15. 정렬

비교정렬

- 재배열 결정이 배열의 원소 쌍 간의 비교에 기반을 두고 있는 *비교 정렬(comparison sort)* 은 $\Theta(n \lg n)$이라는 시간 한계를 넘지 못한다.
- 많은 비교 정렬은 배열에 있는 원소 쌍을 서로 교환하고 있다. 이 연산을 위해 최적화된 swap() 메소드를 사용한다.

- 비교정렬 알고리즘
  - 버블정렬
  - 선택정렬
  - 삽입정렬
  - 합병정렬
  - 퀵정렬

버블정렬

- LISTING 15.1: The Bubble Sort

```
1 void sort(int[] a) {
2     for (int i = a.length-1; i > 0; i--)
3         for (int j = 0; j < i; j++)
4             if (a[j] > a[j+1]) swap(a, j, j+1);
5 }
```

실행시간 ; $\Theta(n^2)$

버블 정렬을 수행하라

Array = {66, 33, 99, 88, 44, 55, 22, 77}

선택 정렬

- LISTING 15.4: The Selection Sort

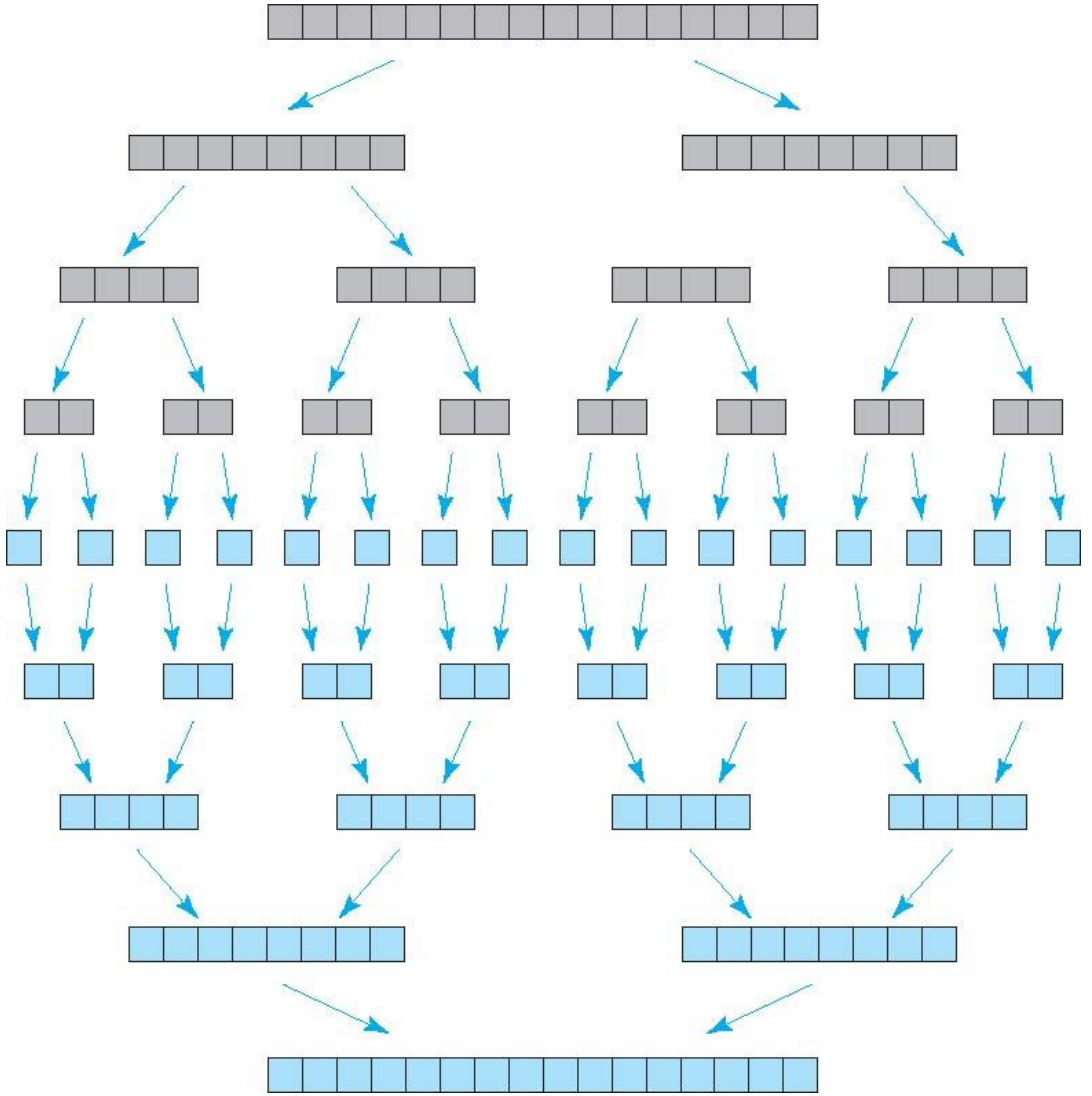
```
1 void sort(int[] a) {
2     for (int i = a.length-1; i > 0; i--) {
3         int m=0;
4         for (int j = 1; j <= i; j++)
5             if (a[j] > a[m]) m = j;
6         swap(a, i, m);
7     }
8 }
```

삽입 정렬

- LISTING 15.5: The Insertion Sort

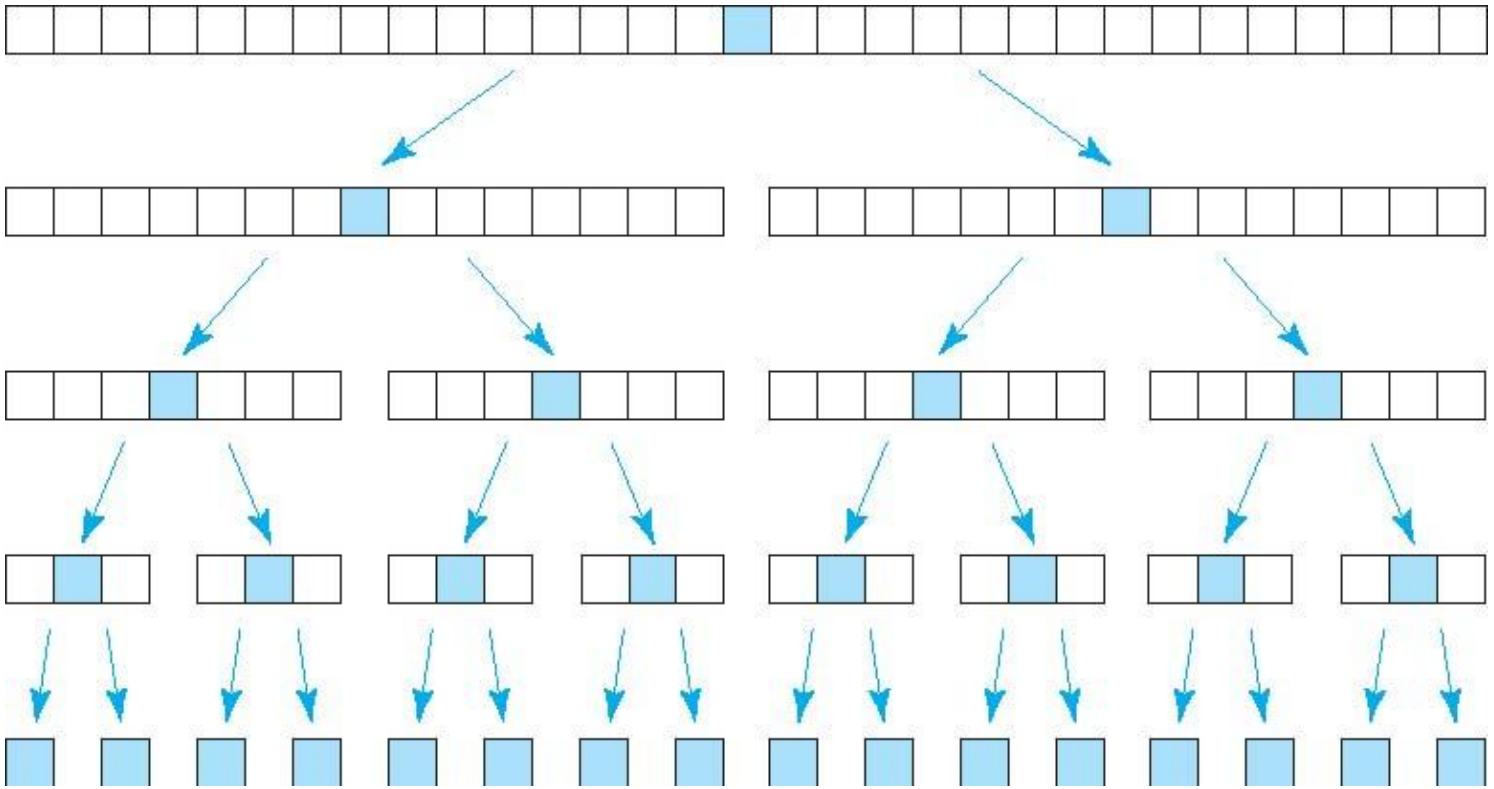
```
1 void sort(int[] a) {
2     for (int i = 1; i < a.length; i++) {
3       int ai = a[i], j = i;
4       for (j = i; j > 0 && a[j-1] > ai; j--)
5               a[j] = a[j-1];
6       a[j] = ai;
7     }
8 }
```

합병 정렬



$\Theta(n\lg n)$

최선의 경우 퀵 정렬



평균적인 경우, 최선의 경우: $\Theta(n\lg n)$

최악의 경우: $\Theta(n^2)$

15.8 히프 정렬

- 합병 정렬과 퀵 정렬의 $\Theta(n \lg n)$ 실행 시간은 선택 정렬이나 삽입 정렬의 $\Theta(n^2)$ 실행 시간에 비해 매우 좋은 것이다. 로그 시간은 두 알고리즘의 분할-정복 전략으로부터 나온다.
- 히프 정렬은 히프 자료 구조에 기반을 두고 있다. 이 자료 구조는 각각의 노드에 있는 키가 부모의 키보다 크기 않은 완전 이진 트리이다.
- 히프는 일반적으로 배열에 저장된다.
- 히프 정렬의 기본 아이디어는 선택 정렬과 동일하다. 즉, 아직 정렬되지 않은 세그먼트로부터 최대 원소를 선택하고 나서, 그것을 정확한 위치로 교환한다. 이러한 과정은 n-1 번 수행된다.
- 히프 정렬은 선형 시간이 아니라 로그 시간에 각각의 선택을 수행할 수 있도록 선택 정렬을 개선한 것이다.

히프 정렬 (2)

- 이 방법은 정렬되지 않은 세그먼트를 히프로 유지하고, 최대값이 a[0]에 위치하도록 한다. 여기서 각각의 선택이란 a[0]을 내보내고 나서, 나머지 정렬되지 않은 세그먼트 중에서 히프를 복원하는 것이다. 이러한 복원은 단순히 히프화 연산을 사용하면 되므로, $\Theta(\lg n)$ 단계가 걸리게 된다.

히프 정렬 알고리즘

1. 시이퀀스에 대해 히프 구축 알고리즘(알고리즘 14.2)를 적용.
2. i=n-1에서 1까지 내려가면서 단계 3-4를 반복.
3. a_i와 a_0를 교환.
4. 서브시이퀀스 $\{a_0,...a_{i-1}\}$를 히프화 (알고리즘 14.1).

- Listing 15.12: The Heap Sort

```
1 void sort(int[] a) {
2     for (int i = (a.length-1)/2; i >= 0; i--)
3         heapify(a, i, a.length);
4     for (int j = a.length-1; j > 0; j--) {
5         swap(a, 0, j);
6         heapify(a, 0, j);
7     }
8 }
```

히프구축 $\Theta(n)$

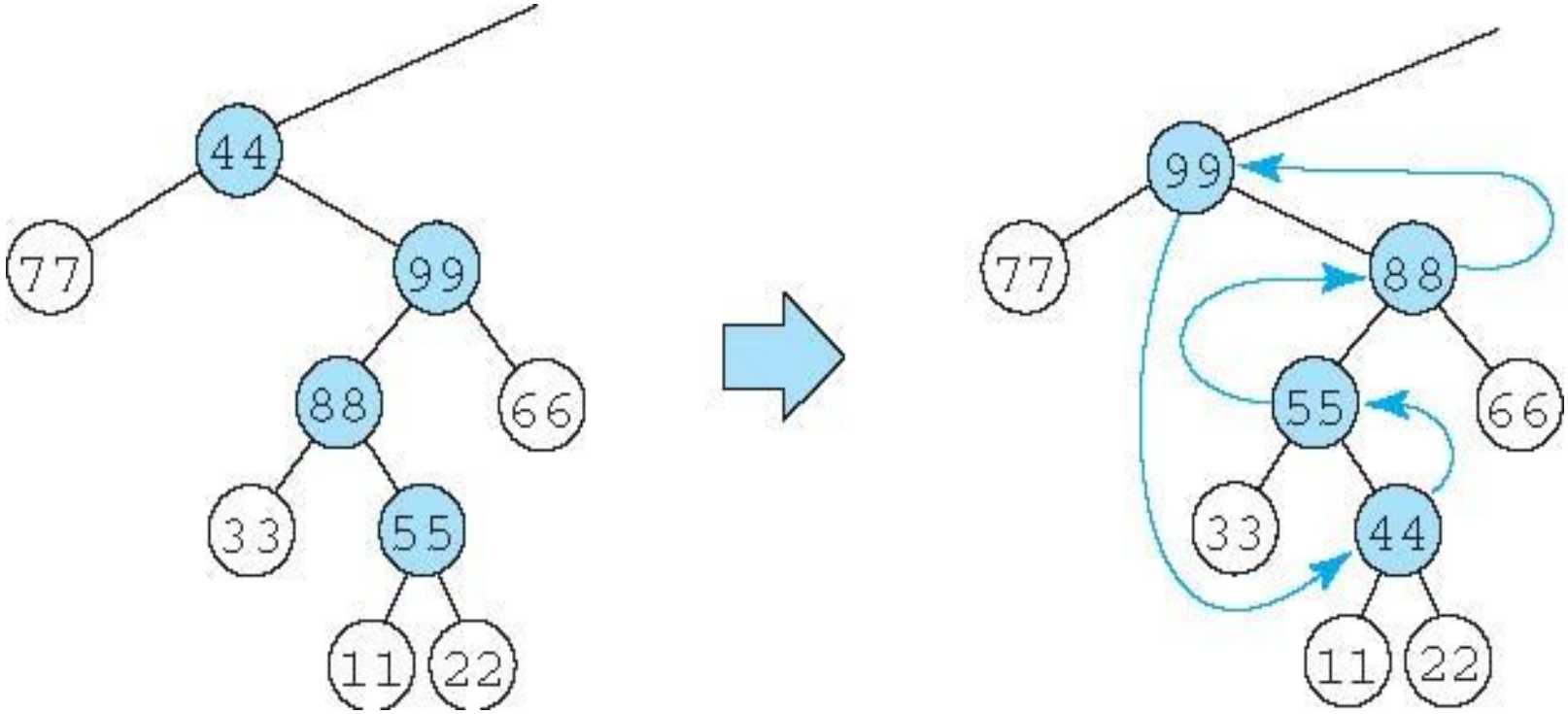
$\Theta(n \log n)$

14.2 히프화 연산 알고리즘 (복습)

- 히프화 연산 알고리즘
  - 입력 : 완전 이진 트리에 있는 노드 x.
  - 선조건 : x의 두 서브트리가 히프임.
  - 후조건 : x가 루트인 서브트리가 히프임.

  1. *temp*=*x*로 설정.
  2. *x*가 리프가 아닌 동안, 단계 3-4를 수행.
  3. *y*를 *x*의 큰 자식으로 설정.
  4. 만일 *y.key* > *x.key*이면 단계 5-6을 수행.
  5. *y*를 *x*로 복사.
  6. *x*=*y*로 설정.
  7. *temp*를 *x*로 복사.

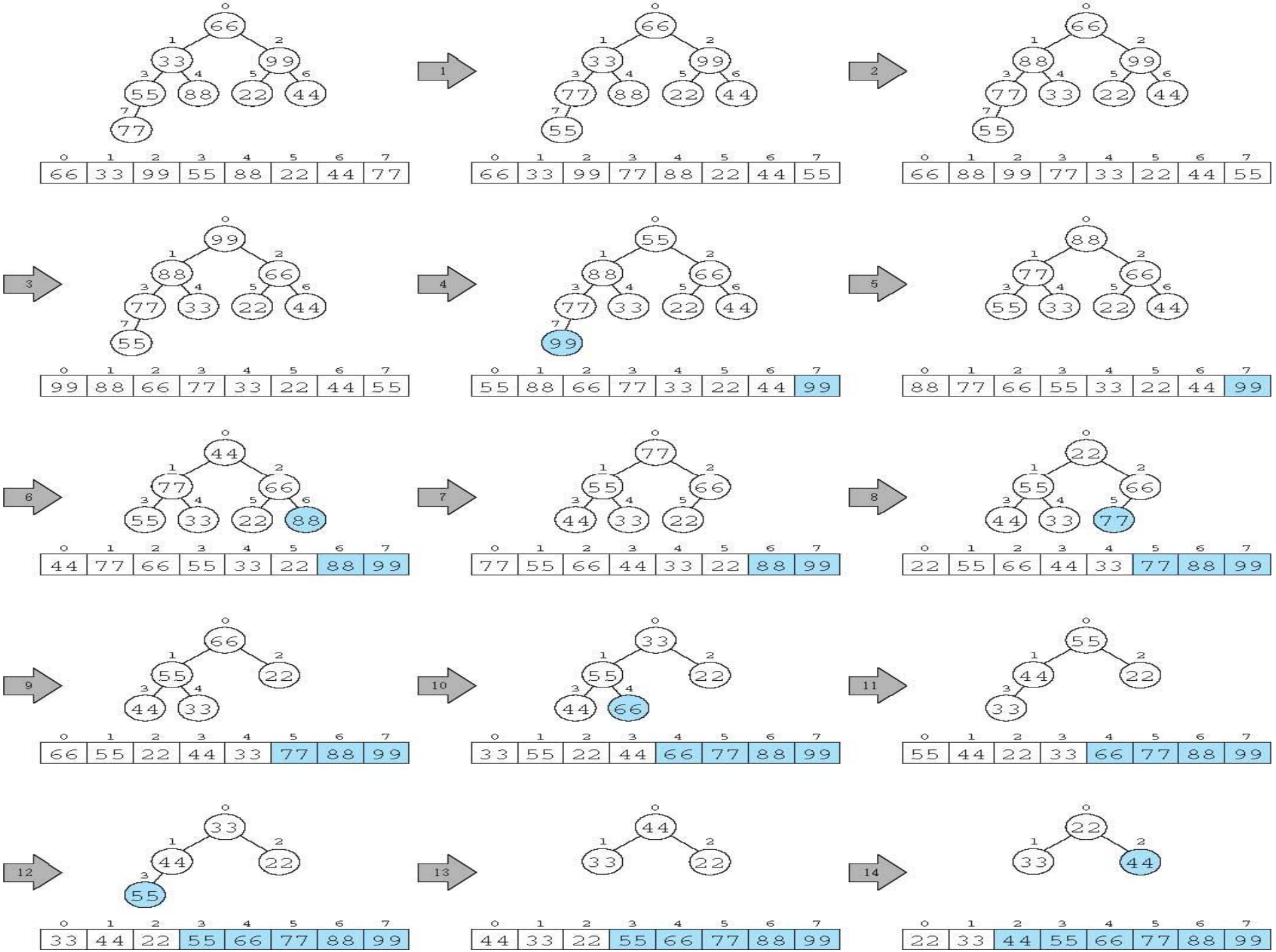
히프화 경로



히프화 메소드

- LISTING 14.1: The heapify() Method

```
1 void heapify(int[] a, int i, int n) {
2     int ai = a[i];
3     while (i < n/2) {                  // a[i] is not a leaf
4       int j = 2*i + 1;                 // a[j] is ai's left child
5       if (j+1 < n && a[j+1] > a[j])
            ++j;                         // a[j] is ai's larger child
6       if (a[j] <= ai) break;           // a[j] is not out of order
7       a[i] = a[j];                     // promote a[j]
8       i = j;                           // move down to next level
9     }
10    a[i] = ai;
11 }
```


66 33 99 55 88 22 44 77
1
66 33 99 77 88 22 44 55
2
66 88 99 77 33 22 44 55
3
99 88 66 77 33 22 44 55
4
55 88 66 77 33 22 44 99
5
88 77 66 55 33 22 44 99
6
44 77 66 55 33 22 88 99
7
77 55 66 44 33 22 88 99
8
22 55 66 44 33 77 88 99
9
66 55 22 44 33 77 88 99
10
33 55 22 44 66 77 88 99
11
55 44 22 33 66 77 88 99
12
33 44 22 55 66 77 88 99
13
44 33 22 55 66 77 88 99
14
22 33 44 55 66 77 88 99

15.9 버켓 정렬

- $\Theta(n^2)$: (버블, 선택, 삽입) 정렬
- $\Theta(n \lg n)$: (합병, 퀵, 히프) 정렬
- $\Theta(n)$: (버켓, 기수) 정렬

- *버켓 정렬(bucket sort)*
  - 임시로 키를 가지고 있는 "버켓(bucket)"이라 불리는 저장소 배열을 사용한다.
  - 이것은 키를 버켓에 분배하는 것으로 시작하는 "분배 정렬(distribution sort)"의 한 예를 보여 준다.
  - 아이디어는 n 개의 원소를 정렬하기 위해 n 개의 버켓을 사용하는 것이다.

버켓 정렬 알고리즘

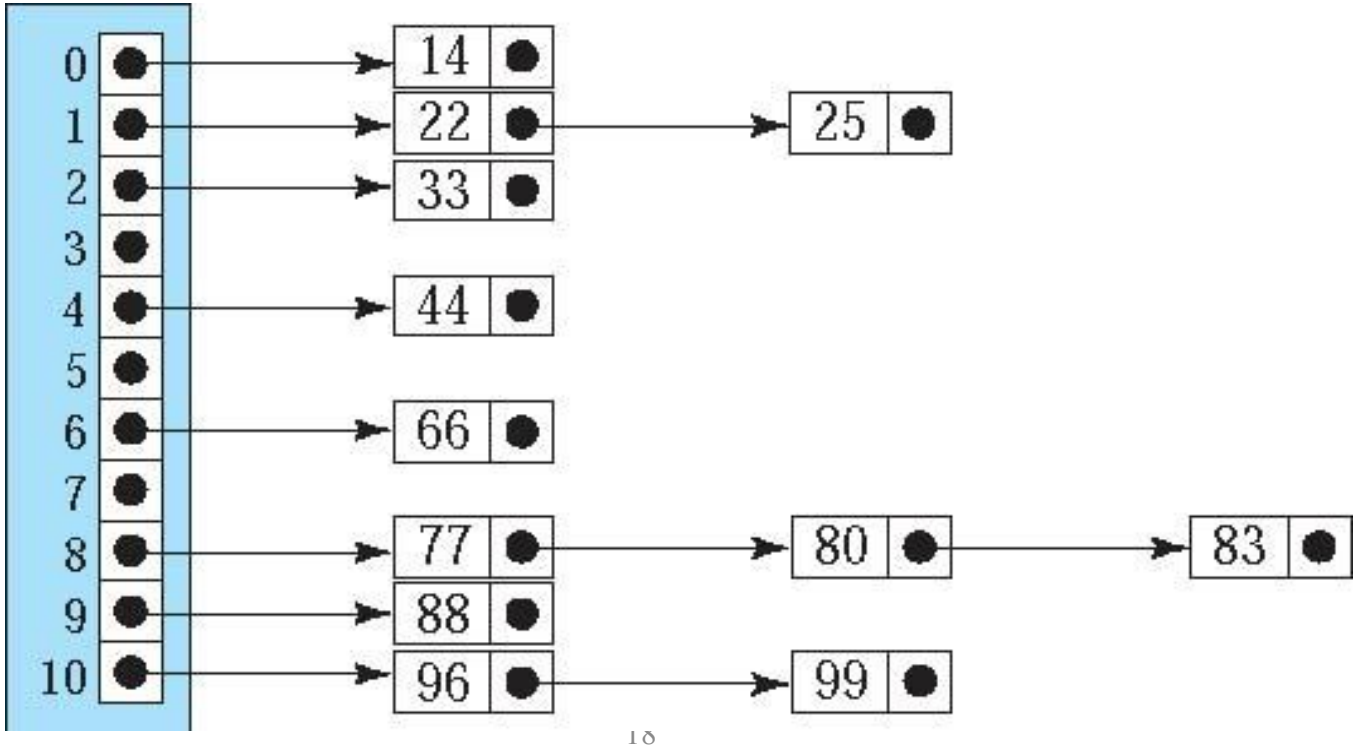
- 버켓 정렬 알고리즘
  - 입력 : n 개의 숫자 키 $\{a_0, a_1, a_2, \ldots, a_{n-1}\}$의 시이퀀스.
  - 선조건 : 키는 간격 $I=[a,b)=\{x|a \le x < b\}$ 내에서 균일하게 분포되어 있음.
  - 후조건 : 시이퀀스는 오름차순이 됨.

버켓 정렬 알고리즘

1. n 개의 버켓 $\{B_0, B_1, B_2, \ldots, B_{n-1}\}$을 할당.
2. 간격 [a,b)를 n 개의 동일한 부분간격(subinterval) $\{B_0, B_1, B_2, \ldots, B_{n-1}\}$ 의 시이퀀스로 분할.
   여기서 각각의 $I_j=[x_j, x_{j+1})$은 길이 $\Delta x=(b-a)/n$이므로,
   $x_0=a$, $x_1=a+\Delta x$, $x_2=a+2\Delta x$ 등이 된다.
3. a_i가 어떤 부분간격에 속하는지에 따라 버켓에 키 a_i 를 배분. 즉, $a_i \in I_j \Rightarrow a_i$는 버켓 B_j에 들어감.
4. 각각의 버켓을 정렬.
5. 각각의 버켓 내의 순서를 유지하도록
   버켓 $B_0, B_1, B_2, \ldots$의 순서에 따라 버켓으로부터
   키를 시이퀀스로 다시 복사.

버켓 정렬의 적용 예

예, 11 개의 키 66, 96, 22, 80, 14, 83, 77, 44, 25, 88, 33을 정렬하기 위해서는 Δx=(100-12)/11=8 이기 때문에, 부분간격은 12-19, 20-27, 28-35, …, 92-99가 된다.



15.10 계수 정렬

- *계수 정렬(counting sort)*
  - 이것은 또 다른 분배 정렬이다. 버켓 정렬과 같이, 이것도 값에 따라 키들을 분배하는데, 숫자 값이 키의 개수보다 훨씬 적을 경우 잘 동작한다.

- 계수 정렬 알고리즘
  - 입력 : 키 a= {a_0, a_1, a_2, …, a_{n-1}} 의 시이퀀스.
  - 선조건 : 시이퀀스에 있는 서로 다른 값의 개수 m이 길이 n에 비해 매우 적고, 그 값들은 정수 0, 1, …, m-1이 됨.
  - 후조건 : 시이퀀스는 오름차순이 됨.

계수 정렬 알고리즘

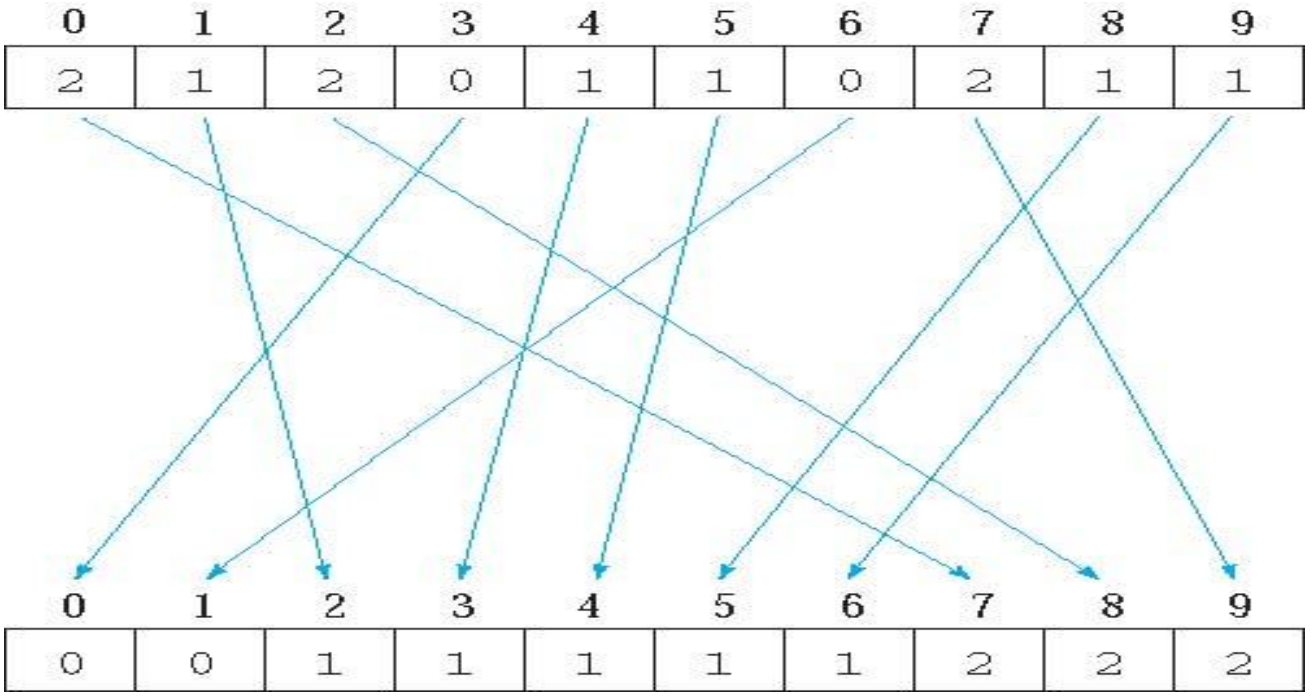
1. 각각의 원소가 0으로 초기화된 도수 시이퀀스 $c=\{c_0, c_1, c_2, \ldots, c_{m-1}\}$를 할당.
2. c에 있는 키의 개수를 세어 각각의 c_k가 k와 같은 키의 개수가 되도록 함.
3. 누적 분포를 이루도록 c를 재처리하여, 각각의 c_k가 k와 작거나 같은 키의 개수가 되도록 함.
4. 임시 시이퀀스 $b=\{b_0, b_1, b_2, \ldots, b_{n-1}\}$를 할당.
5. i=n-1에서 0으로 내려가면서, a_i=k이고 c_k=j때, a_i를 b_{j-1}로 복사하고, c_k를 감소시킴.
6. b를 a로 복사.

LISTING 15.13: The Counting Sort

```
1  void sort(int[] a) {
2      int n = a.length;
3      int m = size(a); // the number of distinct values of a[]
4      int[] c = new int[m]; // step 1
5      for (int i = 0; i < n; i++) // step 2
6          ++c[ a[i] ];
7      for (int k = 1; k < m; k++) // step 3
8          c[k] += c[k-1];
9      int[] b = new int[n]; // step 4
10     for (int i = n-1; i >= 0; i--) // step 5
11         b[ --c[ a[i] ] ] = a[i];
12     System.arraycopy(b, 0, a, 0, n); // step 6
13 }
```

계수 정렬의 수행 과정

a = {2, 1, 2, 0, 1, 1, 0, 2, 1, 1} 일 때,
도수 배열 *c* = {2, 5, 3} -> *c* = {2, 7, 10}



* 동일한 키가 뒤섞이지 않도록 하는 정렬알고리즘을 안정적(stable)하다고 한다

15.11 기수 정렬

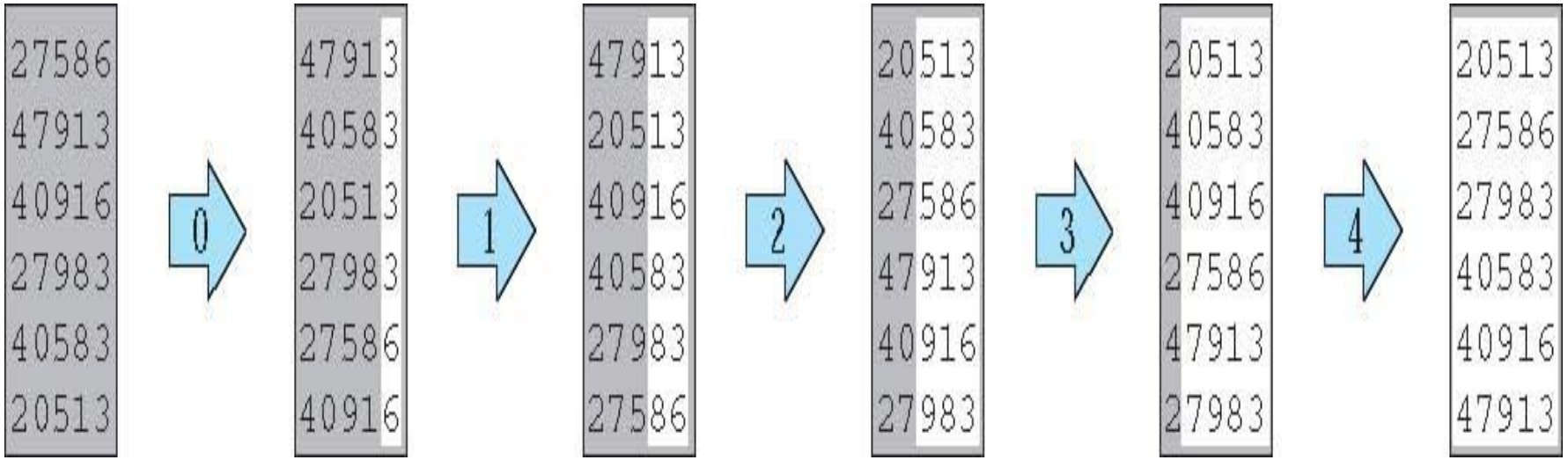
- *기수 정렬(radix sort)*
  - 모든 키가 동일한 길이 d를 가지는 스트링이고 스트링에 있는 모든 문자는 정수 집합 {0, 1, ..., r-1}에 속해 있다고 가정한다. 상수 r을 스트링의 *기수(radix)*라고 부른다.
  - 예를 보자.
    1. 미국의 사회 보장 번호(Social Security Number) (예를 들어, 055-36-8161) : d=9이고 r=10.
    2. 차량 식별 번호(예를 들어, 1HGCG1659YA065987) : d=17이고 r=36.
    3. 도서에 있는 ISBN(예를 들어, 3-89028-941-X) : d=10이고 r=11.
    4. 공항 코드(예를 들어, RIC) : d=3이고 r=26.

기수 정렬 알고리즘

- 기수 정렬 알고리즘
  - 입력 : 기수 r인 문자들의 d-숫자 문자열의 시이퀀스
  - 후조건 : 시이퀀스는 오름차순이 됨.

1. 최하위(least significant) j=0부터 시작하여 각각의 숫자 j에 대해 단계 2-3를 수행.
2. 숫자 j에 대한 키를 사용하여 해당하는 큐에 넣는다
3. 순서대로 큐 안에 넣어진 각 요소들을 모아 새로운 리스트를 형성한다.

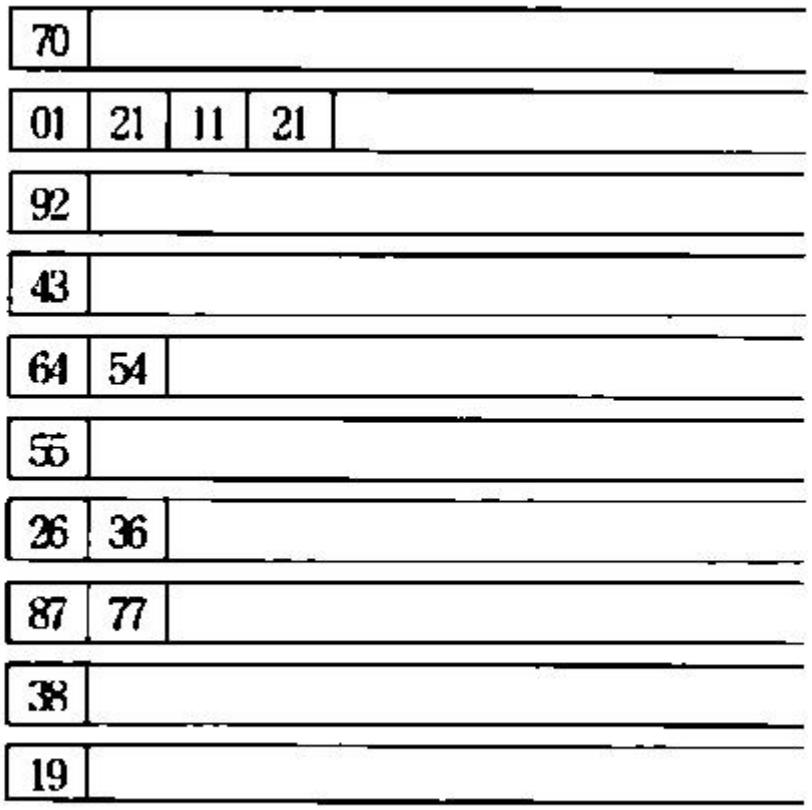
기수 정렬의 수행 과정


27586
47913
40916
27983
40583
20513
0
47913
40583
20513
27983
27586
40916
1
47913
20513
40916
40583
27983
27586
2
20513
40583
27586
47913
40916
27983
3
20513
40583
40916
27586
47913
27983
4
20513
27586
27983
40583
40916
47913

기수정렬 예

초기 리스트 19, 01, 26, 43, 92, 87, 21, 38, 11, 55, 21, 64, 54, 70, 36, 77

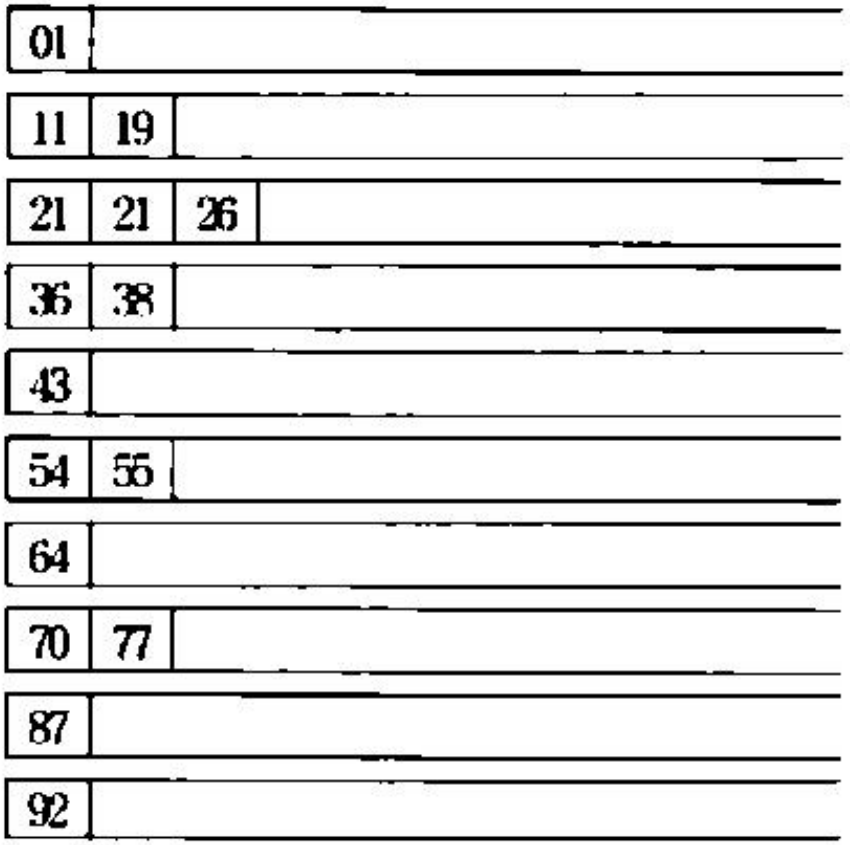
패스 1의 큐



재구성 리스트 70, 01, 21, 11, 21, 92, 43, 64, 54, 55, 26, 36, 87, 77, 38, 19

재구성 리스트 70, 01, 21, 11, 21, 92, 43, 64, 54, 55, 26, 36, 87, 77, 38, 19

패스 2의 큐



재구성 리스트 01, 11, 19, 21, 21, 26, 36, 38, 43, 54, 55, 64, 70, 77, 87, 92
(정렬된 리스트)