

# CÁC THUẬT GIẢI SẮP XẾP NÂNG CAO

- Định nghĩa thuật giải sắp xếp
- Heapsort
- Quicksort
- Sắp xếp thời gian tuyến tính

# ĐỊNH NGHĨA THUẬT GIẢI SẮP XẾP

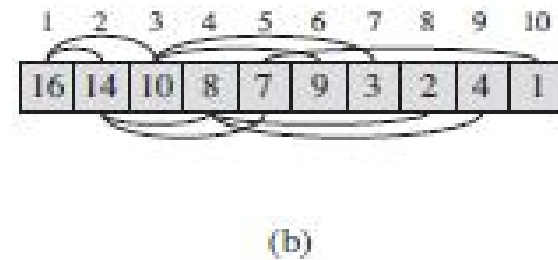
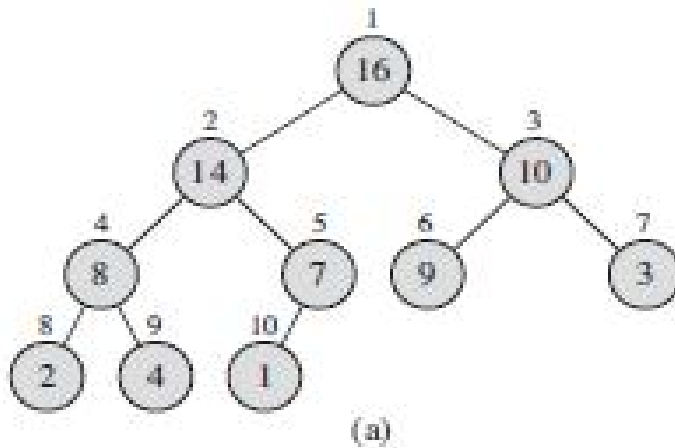
- Input: một dãy  $n$  số  $(a_1, a_2, \dots, a_n)$
- Output: một hoán vị của input  $(a'_1, a'_2, \dots, a'_n)$  sao cho
$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$
- Dùng mảng hoặc DSLK để biểu diễn dãy số

# THUẬT GIẢI HEAPSORT

- Cho một mảng A, một Heap biểu diễn A là một cây nhị phân có các tính chất sau:
  - Cây có thứ tự, mỗi nút tương ứng với một phần tử của mảng, gốc ứng với phần tử đầu tiên của mảng
  - Cây cân bằng và được lấp đầy trên tất cả các mức, ngoại trừ mức thấp nhất được lấp đầy từ bên trái sang (có thể chưa lấp đầy)

# THUẬT GIẢI HEAPSORT

- Một MaxHeap là một Heap mà giá trị của **mỗi nút lớn hơn (hoặc bằng) giá trị các nút con**



# THUẬT GIẢI HEAPSORT

- Gọi  $i$  là chỉ số thứ tự của một nút (của mảng), thì chỉ số các nút cha, con trái và con phải là:
  - $\text{Parent}(i) = \lfloor i/2 \rfloor$
  - $\text{Left}(i) = 2i$
  - $\text{Right}(i) = 2i + 1$

# THUẬT GIẢI HEAPSORT

- Thuật giải HeapSort **biến đổi mảng về MaxHeap** để sắp xếp như sau:
  - Biến đổi Heap (mảng) về một MaxHeap
  - Hoán đổi giá trị  $A[1]$  với  $A[n]$
  - **Loại nút  $n$  ra khỏi Heap** và chuyển Heap  $A[1..(n-1)]$  thành một MaxHeap (ít hơn một phần tử)
  - **Lặp lại các bước** trên cho đến khi Heap chỉ còn một phần tử

# THUẬT GIẢI HEAPSORT

- Cần hai thủ tục (thuật giải) hỗ trợ cho HeapSort:
  - $\text{MaxHeappify}(A[1..n], i)$ , biến đổi cây con có gốc tại  $i$  thành một MaxHeap (gốc tại  $i$ )
  - $\text{BuildMaxHeap}(A[1..n])$ , biến đổi mảng (Heap) thành MaxHeap

# MaxHeapify

- Đầu vào là một mảng (heap)  $A$  và chỉ số  $i$  trong mảng
- Các cây nhị phân được định gốc tại  $\text{Left}(i)$  và  $\text{Right}(i)$  là các MaxHeap nhưng  $A[i]$  có thể nhỏ hơn các con của nó
- MaxHeapify đẩy giá trị  $A[i]$  xuống sao cho cây con định gốc tại  $A[i]$  là một MaxHeap

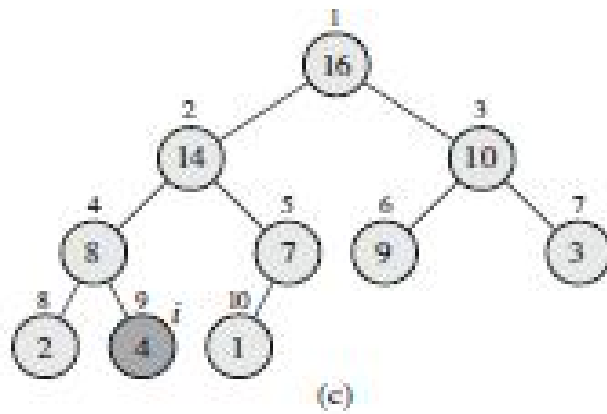
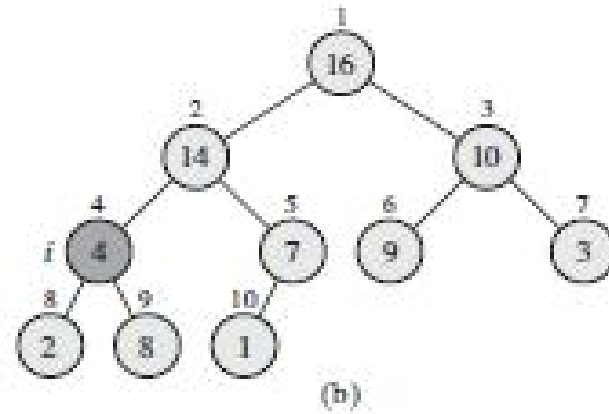
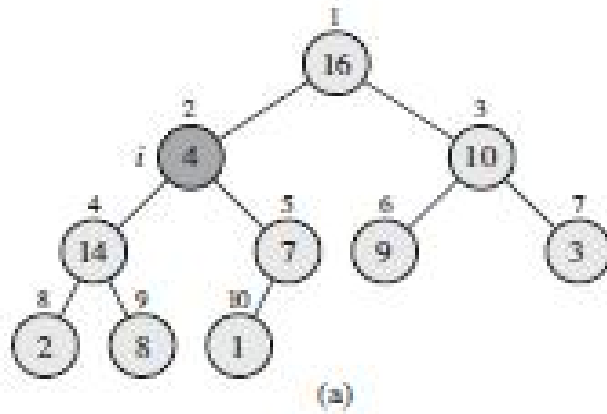


# MaxHeapify

MaxHeapify( $A[1..n]$ ,  $i$ )

```
1  l = Left(i); r = Right(i)
2  if l ≤ n and A[l] > A[i] largest = l
3  else largest = i
4  if r ≤ n and A[r] > A[largest] largest = r
5  if largest ≠ i
6      Exchange(A[i], A[largest])
7      MaxHeapify(A, largest)
```

# MaxHeapify



# BuildMaxHeap

- Các nút có chỉ số  $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$  trong  $A[1..n]$  là **các lá của cây**, mỗi nút như vậy là một MaxHeap
- BuildMaxHeap áp dụng MaxHeapify cho các nút con khác lá của cây từ **dưới lên gốc bắt đầu từ nút  $\lfloor n/2 \rfloor$**
- Kết quả là một MaxHeap tương ứng với  $A[1..n]$

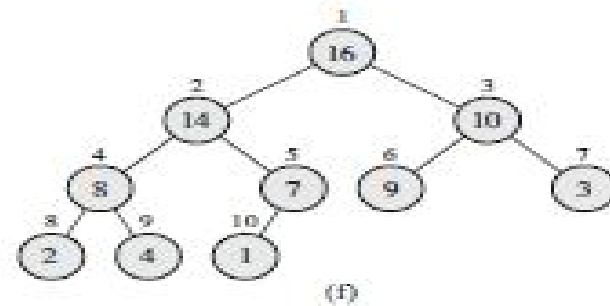
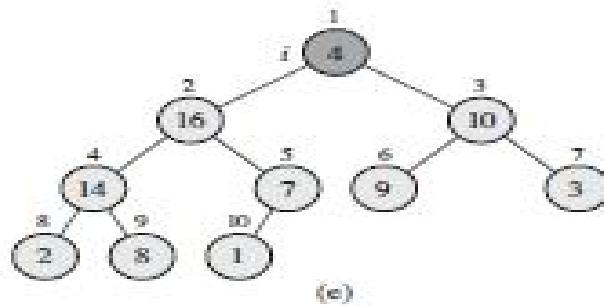
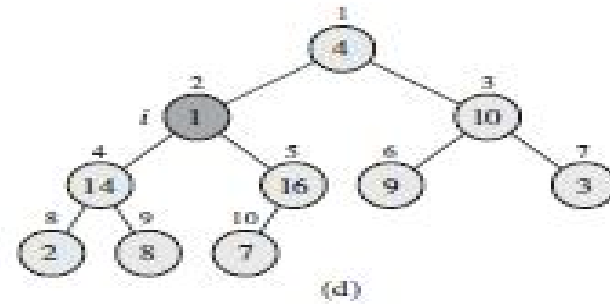
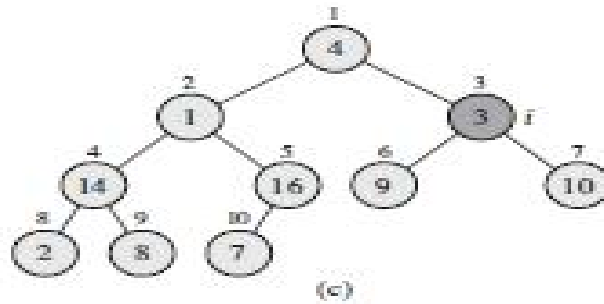
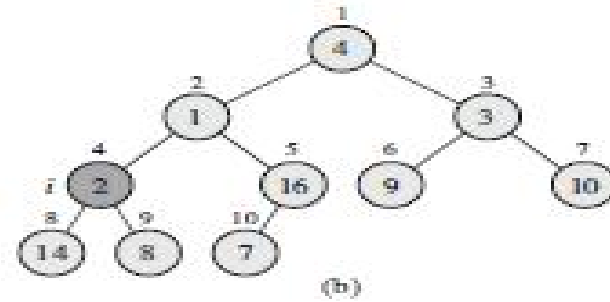
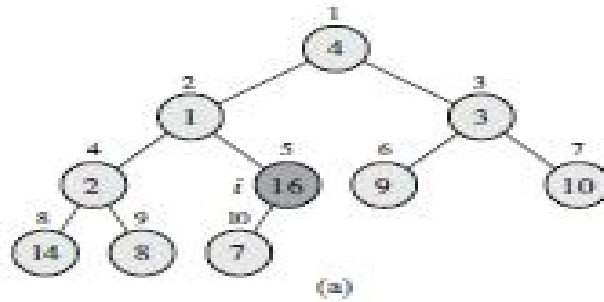
# BuildMaxHeap

BuildMaxHeap( $A[1..n]$ )

```
1  for  $i = n/2$  downto 1  
2      do MaxHeapify( $A, i$ )
```

# BuildMaxHeap

A [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]



# HeapSort

HeapSort( $A[1..n]$ )

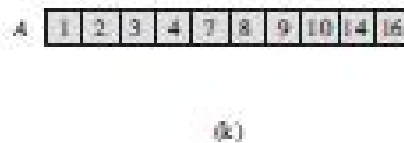
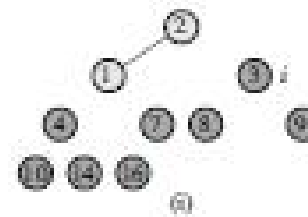
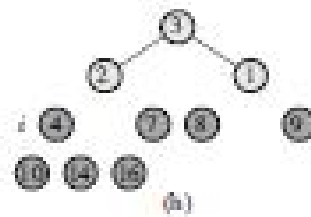
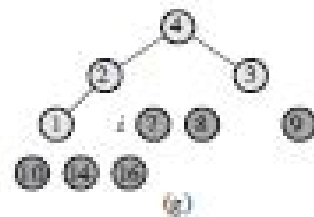
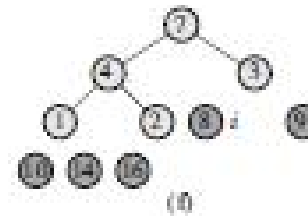
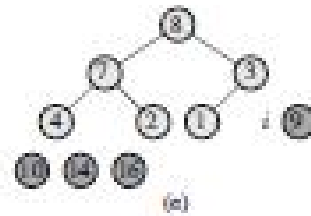
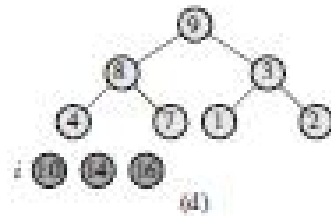
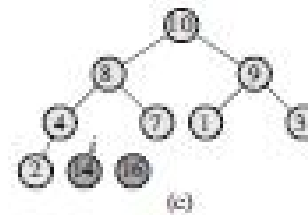
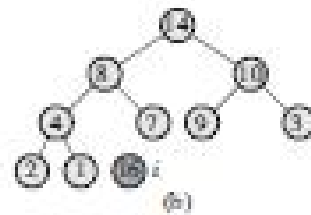
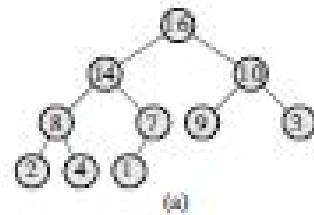
1 BuildMaxHeap( $A$ )

2 **for**  $i = n$  **downto** 2

3     **do** Exchange( $A[1]$ ,  $A[i]$ )

4         MaxHeapify( $A[1..i-1]$ , 1)

# HeapSort



# HeapSort

- Kích thước đầu vào là  $n$  (số phần tử của mảng)
- Gọi chiều cao của cây là  $h$ , do cây cân bằng nên  $h = \lfloor \log_2 n \rfloor$ 
  - Thời gian chạy của MaxHeapify là  $O(h) = O(\log_2 n)$
  - Thời gian chạy của BuildMaxHeap tối đa là  $O(n \log_2 n)$
  - Thời gian chạy của vòng lặp 2-4 là  $O(n \log_2 n)$
- Vậy thời gian chạy của HeapSort là
$$T(n) = O(n \log_2 n) + O(n \log_2 n) = O(n \log_2 n)$$



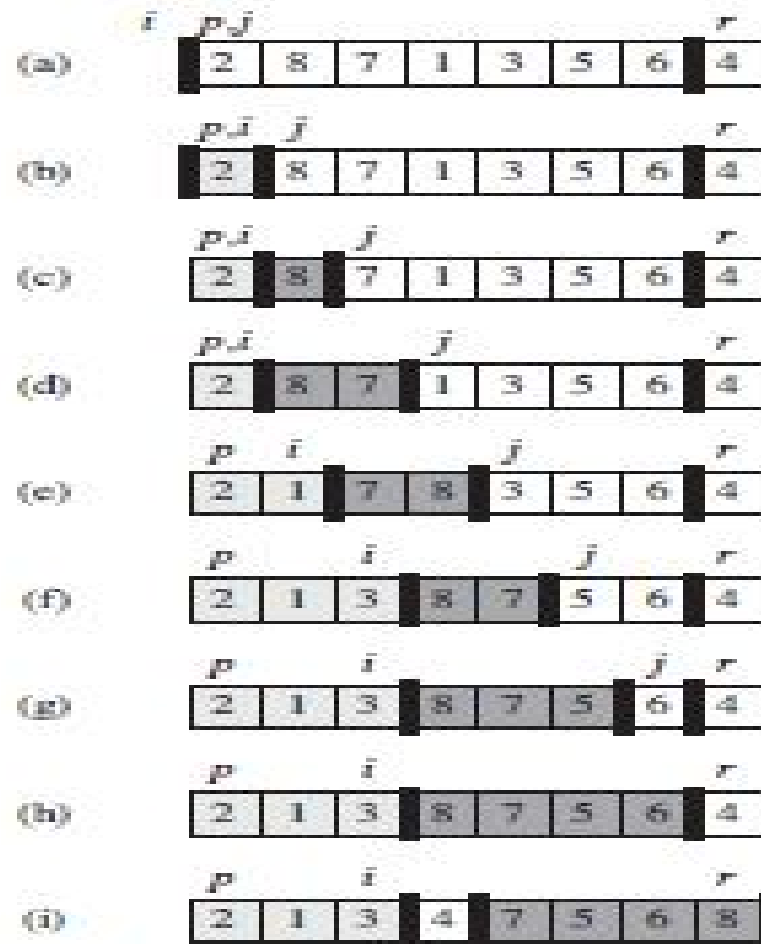
# QUICKSORT

- Được thiết kế dựa trên kỹ thuật chia để trị (divide-and-conquer):
  - **Divide:** Phân hoạch  $A[p..r]$  thành hai mảng con  $A[p..q-1]$  và  $A[q+1..r]$  có các phần tử tương ứng **nhỏ hơn hoặc bằng  $A[q]$  và lớn hơn  $A[q]$**
  - **Conquer:** Sắp xếp hai mảng con  $A[p..q-1]$  và  $A[q+1..r]$  bằng lời gọi đệ qui

# Quicksort (C++)

```
void QuickSort(int A[], int p, int r)
{
    int q;
    if(p<r)
    {
        q=Partition(A,p,r);
        QuickSort(A, p,q-1);
        QuickSort(A, q+1,r);
    }
}
```

# Partition



# Partition (C++)

```
int Partition(int A[], int p, int r)
{
    int i, j, x;
    x=A[r];
    i=p-1;
    for(j=p; j<r; j++)
        if(A[j]<=x)
        {
            i=i+1;
            exchange(A[i], A[j]); //hoán vị A[i] và A[j]
        }
    exchange(A[i+1], A[r]);
    return i+1;
}
```

# ĐỘ PHỨC TẠP QUICKSORT

- Partition có  $T(n) = O(n)$

- Đối với Quicksort

Tốt nhất  $T(n) = O(n \log_2 n)$

Xấu nhất  $T(n) = O(n^2)$

Trung bình  $T(n) = O(n \log_2 n)$

# SẮP XẾP THỜI GIAN TUYỂN TÍNH

- Khái niệm
- Sắp xếp bằng đếm
- Sắp xếp theo lô

# KHÁI NIỆM

- Thuật giải sắp xếp **thời gian tuyến tính** là thuật giải có **thời gian chạy  $O(n)$**
- Các thuật giải tốt như Heapsort, Quicksort có thời gian chạy  $O(n \lg n)$

# KHÁI NIỆM

- Các thuật giải Heapsort, Quicksort dùng phương pháp so sánh, hoán đổi để sắp xếp
- Các thuật giải tuyến tính dựa trên thông tin của các phần tử để sắp xếp nên giảm được bậc của độ phức tạp



# SẮP XẾP BẰNG ĐẾM

- Cho  $k$  là một số nguyên, sắp xếp **bằng đếm** (counting sort) giả sử mỗi một phần tử trong dãy input là một **số nguyên trong miền từ 0 đến  $k$**

# SẮP XẾP BẰNG ĐẾM

- Ý tưởng là đếm số phần tử nhỏ hơn phần tử  $x$  trong mảng nhập để đặt  $x$  trực tiếp vào vị trí của nó trong mảng xuất
- Chẳng hạn, nếu có 17 phần tử nhỏ hơn hoặc bằng  $x$  thì  $x$  được đặt vào vị trí 17

# SẮP XẾP BẰNG ĐẾM

COUNTING-SORT( $A, B, k$ )

```
1  let  $C[0 \dots k]$  be a new array
2  for  $i = 0$  to  $k$ 
3       $C[i] = 0$ 
4  for  $j = 1$  to  $A.length$ 
5       $C[A[j]] = C[A[j]] + 1$ 
6  //  $C[i]$  now contains the number of elements equal to  $i$ .
7  for  $i = 1$  to  $k$ 
8       $C[i] = C[i] + C[i - 1]$ 
9  //  $C[i]$  now contains the number of elements less than or equal to  $i$ .
10 for  $j = A.length$  downto 1
11      $B[C[A[j]]] = A[j]$ 
12      $C[A[j]] = C[A[j]] - 1$ 
```

# SẮP XẾP BẰNG ĐẾM

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

(c)

	1	2	3	4	5	6	7	8
B		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

(d)

	1	2	3	4	5	6	7	8
B		0				3	3	

	0	1	2	3	4	5
C	1	2	4	5	7	8

(e)

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

(f)

# SẮP XẾP BẰNG ĐẾM

- Chi phí cho lệnh 1-2 là  $O(k)$
- Chi phí cho lệnh 3-4 là  $O(n)$
- Chi phí cho 6-7 là  $O(k)$
- Chi phí cho 9-11 là  $O(n)$ 
  - Vì vậy tổng chi phí thời gian là  $O(k + n)$
  - Nếu  $k = O(n)$  thì tổng chi phí là  $O(n)$ .

# SẮP XẾP BẰNG ĐẾM

- COUNTING-SORT chạy thời gian tuyến tính và hiệu quả hơn các thuật giải sắp xếp bằng so sánh
- COUNTING-SORT chỉ sắp xếp các phần tử có khoá trong một miền nhất định (nhỏ hơn hoặc bằng  $k$  cho trước)
- COUNTING-SORT phải sử dụng thêm các mảng trung gian

# SẮP XẾP THEO LÔ

- Sắp xếp **theo lô** (Bucket sort) giả sử input là một mảng  $n$  số không âm nhỏ hơn 1

# SẮP XẾP THEO LÔ

- Ý tưởng của Bucketsort
  - Phân bố mảng input vào  $n$  khoảng con (lô) của khoảng  $[0, 1)$
  - Sắp xếp các phần tử trong mỗi lô và nối các lô để có mảng được sắp

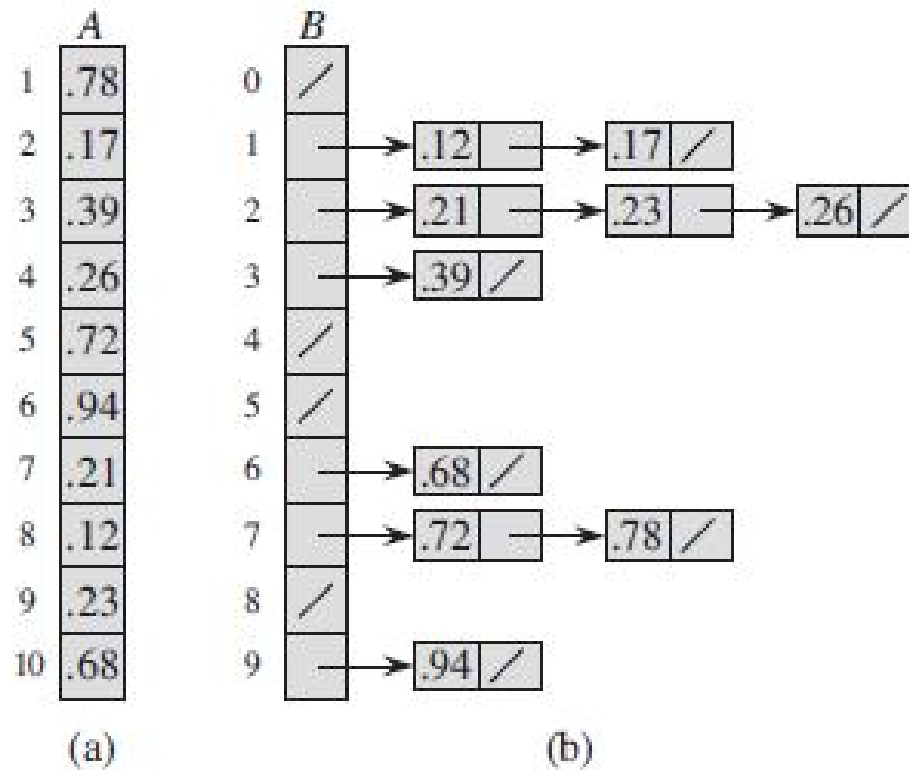


# SẮP XẾP THEO LÔ

BUCKET-SORT( $A$ )

```
1  let  $B[0..n-1]$  be a new array
2   $n = A.length$ 
3  for  $i = 0$  to  $n-1$ 
4      make  $B[i]$  an empty list
5  for  $i = 1$  to  $n$ 
6      insert  $A[i]$  into list  $B[\lfloor nA[i] \rfloor]$ 
7  for  $i = 0$  to  $n-1$ 
8      sort list  $B[i]$  with insertion sort
9  concatenate the lists  $B[0], B[1], \dots, B[n-1]$  together in order
```

# SẮP XẾP THEO LÔ



# SẮP XẾP THEO LÔ

- Do phân bố ngẫu nhiên  $n$  phần tử vào  $n$  khoảng con nên trung bình mỗi lô có 1 phần tử, vì vậy thời gian sắp xếp chèn là  $O(1)$
- Từ đó, chi phí toàn bộ của thuật giải là  $O(n)$

# SẮP XẾP THEO LÔ

- BUCKET-SORT chạy thời gian tuyến tính và hiệu quả hơn các thuật giải sắp xếp bằng so sánh
- BUCKET-SORT chỉ sắp xếp các phần tử có khoá trong khoảng  $[0, 1)$
- Không phải mọi phân bố sẽ cho mỗi lô chứa 1 phần tử

# ĐỌC VÀ TÌM HIỂU Ở NHÀ

- Đọc chương 6, 7, 8 sách Introduction to Algorithms của Thomas H. Cormen, Charles E. Leiserson, Ronald D. Rivest
- Làm bài tập về nhà chương 2 đã cho trong DS bài tập