            v[minPos]=x;
        }
    }
}
```

선택 정렬: 비정렬 리스트 내 최소값을 찾아 비정렬리스트의 첫 위치 자료와 교환하는작업을 반복함으로써 정렬을 수행한다.



비정렬리스트 내 최소값 4를
비정렬리스트의 첫 위치 자료 9와 교환



선택정렬

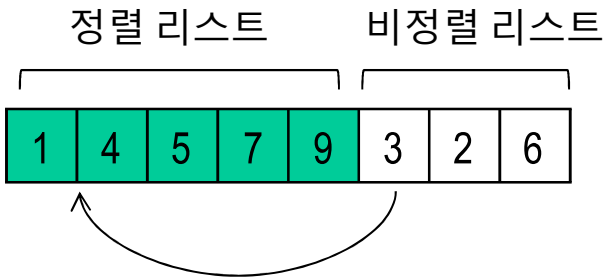
					오름차순 정렬 가정
5	4	3	2	1	최초 상태, 비정렬 목록 [5 4 3 2 1]
5	4	3	2	**1**	비정렬 목록 [5 4 3 2 1]에서 최소값 1 결정
1	4	3	2	5	비정렬 목록의 첫 위치 자료 5와 교환, 비정렬 목록 [4 3 2 5]
1	4	3	**2**	5	비정렬 목록 [4 3 2 5]에서 최소값 2 결정
1	**2**	3	4	5	비정렬 목록의 첫 위치 자료 4와 교환, 비정렬 목록 [3 4 5]
1	**2**	**3**	4	5	비정렬 목록 [3 4 5]에서 최소값 3 결정
1	**2**	**3**	4	5	비정렬 목록의 첫 위치 자료 3과 교환, 비정렬 목록 [4 5]
1	**2**	**3**	**4**	5	비정렬 목록 [4 5]에서 최소값 4 결정
1	**2**	**3**	**4**	5	비정렬 목록의 첫 위치 자료 4와 교환, 비정렬 목록 [5]

$$1 + 2 + \cdots (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

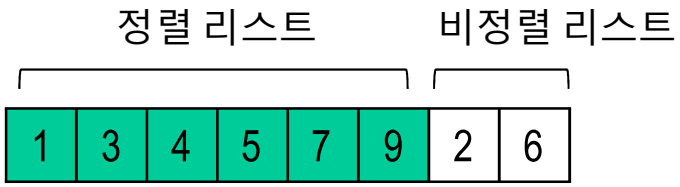
삽입정렬


```
public class Test {
	public static void main(String[] args) {
		int	v[]={5,8,1,9,3,5,1,5};
		insertionSort(v);
		System.out.println(Arrays.toString(v));
	}
	private static void insertionSort(int[] v) {
		for (int i = 1; i < v.length; i++) {
			for (int j = i; j>0 && v[j-1]>v[j]; j--) {
				int	x=v[j];
				v[j]=v[j-1];
				v[j-1]=x;
			}
		}
	}
}
```

삽입 정렬: 비정렬 리스트에서 추출된 자료를 정렬된 리스트 내 올바른 위치에 삽입하는 과정을 반복함으로써 정렬을 수행한다.



비정렬 리스트 내 첫 위치 자료 3을
정렬리스트 내 올바른 위치에 삽입



삽입정렬

https://en.wikipedia.org/wiki/Insertion_sort, CC-BY-SA

					오름차순 정렬 가정
5	4	3	2	1	최초 상태, 정렬 목록 [5], 비정렬 목록 [4 3 2 1]
5	**4**	3	2	1	비정렬 목록 [4 3 2 1]의 첫 자료 4를 정렬 목록 [5]에 삽입
4	**5**	3	2	1	삽입 후 비정렬 목록 [3 2 1]
4	**5**	**3**	2	1	비정렬 목록 [3 2 1]의 첫 자료 3을 정렬 목록 [4 5]에 삽입
3	**4**	**5**	2	1	삽입 후 비정렬 목록 [2 1]
3	**4**	**5**	**2**	1	비정렬 목록 [2 1]의 첫 자료 2를 정렬 목록 [3 4 5]에 삽입
2	**3**	**4**	**5**	1	삽입 후 비정렬 목록 [1]
2	**3**	**4**	**5**	**1**	비정렬 목록 [1]의 첫 자료 1을 정렬 목록 [2 3 4 5]에 삽입
1	**2**	**3**	**4**	**5**	삽입 후 비정렬 목록 []

$$1 + 2 + \cdots (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

선택정렬 vs. 삽입정렬

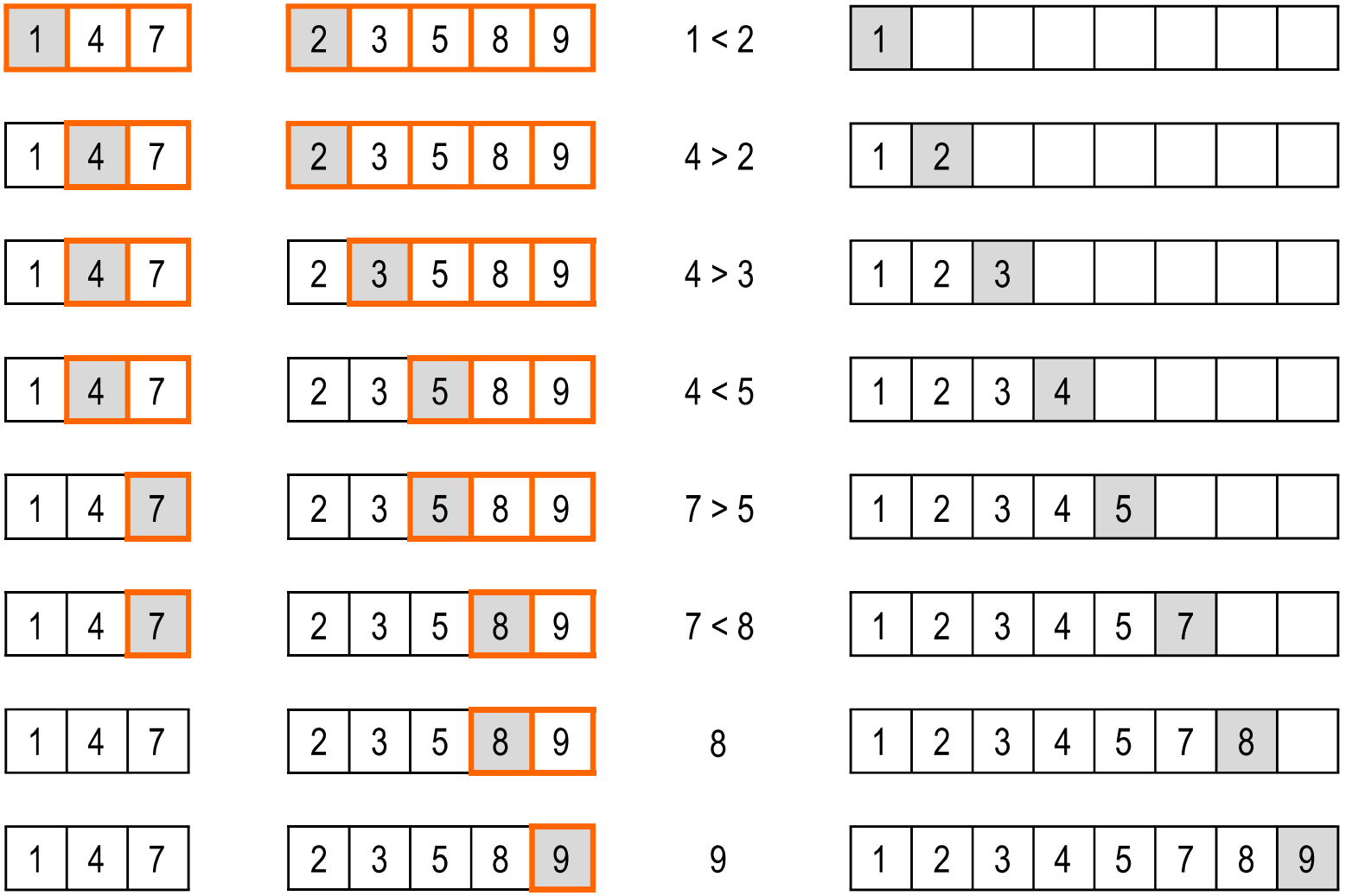
- 선택정렬
  - 매 단계마다 최소값 탐색 위해 **비정렬 목록 전체 검사**(read) 필요
  - 매 단계 최소값의 위치는 1회 교환(write)으로 가능
    - Write 시간이 read보다 월등히 큰 경우 선택정렬이 삽입정렬보다 유리
- 삽입정렬
  - 정렬된 목록 내 임의 자료의 삽입 위치 결정 위해 **평균적으로 정렬 목록의 절반 탐색**(read) 및 자리 이동(write) 필요
    - 일반적으로 삽입정렬이 선택정렬보다 효율적
  - 거의 정렬된 자료 모음의 경우 삽입 시간 최소화 가능

선택정렬 vs. 삽입정렬

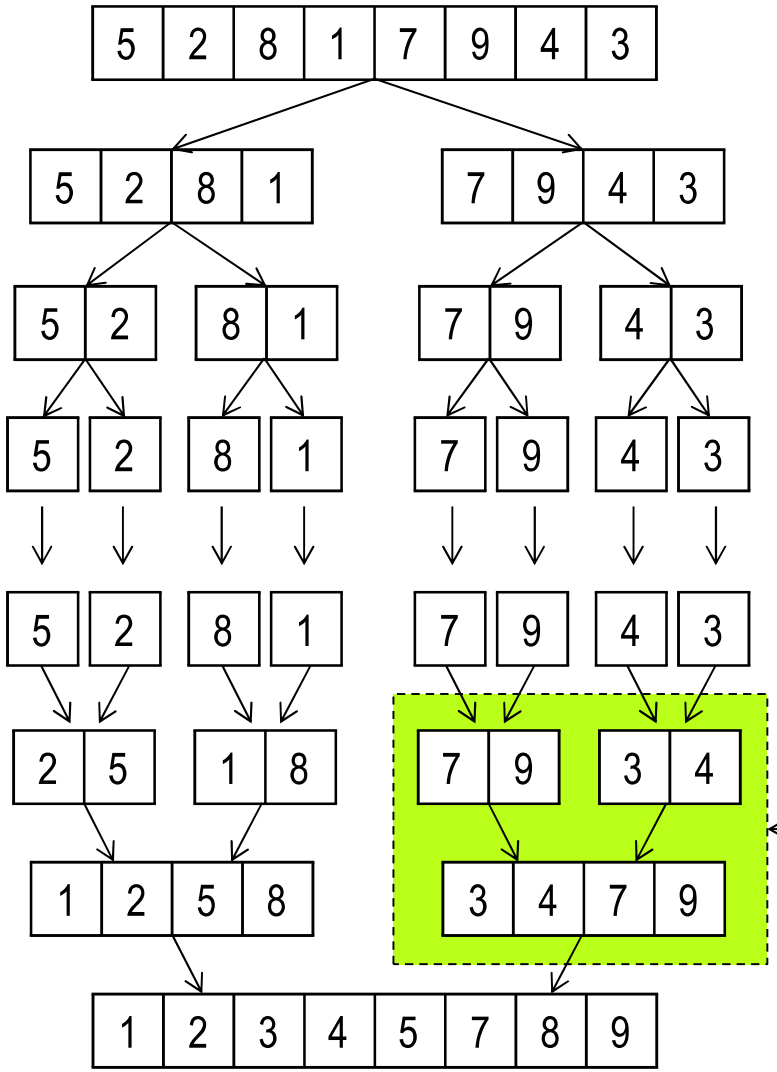
					정렬된 자료 모음에 대한 선택정렬
1	2	3	4	5	초기 상태, 비정렬 목록 [1 2 3 4 5]
1	2	3	4	5	최소값 1 탐색 위해 비정렬 목록 전체 [1 2 3 4 5] 검사
1	**2**	3	4	5	최소값 2 탐색 위해 비정렬 목록 전체 [2 3 4 5] 검사
1	**2**	**3**	4	5	최소값 3 탐색 위해 비정렬 목록 전체 [3 4 5] 검사
1	**2**	**3**	**4**	5	최소값 4 탐색 위해 비정렬 목록 전체 [4 5] 검사

					정렬된 자료 모음에 대한 삽입정렬
1	2	3	4	5	초기 상태, 정렬 목록 [1], 비정렬 목록 [2 3 4 5]
1	2	3	4	5	정렬된 자료의 경우 자료 2 삽입 시 정렬 목록 내 위치 이동 불필요
1	**2**	3	4	5	정렬된 자료의 경우 자료 3 삽입 시 정렬 목록 내 위치 이동 불필요
1	**2**	**3**	4	5	정렬된 자료의 경우 자료 4 삽입 시 정렬 목록 내 위치 이동 불필요
1	**2**	**3**	**4**	5	정렬된 자료의 경우 자료 5 삽입 시 정렬 목록 내 위치 이동 불필요

정렬된 두 배열 합병


1 4 7
2 3 5 8 9
1 < 2
1
1 4 7
2 3 5 8 9
4 > 2
1 2
1 4 7
2 3 5 8 9
4 > 3
1 2 3
1 4 7
2 3 5 8 9
4 < 5
1 2 3 4
1 4 7
2 3 5 8 9
7 > 5
1 2 3 4 5
1 4 7
2 3 5 8 9
7 < 8
1 2 3 4 5 7
1 4 7
2 3 5 8 9
8
1 2 3 4 5 7 8
1 4 7
2 3 5 8 9
9
1 2 3 4 5 7 8 9

합병정렬




```
public class Test {
    public static void main(String[] args) {
        int    v[]={5,2,8,1,7,9,4,3};
        mergeSort(v, 0, v.length-1);
        System.out.println(Arrays.toString(v));
    }
    private static void mergeSort(int[] v, int left, int right) {
        if(left==right) return;
        int    m=(left+right)/2;
        mergeSort(v, left, m);
        mergeSort(v, m+1, right);
        mergeArray(v, left, m, m+1, right);
    }
    private static void mergeArray(int[] v, int l1, int r1, int l2, int r2) {
    }
}
```

정렬된 두 배열을 하나로 합병

합병정렬

								오름차순 정렬 가정
5	2	8	1	7	9	4	3	mergeSort(v,0,7)
5	2	8	1					mergeSort(v,0,3)
5	2							mergeSort(v,0,1)
5								mergeSort(v,0,0) & return
	2							mergeSort(v,1,1) & return
2	5							mergeArray(v,0,0,1,1)
		8	1					mergeSort(v,2,3)
		8						mergeSort(v,2,2) & return
			1					mergeSort(v,3,3) & return
		1	8					mergeArray(v,2,2,3,3)
1	2	5	8					mergeArray(v,0,1,2,3)
				3	4	7	9	mergeSort(v,4,7)
1	2	3	4	5	7	8	9	mergeArray(v,0,3,4,7)

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 프로그래밍 콘테스트 챌린징, Akiba 등 공저, 로드북, 2011.
- https://introcs.cs.princeton.edu/
- Introduction to Algorithms, Cormen et al., 3rd Edition (The MIT Press)