CSE2010 자료구조론

Week 12: Sorting 2

ICT융합학부 한진영

버블정렬(Bubble Sort)

- 인접한 2개의 레코드를 비교하여 순서대로 되어 있지 않으면 서로 교환
- 이러한 비교-교환 과정을 리스트의 왼쪽 끝에서 오른쪽 끝까지 반복(스캔)

버블정렬 알고리즘

```
BubbleSort(A, n)

for i←n-1 to 1 do
   for j←0 to i-1 do //하나의 스캔 과정
         j와 j+1번째의 요소가 크기 순이 아니면 교환
         j++;
   i--;
```

버블정렬 복잡도

- 비교 횟수(최상, 평균, 최악의 경우 모두 일정)

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

- 이동 횟수
  - 역순으로 정렬된 경우(최악의 경우) :
    - 이동 횟수 = 3 * 비교 횟수
  - 이미 정렬된 경우(최선의 경우) :
    - 이동 횟수 = 0
  - 평균의 경우 : O(n^2)

- 레코드의 이동 과다
  - 이동 연산은 비교 연산 보다 더 많은 시간이 소요됨

셸정렬(Shell Sort)

- 삽입정렬이 어느 정도 정렬된 리스트에서 대단히 빠른 것에 착안
  - 삽입 정렬은 요소들이 이웃한 위치로만 이동하므로, 많은 이동에 의해서만 요소가 제자리를 찾아감
  - 요소들이 멀리 떨어진 위치로 이동할 수 있게 하면, 보다 적게 이동하여 제자리 찾을 수 있음
- 전체 리스트를 일정 간격의 부분 리스트로 나눔
  - 나뉘어진 각각의 부분 리스트를 삽입정렬 함
- 간격을 줄임
  - 부분 리스트의 수는 더 작아짐
  - 각 부분 리스트의 크기는 더 커짐
- 간격이 1이 될 때 까지 부분 리스트의 삽입 정렬 과정 반복
- 평균적인 경우의 시간복잡도 : $O(n^{1.5})$

셸정렬 예(1)

10	–	–	–	–	3	–	–	–	–	16
	8	–	–	–	–	22				
		6	–	–	–	–	1			
			20	–	–	–	–	0		
				4	–	–	–	–	15	

(a) 간격 5로 만들어진 부분 리스트

3	–	–	–	–	10	–	–	–	–	16
	8	–	–	–	–	22				
		1	–	–	–	–	6			
			0	–	–	–	–	20		
				4	–	–	–	–	15	

(b) 부분 리스트들이 정렬된 후의 리스트

셸정렬 예(2)

입력 배열	10	8	6	20	4	3	22	1	0	15	16
간격 5일때의 부분 리스트	10					3					16
		8					22				
			6					1			
				20					0		
					4					15	
부분 리스트 정렬 후	3					10					16
		8					22				
			1					6			
				0					20		
					4					15	
간격 5 정렬 후의 전체 배열	3	8	1	0	4	10	22	6	20	15	16
간격 3일때의 부분 리스트	3			0			22			15	
		8			4			6			16
			1			10			20		
부분 리스트 정렬 후	0			3			15			22	
		4			6			8			16
			1			10			20		
간격 3 정렬 후의 전체 배열	0	4	1	3	6	10	15	8	20	22	16
간격 1 정렬 후의 전체 배열	0	1	3	4	6	8	10	15	16	20	22

셸정렬 복잡도

- 셸 정렬의 장점
  - 불연속적인 부분 리스트에서 원거리 자료 이동으로 보다 적은 위치 교환으로 제자리 찾을 가능성 증대
  - 부분 리스트가 점진적으로 정렬된 상태가 되므로 삽입 정렬 속도 증가

- 시간적 복잡도
  - 최악의 경우 $O(n^2)$
  - 평균적인 경우 $O(n^{1.5})$

합병정렬(Merge Sort)

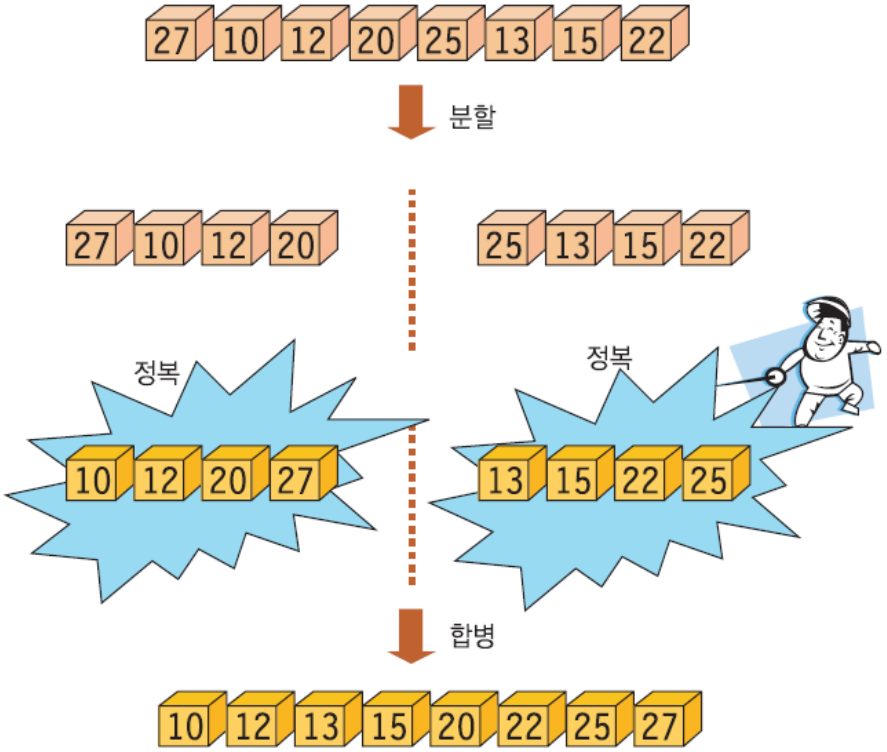
- 리스트를 두 개의 균등한 크기로 분할하고 분할된 부분리스트를 정렬
- 정렬된 두 개의 부분 리스트를 합하여 전체 리스트를 정렬함
- 분할 정복(divide and conquer) 방법 사용
  - 문제를 보다 작은 2개의 문제로 분리하고 각 문제를 해결한 다음,결과를 모아서 원래의 문제를 해결하는 전략
  - 분리된 문제가 아직도 해결하기 어렵다면(즉 충분히 작지 않다면) 분할정복방법을 다시 적용(재귀호출 이용)

1.분할(Divide): 배열을 같은 크기의 2개의 부분 배열로 분할
2.정복(Conquer): 부분배열을 정렬한다. 부분배열의 크기가 충분히 작지 않으면 재귀호출을 이용하여 다시 분할정복기법 적용
3.결합(Combine): 정렬된 부분배열을 하나의 배열에 통합

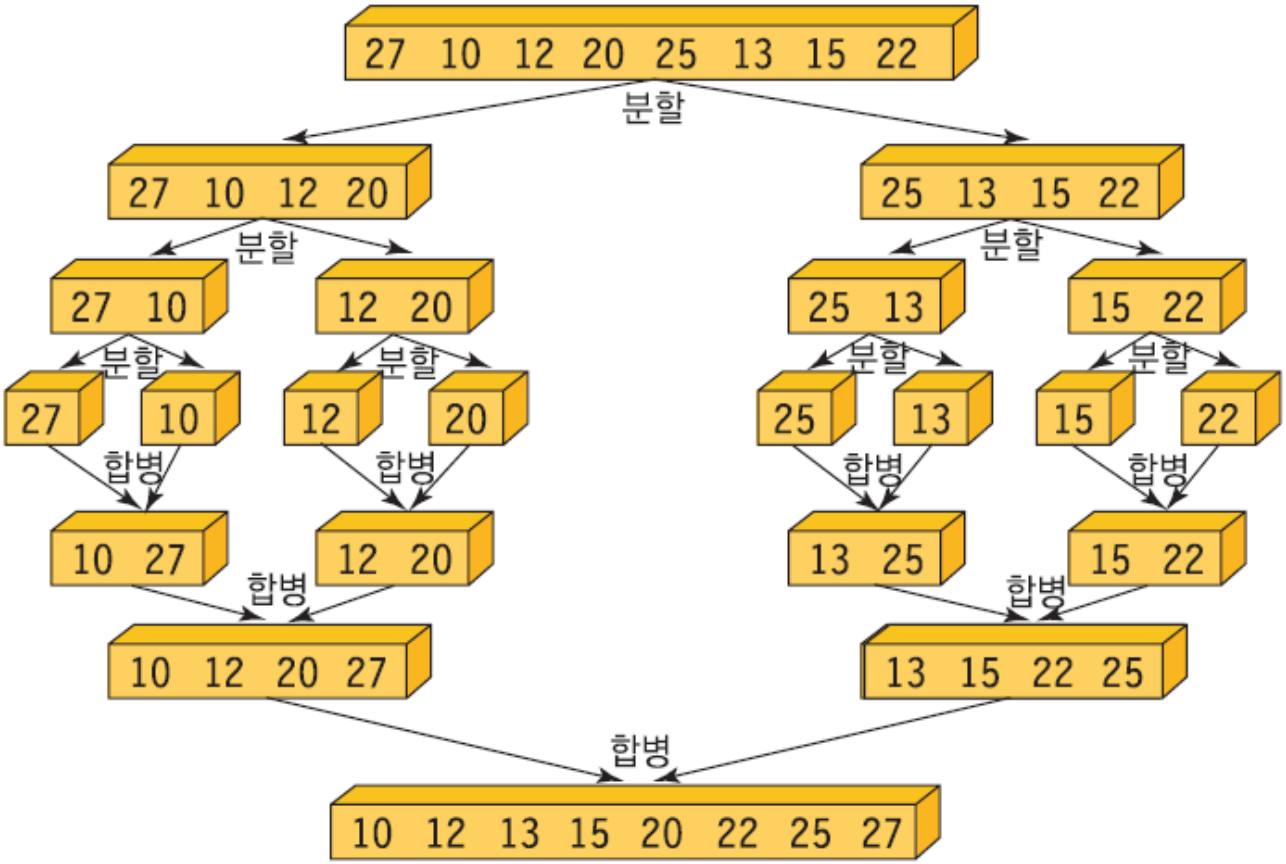
합병정렬 아이디어: D & C

입력: (27 10 12 20 25 13 15 22)

1.분할(Divide) : 전체 배열을 (27 10 12 20), (25 13 15 22) 2개 부분배열로 분리
2.정복(Conquer): 각 부분배열 정렬 (10 12 20 27), (13 15 22 25)
3.결합(Combine): 2개의 정렬된 부분배열 통합 (10 12 13 15 20 22 25 27)



합병정렬 과정


27 10 12 20 25 13 15 22
분할
27 10 12 20
25 13 15 22
분할
분할
27 10
12 20
25 13
15 22
분할
분할
분할
분할
27
10
12
20
25
13
15
22
합병
합병
합병
합병
10 27
12 20
13 25
15 22
합병
합병
10 12 20 27
13 15 22 25
합병
10 12 13 15 20 22 25 27

합병정렬 알고리즘

```
merge_sort(list, left, right)

1. if left < right
2.    mid = (left+right)/2;
3.    merge_sort(list, left, mid);
4.    merge_sort(list, mid+1, right);
5.    merge(list, left, mid, right);
```

1. 만약 나누어진 구간의 크기가 1이상이면

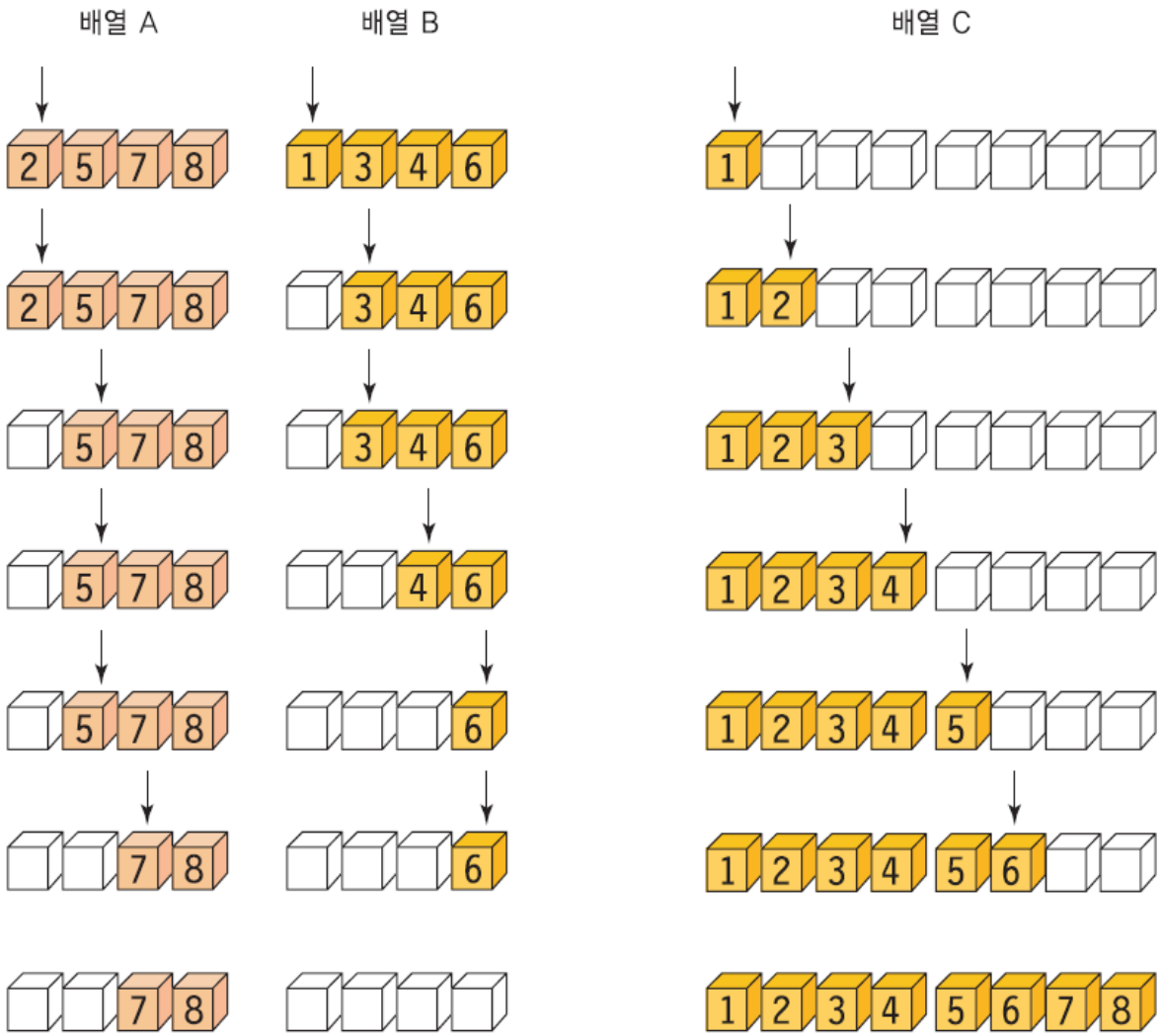
2. 중간 위치 계산

3. 왼쪽 부분 배열 정렬: merge_sort 함수 순환 호출

4. 오른쪽 부분 배열을 정렬: merge_sort 함수 순환 호출

5. 정렬된 2개의 부분 배열을 통합하여 하나의 정렬된 배열 만듦

합병과정


배열 A
배열 B
배열 C
2 5 7 8
1 3 4 6
1
2 5 7 8
3 4 6
1 2
5 7 8
3 4 6
1 2 3
5 7 8
4 6
1 2 3 4
5 7 8
6
1 2 3 4 5
7 8
6
1 2 3 4 5 6
7 8
1 2 3 4 5 6 7 8

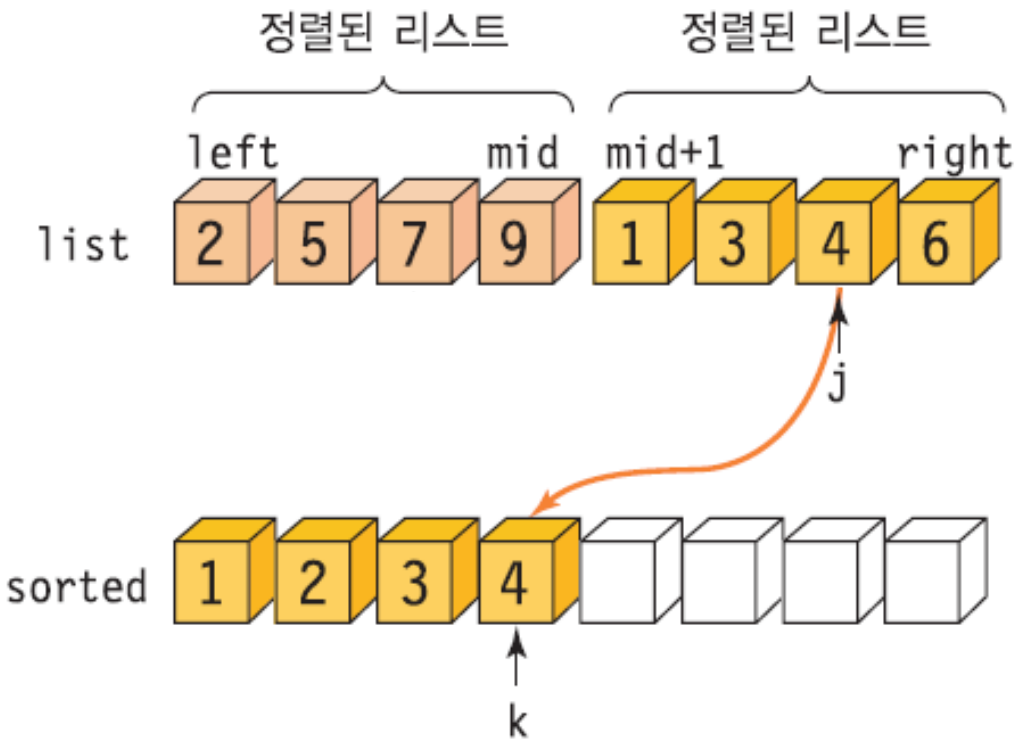
합병 알고리즘

```
merge(list, left, mid, right)
// 2개의 인접한 배열 list[left..mid]와 list[mid+1..right]를 합병
i←left;
j←mid+1;
k←left;
sorted 배열 생성; //합병된 리스트를 임시로 저장하기 위한 배열
while i≤left and j≤right do
      if(list[i]<list[j])
        then
          sorted[k]←list[i];
          k++;
          i++;
        else
          sorted[k]←list[j];
          k++;
          j++;

요소가 남아있는 부분배열을 sorted로 복사한다;
sorted를 list로 복사한다;
```

합병 중간 상태

배열 list에 있는 두 개의 list를 배열 sorted에 합병하는 세부 과정



합병정렬 복잡도(1)

- 합병 정렬은 순환 호출 구조로 되어있음
- 레코드 개수 n이 2의 거듭제곱이라 가정하면($n=2^k$) 순환 호출의 깊이는 k가 됨, 여기서 $k=\log_2 n$
- Merge 함수에서 비교연산과 이동연산 수행
  - 순환 호출 깊이만큼 합병 연산 필요함
- 비교 연산
  - 크기 n인 리스트를 정확히 균등 분배하므로 log(n)개의 패스
  - 각 패스에서 리스트의 모든 레코드 n개를 비교하므로 n번의 비교 연산, 합병단계가 $k=\log_2 n$만큼 있으므로 총 비교연산은 $n\log_2 n$

합병정렬 복잡도(2)

- 이동 연산
  - 레코드의 이동이 각 패스에서 2n번 발생하므로 전체 레코드의 이동은 2n*log(n)번 발생
    - 하나의 합병 단계에서 임시 배열에 복사했다가 다시 가져와야 하기 때문에, 이동 연산은 총 부분 배열에 들어 있는 요소의 개수가 n인 경우, 총 2n번 발생함
    - 총 $\log_2 n$개의 합병 단계가 필요하므로 총 2n*log(n)번 이동 발생
  - 레코드의 크기가 큰 경우에는 매우 큰 시간적 낭비 초래
  - 레코드를 연결 리스트로 구성하여 합병 정렬할 경우,
    - 링크 인덱스만 변경되므로 데이터의 이동은 무시할 수 있을 정도로 작아짐
    - 따라서 크기가 큰 레코드를 정렬할 경우, 다른 어떤 정렬 방법보다 매우 효율적
- 최적, 평균, 최악의 경우 큰 차이 없이 O(n*log(n))의 복잡도
- 안정적이며 데이터의 초기 분포 정도에 영향을 덜 받음

Week 12: Sorting 2