CSE2010 자료구조론

Week 12: Sorting 1

ICT융합학부 한진영

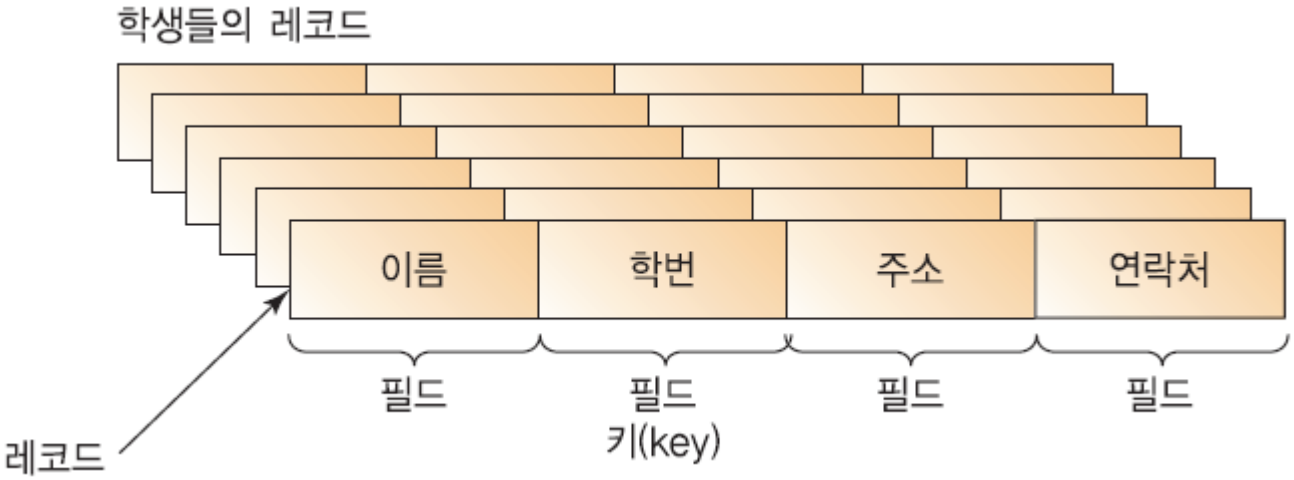
정렬(sorting)이란?

- 크기 순으로 오름차순이나 내림차순으로 나열하는 것
  - 정렬은 컴퓨터공학을 포함한 모든 과학기술분야에서 가장 기본적이고 중요한 알고리즘 중 하나

- 정렬은 자료 탐색에 있어서 필수적
  - (예) 만약 영어사전에서 단어들이 알파벳 순으로 정렬되어 있지 않다면?

정렬의 대상

- 일반적으로 정렬시켜야 될 대상은 레코드(record)
  - 레코드는 필드(field)라는 보다 작은 단위로 구성
  - 키(key) 필드: 레코드와 레코드를 구별



정렬 알고리즘(1)

- 모든 경우에 최적인 정렬 알고리즘은 없음

- 각 응용 분야에 적합한 정렬 방법 사용해야 함
  - 레코드 수의 많고 적음
  - 레코드 크기의 크고 작음
  - Key의 특성(문자, 정수, 실수 등)
  - 메모리 내부/외부 정렬

- 정렬 알고리즘의 평가 기준
  - 비교 횟수의 많고 적음
  - 이동 횟수의 많고 적음

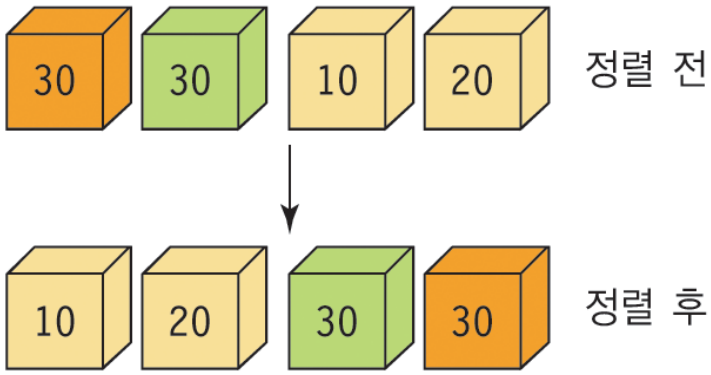
정렬 알고리즘(2)

- 단순하지만 비효율적인 방법: 삽입 정렬, 선택 정렬, 버블 정렬 등
- 복잡하지만 효율적인 방법: 퀵 정렬, 히프(heap) 정렬, 합병 정렬, 기수 정렬 등

- 내부 정렬(internal sorting): 모든 데이터가 주기억장치에 저장된 상태에서 정렬
- 외부 정렬(external sorting): 외부 기억장치에 대부분의 데이터가 있고 일부만 주기억장치에 저장된 상태에서 정렬

정렬 알고리즘의 안정성

- 정렬 알고리즘의 안정성(stability)
  - 입력 데이터에 동일한 키 값을 갖는 레코드가 여러 개일 경우, 동일한 키 값을 갖는 레코드들의 상대적인 위치가 정렬 후에도 바뀌지 않음
  - 안정성이 낮은 정렬의 예



정렬의 안정성이 중요한 경우,
안정성을 추구하는 삽입 정렬,
합병 정렬 등을 사용!

선택정렬(Selection Sort)

- 선택 정렬의 원리
  - 정렬된 왼쪽 리스트와 정렬 안된 오른쪽 리스트 가정
  - 초기에는 왼쪽 리스트는 비어 있고, 정렬할 숫자들은 모두 오른쪽 리스트에 존재
  - 오른쪽 리스트에서 최소값 선택하여 오른쪽 리스트의 첫번째 수와 교환
    - 왼쪽 리스트 크기 1 증가
    - 오른쪽 리스트 크기 1 감소
  - 오른쪽 리스트가 소진되면 정렬 완료

왼쪽 리스트	오른쪽 리스트	설명
()	(5,3,8,1,2,7)	초기 상태
(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

선택정렬 알고리즘(1)

- 제자리 정렬(in-place sorting)
  - 입력 배열 이외에 추가적인 공간을 사용하지 않음
  - 입력 배열에서 최소값 발견 후, 이 최소값을 배열의 첫 번째 요소와 교환
  - 첫 번째 요소를 제외한 나머지 요소들 중에서 가장 작은 값 선택하고 이를 두 번째 요소와 교환
    - 이러한 과정이 반복됨

선택정렬 알고리즘(2)

```
selection_sort(A, n)

for i←0 to n-2 do // A[0]~A[n-2]까지 정렬되어 있으면, 이미 A[n-1]이 가장
                  // 큰 값이므로 n-1까지 정렬할 필요 없음
    least ← A[i], A[i+1],..., A[n-1] 중에서 가장 작은 값의 인덱스;
    A[i]와 A[least]의 교환;
    i++;
```

선택정렬 복잡도 분석

- 시간 복잡도: $O(n^2)$
  - 2개의 for 루프
- 안정성을 만족하지 않음
  - 값이 같은 레코드가 있을 경우, 상대적인 위치 변경 가능

삽입정렬(Insertion Sort)

- 손안의 카드를 정렬하는 방법과 유사
  - 새로운 카드를 기존의 정렬된 카드 사이의 올바른 자리를 찾아 삽입
- 삽입정렬
  - 정렬되어 있는 리스트에 새로운 레코드를 올바른 위치에 삽입하는 과정 반복

삽입정렬 과정


5 3 8 1 2 7
초기 상태
5 3 8 1 2 7
3을 삽입
3 5 8 1 2 7
8은 이미 제자리에
3 5 8 1 2 7
1을 삽입
1 3 5 8 2 7
2를 삽입
1 2 3 5 8 7
7을 삽입
1 2 3 5 7 8
정렬 완료

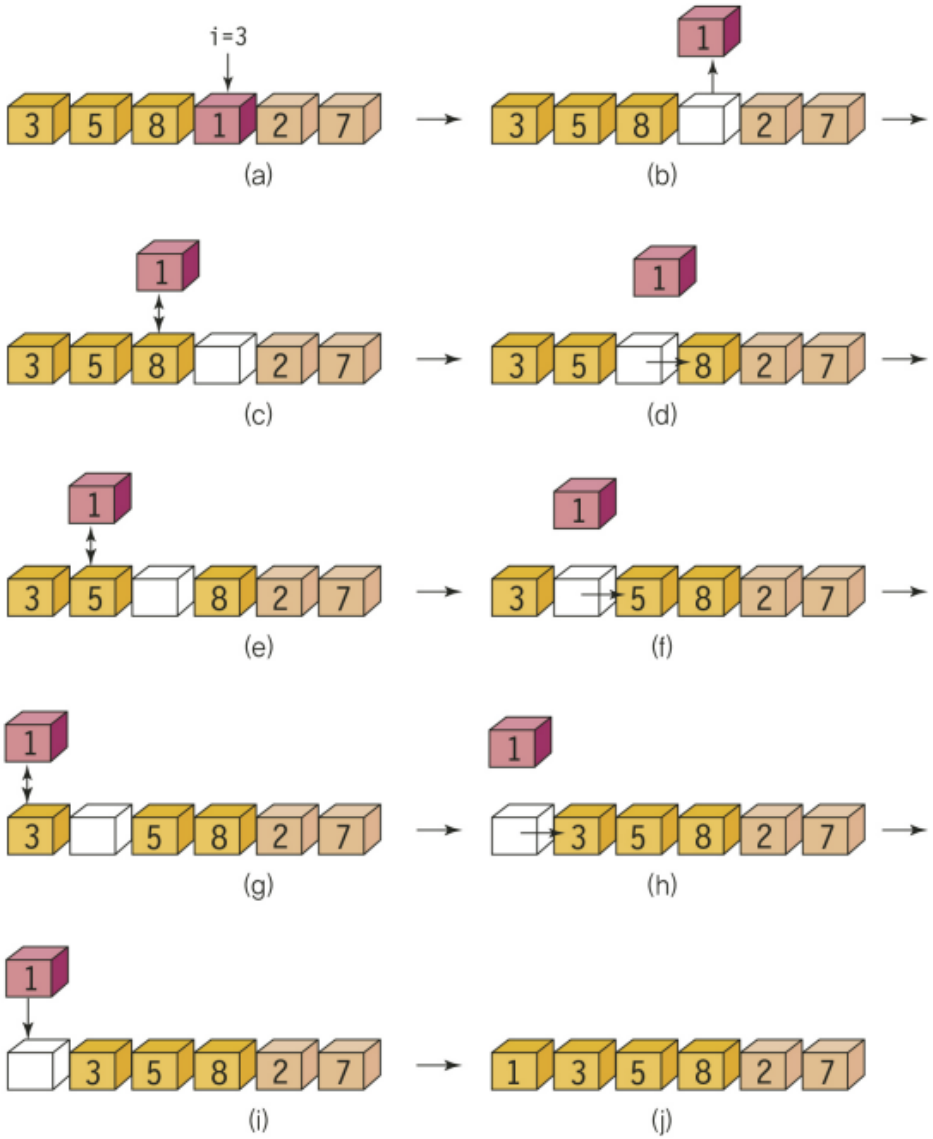
삽입정렬 알고리즘

```
insertion_sort(A, n)

1. for i ← 1 to n-1 do
2.     key ← A[i];
3.     j ← i-1;
4.     while j≥0 and A[j]>key do
5.           A[j+1] ← A[j];
6.           j ← j-1;
7.      A[j+1] ← key
```

- 1. 인덱스 1부터 시작
- 2. 현재 삽입될 숫자인 i번째 정수를 key 변수로 복사
- 3. 현재 정렬된 배열은 i-1까지 이므로, i-1번째부터 역순으로 조사
- 4. j값이 음수가 아니어야 되고 key값보다 정렬된 배열에 있는 값이 크면 수행
- 5. j번째를 j+1번째로 이동
- 6. j를 하나 감소
- 7. j번째 정수가 key보다 작으므로 j+1번째가 key값이 들어갈 위치

삽입정렬 알고리즘 수행 예



삽입정렬 복잡도 분석

- 최선의 경우 O(n): 이미 정렬되어 있는 경우
  - 비교: *n-1* 번
- 최악의 경우 $O(n^2)$: 역순으로 정렬되어 있는 경우
  - 모든 단계에서 앞에 놓인 자료 전부 이동
- 평균의 경우 $O(n^2)$

- 특징
  - 많은 이동 필요 -> 레코드가 클 경우 불리
  - 안정된 정렬 방법
  - 대부분 정렬되어 있으면 매우 효율적

Week 12: Sorting 1