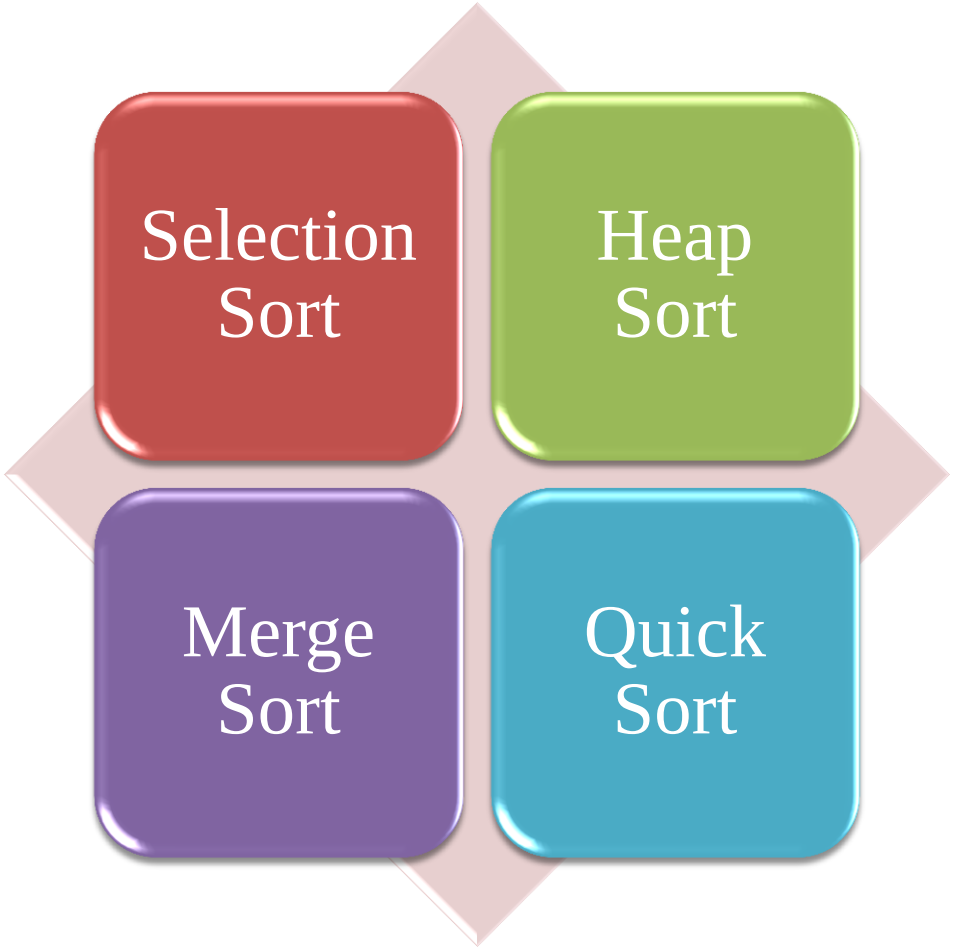
Cấu trúc dữ liệu và giải thuật

CÁC THUẬT TOÁN SẮP XẾP

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến
Vũ Thanh Hưng

Nội dung



Giới thiệu

Bài toán sắp xếp

Các thuật toán sắp xếp

Giới thiệu

- Bài toán sắp xếp: Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự thỏa yêu cầu cho trước
- Ví dụ: danh sách trước khi sắp xếp:

  {1, 25, 6, 5, 2, 37, 40}

  Danh sách sau khi sắp xếp:

  {1, 2, 5, 6, 25, 37, 40}
- Thông thường, sắp xếp giúp cho việc tìm kiếm được nhanh hơn.

Giới thiệu

- Các phương pháp sắp xếp thông dụng:
  - Bubble Sort $O(n^2)$
  - Selection Sort $O(n^2)$
  - Insertion Sort $O(n^2)$
  - Quick Sort $O(n\log_2 n)$
  - Merge Sort $O(n\log_2 n)$
  - Heap Sort $O(n\log_2 n)$
  - Radix Sort $O(n)$
- Cần tìm hiểu các phương pháp sắp xếp và lựa chọn phương pháp phù hợp khi sử dụng.

Sắp xếp chọn

Selection Sort

Ý tưởng

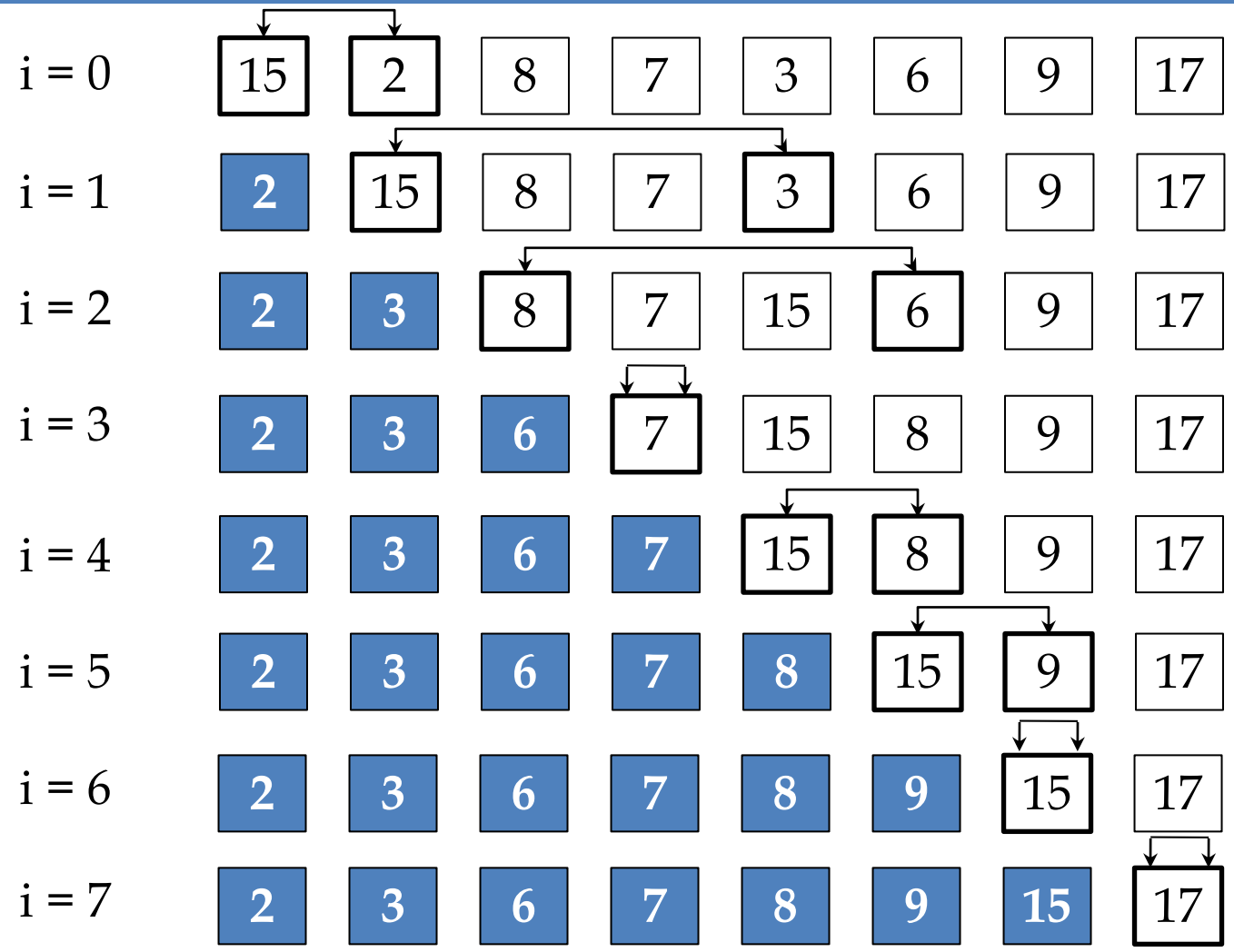
- Mô phỏng cách sắp xếp tự nhiên nhất trong thực tế
  - Chọn phần tử nhỏ nhất và đưa về vị trí đúng là đầu dãy hiện hành.
  - Sau đó xem dãy hiện hành chỉ còn n-1 phần tử.
  - Lặp lại cho đến khi dãy hiện hành chỉ còn 1 phần tử.

Thuật toán

Các bước của thuật toán:

- Bước 1. Khởi gán *i = 0*.
- Bước 2. Bước lặp:
  - 2.1. *Tìm a[min]* nhỏ nhất trong dãy từ a[i] đến a[n-1]
  - 2.2. *Hoán vị a[min] và a[i]*
- Bước 3. So sánh i và n:
  - Nếu i ≤ n thì *tăng i thêm 1* và lặp lại bước 2.
  - Ngược lại: Dừng thuật toán.

Ví dụ


i = 0
15 2 8 7 3 6 9 17
i = 1
2 15 8 7 3 6 9 17
i = 2
2 3 8 7 15 6 9 17
i = 3
2 3 6 7 15 8 9 17
i = 4
2 3 6 7 15 8 9 17
i = 5
2 3 6 7 8 15 9 17
i = 6
2 3 6 7 8 9 15 17
i = 7
2 3 6 7 8 9 15 17

Chú ý:


```
void SelectionSort(int a[], int n)
{
    for(int i = 0;i<n-1;i++)
    {
        for(int j = i+1;j<n;j++)
        {
                if(a[i]> a[j])
                        HoanVi(a[i], a[j]);
        }
    }
}
```

Sắp xếp nhanh

Quick Sort

Ý tưởng

- Giải thuật: dựa trên việc phân hoạch dãy ban đầu thành 2 phần:
  - Dãy con 1: $a_0, a_1, \ldots, a_i$ có giá trị nhỏ hơn x
  - Dãy con 2: $a_j, \ldots, a_{n-1}$ có giá trị lớn hơn x.

  Dãy ban đầu được phân thành 3 phần:

$a_k < x$	$a_k = x$	$a_k > x$
k = 0 …i	k = i+1 … j	k = j+1, … n-1

  - Phần 2 đã có thứ tự
  - Phần 1, 3: cần sắp thứ tự, tiến hành phân hoạch từng dãy con theo cách phân hoạch dãy ban đầu

Thuật toán – Giai đoạn phân hoạch

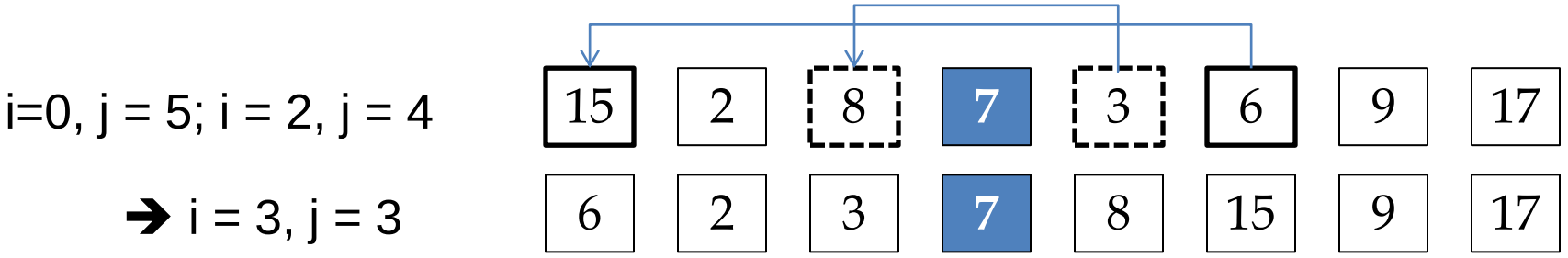
1. Chọn phần tử a[k] trong dãy là giá trị mốc, 0 ≤ k ≤ r-1
   - Gán x = a[k], i = l, j = r.
   - Thường chọn phần tử ở giữa dãy: k = (l+r)/2
2. Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] sai vị trí:
   - 2.1. Trong khi (a[i] < x), tăng i.
   - 2.2. Trong khi (a[j] >x), giảm j.
   - 2.3. Nếu i <= j thì:
     - Hoán vị a[i], a[j],
     - Tăng i và giảm j
3. So sánh i và j:
   - Nếu i < j: lặp lại bước 2
   - Ngược lại: dừng phân hoạch.

Thuật toán Quick Sort

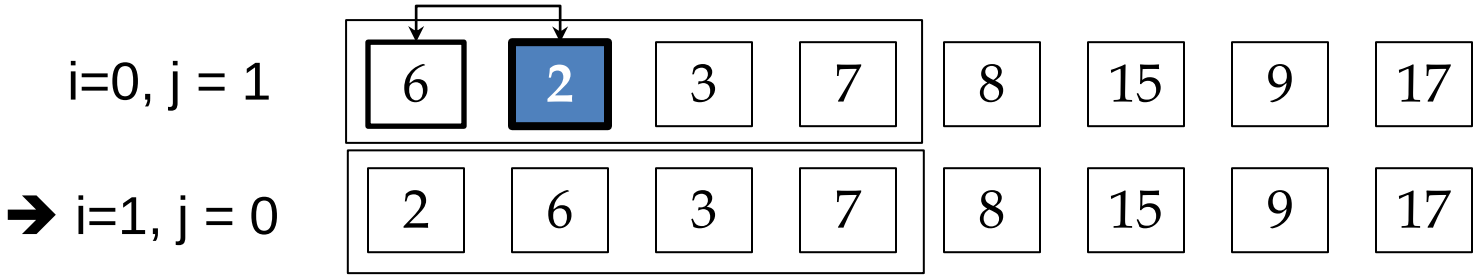
- Bước 1: phân hoạch dãy ban đầu thành 3 dãy:
  - Dãy 1: $a_l \ldots a_j < x$
  - Dãy 2: $a_{j+1} \ldots a_{i-1} = x$
  - Dãy 3: $a_i \ldots a_r > x$
- Bước 2: sắp xếp:
  - Nếu $l < j$: phân hoạch dãy $a_l \ldots a_j$
  - Nếu $i < r$: phân hoạch dãy $a_i \ldots a_r$

Ví dụ

1. Phân hoạch dãy ban đầu: l = 0, r = 7, x = a[3]

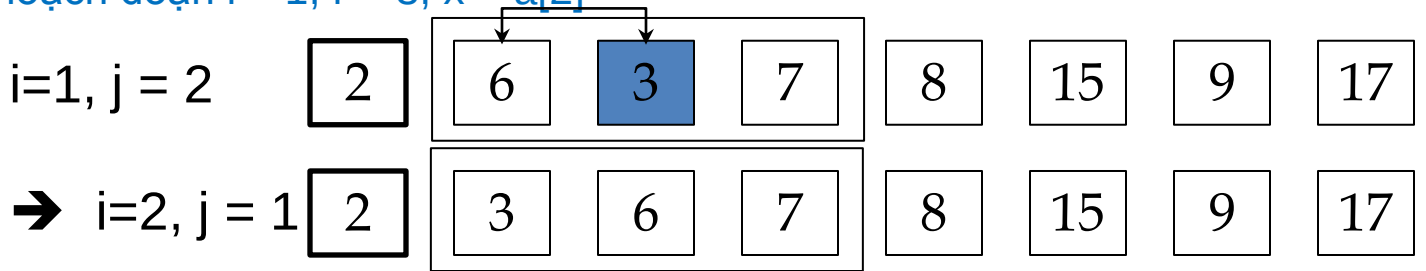


1.1 Phân hoạch đoạn l = 0, r = 3, x = a[1]



1.1.1Phân hoạch đoạn l = 0, r = 0 → một phần tử→không sắp xếp

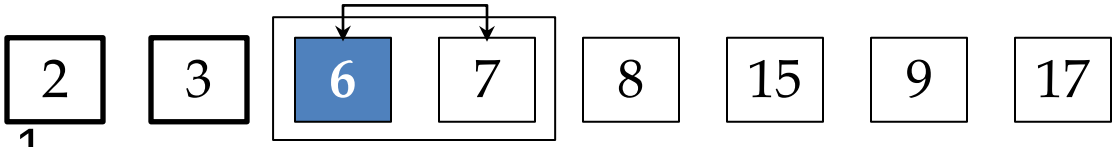
1.1.2Phân hoạch đoạn l = 1, r = 3, x = a[2]



Ví dụ (tt)

- 1.1.2.1 Phân hoạch đoạn l = 1, r = 1 → một phần tử→không sắp xếp
- 1.1.2.2Phân hoạch đoạn l = 2, r = 3, x = a[2]

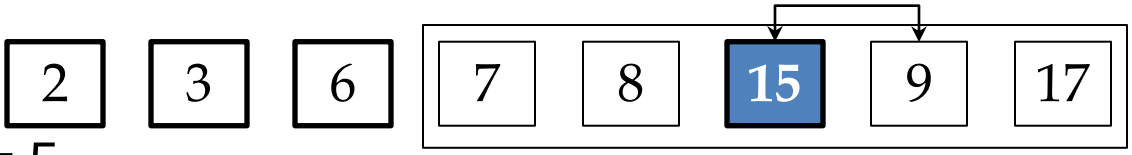
i= 2, j = 2



➔ i= 3, j = 1

- 1.2 Phân hoạch đoạn l = 3, r = 7, x = a[5]

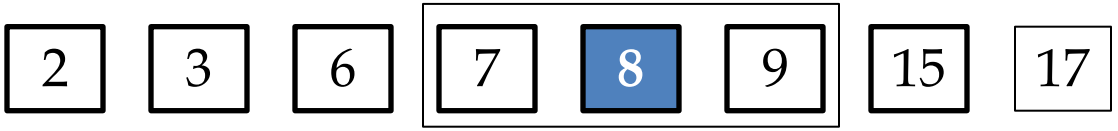
i=5, j = 6



➔ i= 6, j = 5

- 1.2.1Phân hoạch đoạn l = 3, r = 5, x = a[4]

i=4, j = 4



➔ i=5, j = 3

- 1.2.2 Phân hoạch đoạn l = 6, r = 7, x = a[6]

i=6, j = 6

➔ i=7, j = 5

Code

```
void QuickSort(int a[], int l, int r)
{
    i = l;  j = r; x = a[(l+r)/2];
    do {
        while (a[i] < x) i++;
        while (a[j] > x) j--;
        if (i <= j){
            HoanVi(a[i], a[j]);
            i++;     j--;
        }
    } while (i < j);
    if (l < j)
        QuickSort(a, l, j);
    if (i < r)
        QuickSort(a, i, r);
}
```

- 1
  - 1.1
    - 1.1.1
    - 1.1.2
  - 1.2
    - 1.2.1
    - 1.2.2

Bài tập

- Chạy tay thuật toán Quick Sort để sắp xếp mảng A trong 2 trường hợp tăng dần và giảm dần.

  A = {2, 9, 5, 12, 20, 15, -8, 10}

Quick Sort

- Đánh giá giải thuật:
  - Hiệu quả phụ thuộc vào việc chọn giá trị mốc
    - Tốt nhất là phần tử median.
    - Nếu phần tử mốc là cực đại hay cực tiểu thì việc phân hoạch không đồng đều.
  - Bảng tổng kết:

	Độ phức tạp
Tốt nhất	n*log(n)
Trung bình	n*log(n)
Xấu nhất	n^2

Sắp xếp vun đống

Heap Sort

Ý tưởng

- Ý tưởng: khi tìm phần tử nhỏ nhất ở bước i, phương pháp Selection sort không tận dụng được các thông tin đã có nhờ vào các phép so sánh ở bước i-1 → cần khắc phục nhược điểm này.
- J. Williams đã đề xuất phương pháp sắp xếp Heapsort.

Heap

- Định nghĩa Heap:
  - Giả sử xét trường hợp sắp xếp tăng dần, Heap được định nghĩa là một dãy các phần tử $a_l, a_{l+1}, \ldots a_r$ thỏa: với mọi i thuộc [l,r] (chỉ số bắt đầu từ 0)

    $a_i \geq a_{2i+1}$

    $a_i \geq a_{2i+2}$ {(a_i,a_{2i+1}), (a_i,a_{2i+2}) là các cặp phần tử liên đới}

Các tính chất của Heap

- Nếu a_l, a_{l+1}, … a_r là một heap thì phần tử a_l (đầu heap) luôn là phần tử lớn nhất.
- Mọi dãy a_i, a_{i+1}, … a_r với $2i + 1 > r$ là heap.

Thuật toán

- Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành heap (bắt đầu từ phần tử giữa của dãy)
- Giai đoạn 2: sắp xếp dựa trên heap.
  - Bước 1: đưa phần tử lớn nhất về vị trí đúng ở cuối dãy
  - Bước 2:
    - Loại bỏ phần tử lớn nhất ra khỏi heap: r = r – 1
    - Hiệu chỉnh lại phần còn lại của dãy.
  - Bước 3: So sánh r và l:
    - Nếu r > l thì lặp lại bước 2.
    - Ngược lại, dừng thuật toán.

Heap Sort

- Mã giả (Tựa ngôn ngữ lập trình C):

```
void HeapSort(int a[], int n)
{
  TaoHeap(a,n-1);
  r = n-1;
  while(r > 0)
  {
      HoanVi(a[0], a[r]);
      r = r - 1;
      HieuChinh(a,0,r);
  }
}
```

Heap Sort

- Mã giả:

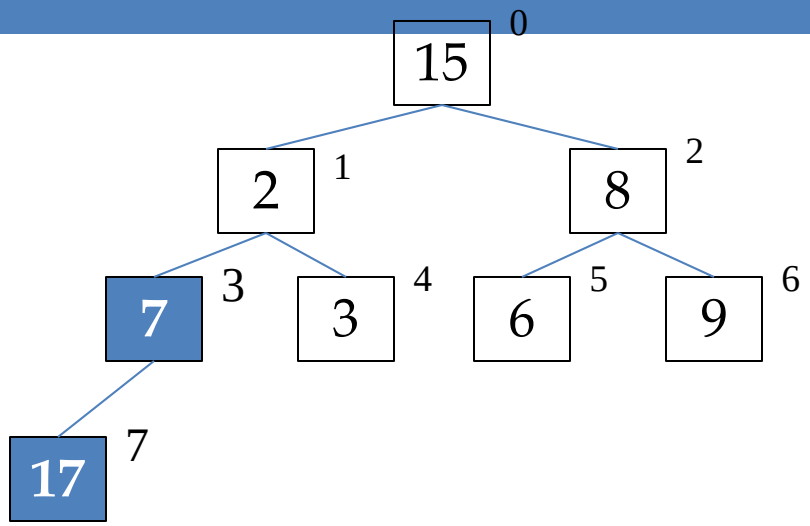
```
void TaoHeap(int a[], int r)
{
  int l = r/2;
  while(l > 0)
  {
     HieuChinh(a,l,r);
     l = l - 1;
  }
}
```

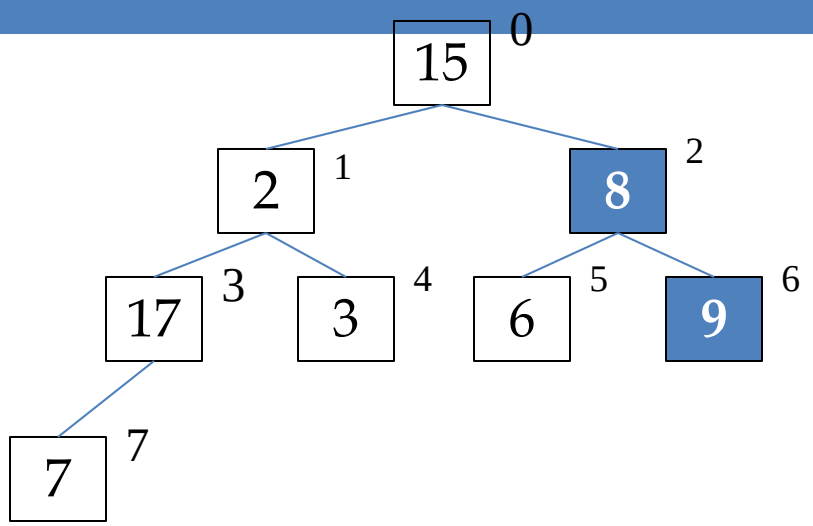
Heap Sort

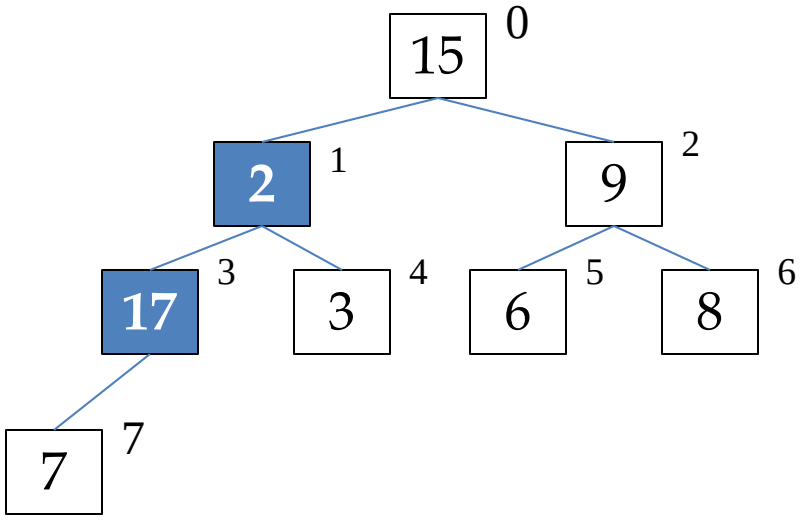
- Mã giả:

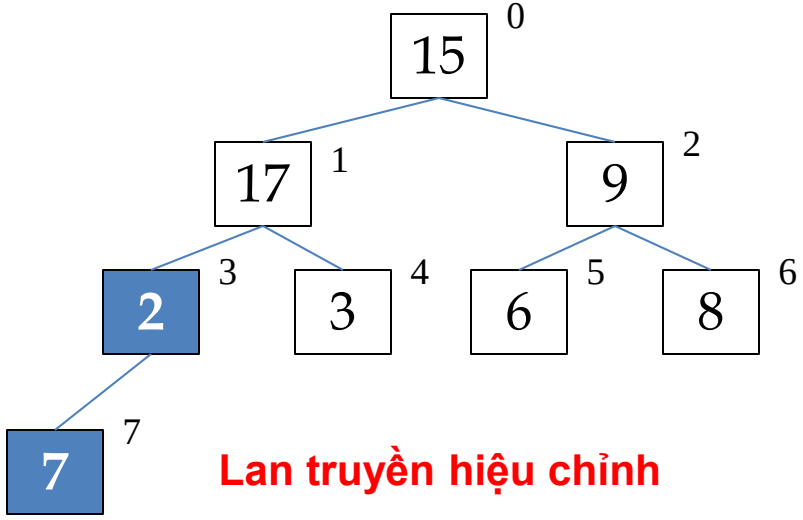
```
void HieuChinh(int a[], int l, int r)
{
   i = l; j = 2*i+1; x = a[i];
   while(j <= r)
   {
        if(có đủ 2 phần tử liên đới)
               //xác định phần tử liên đới lớn nhất
        if(a[j] < x) //thỏa quan hệ liên đới
               //dừng
        else
               //hiệu chỉnh
            //xét khả năng hiệu chỉnh lan truyền
   }
}
```

Ví dụ



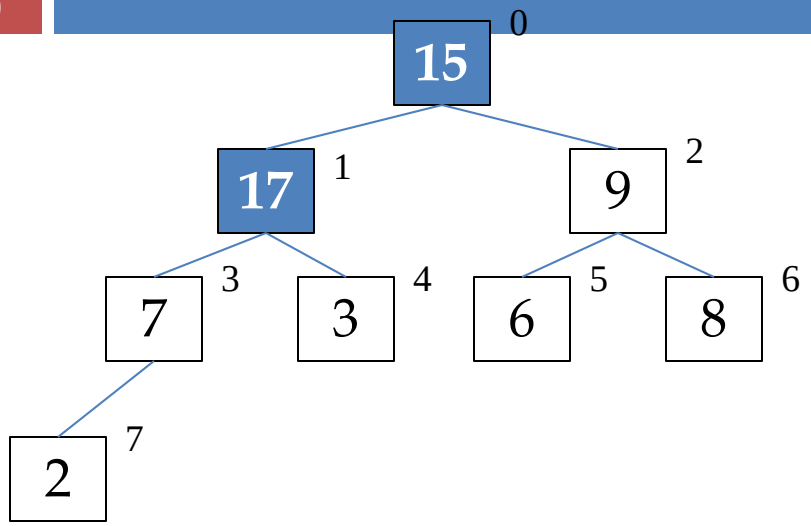


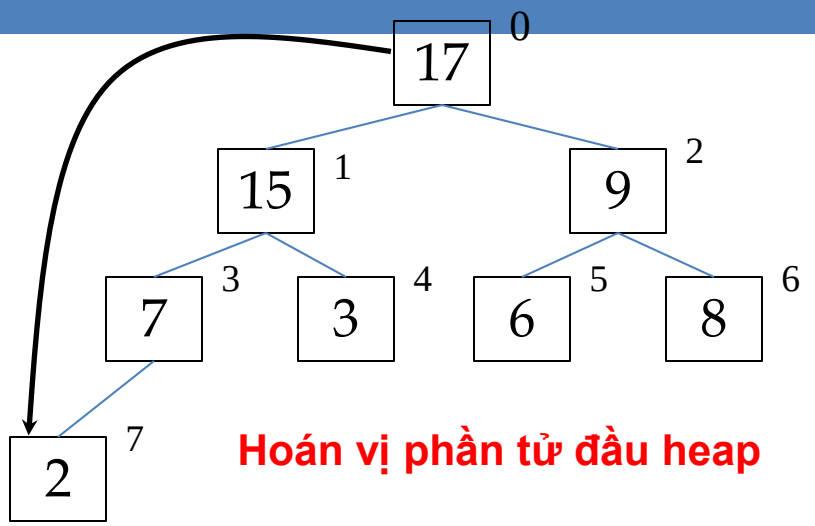




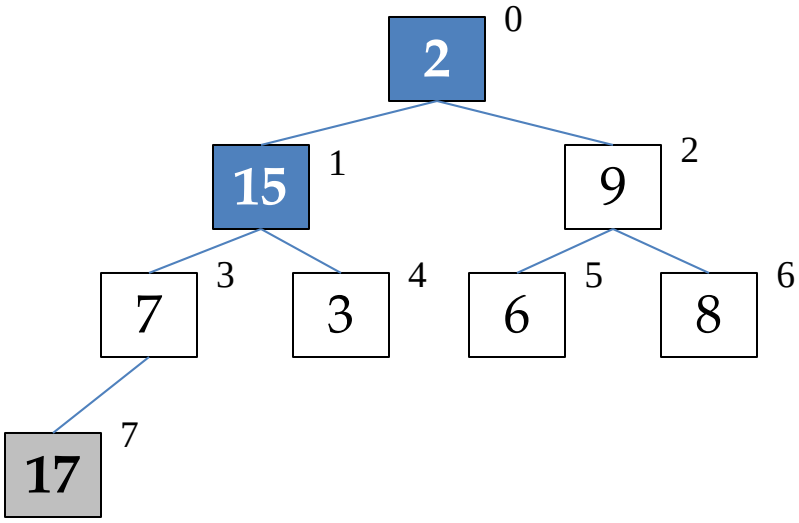
Lan truyền hiệu chỉnh

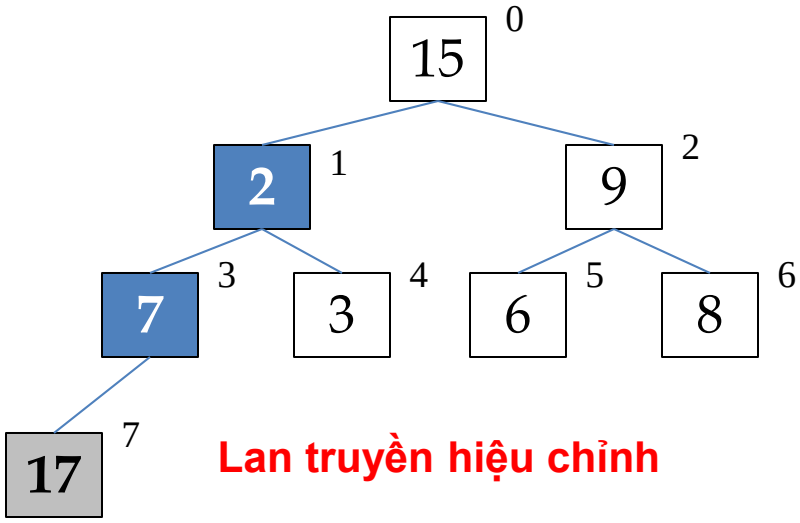
Ví dụ





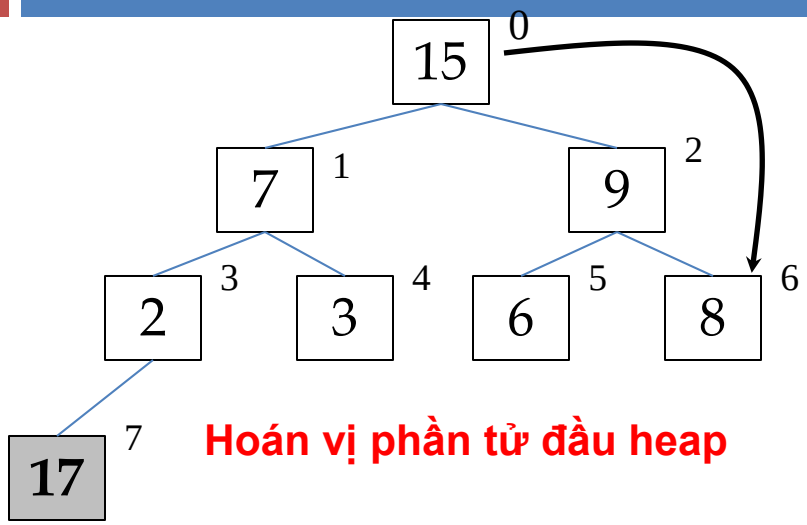
Hoán vị phần tử đầu heap



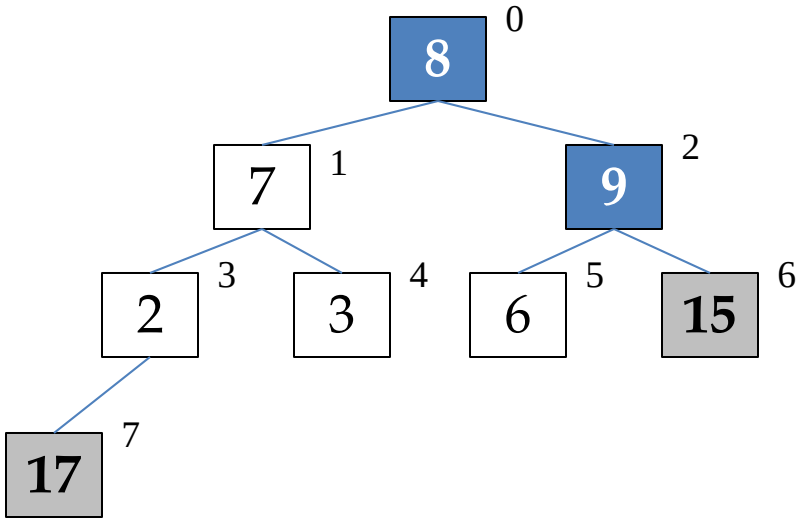


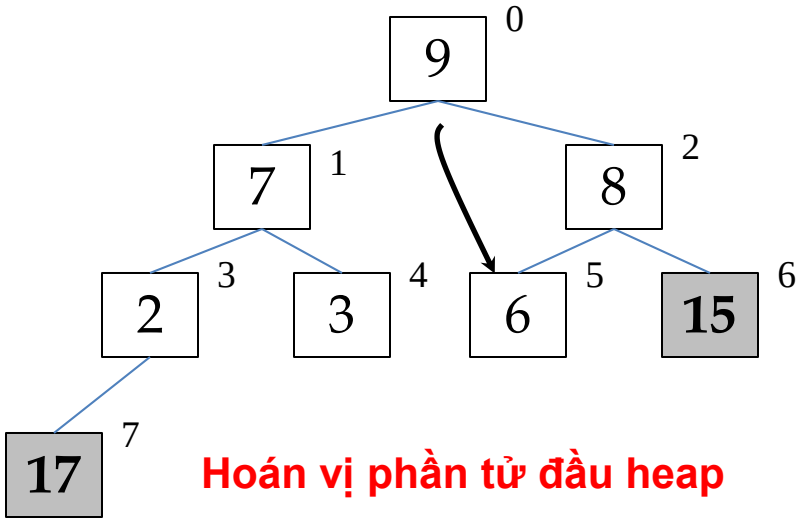
Lan truyền hiệu chỉnh

Ví dụ

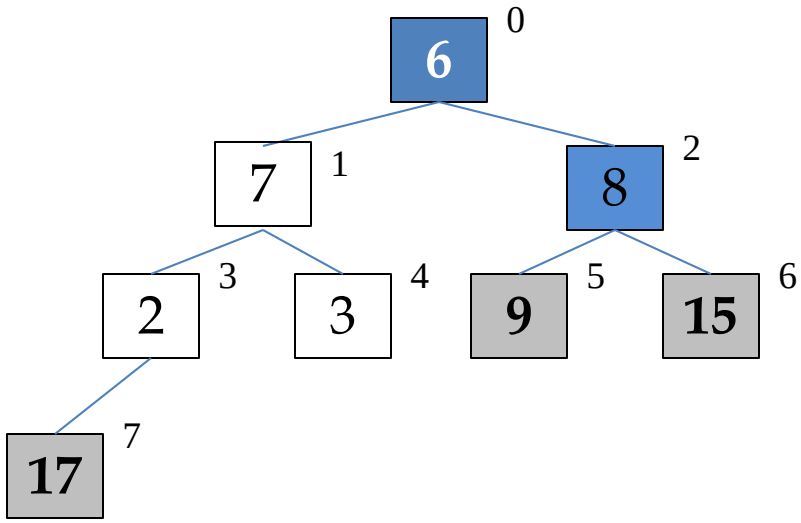


Hoán vị phần tử đầu heap

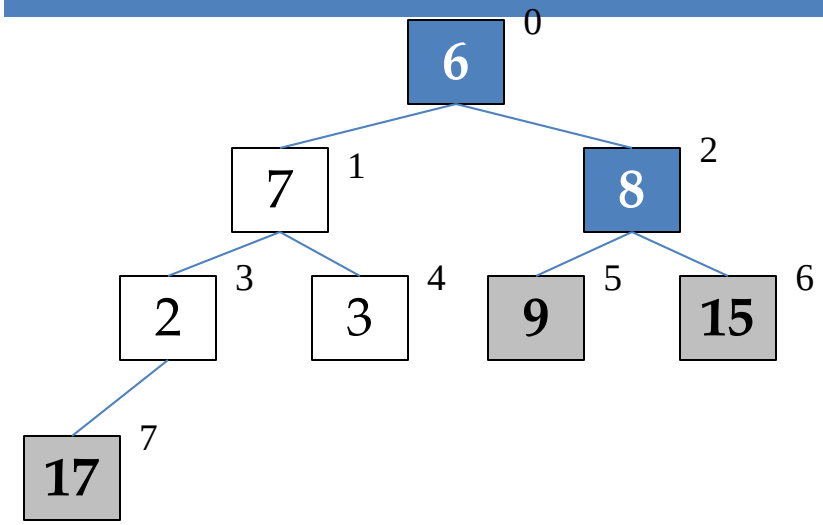


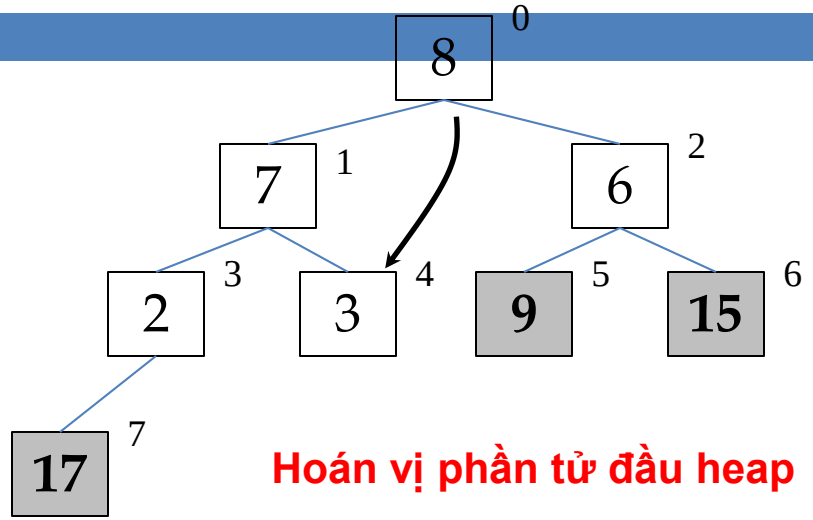


Hoán vị phần tử đầu heap

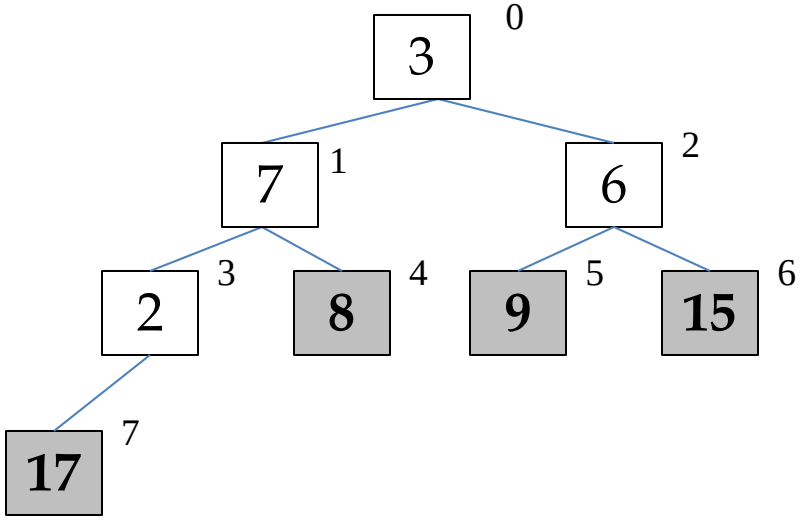


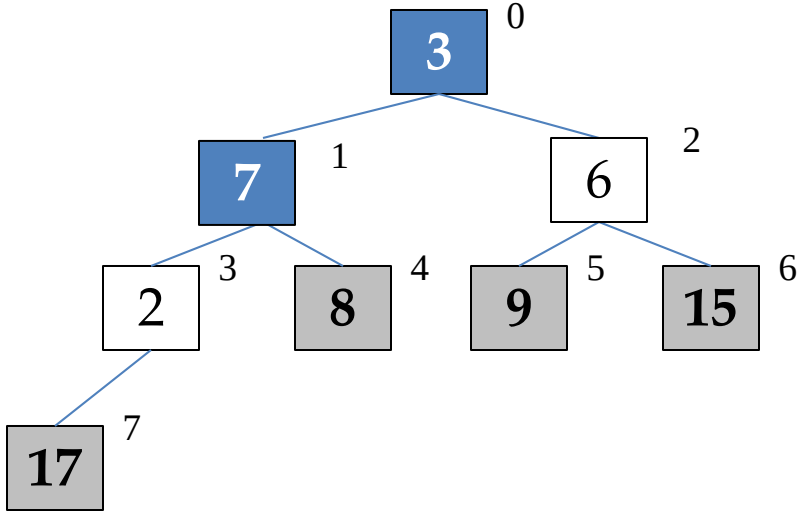
Ví dụ



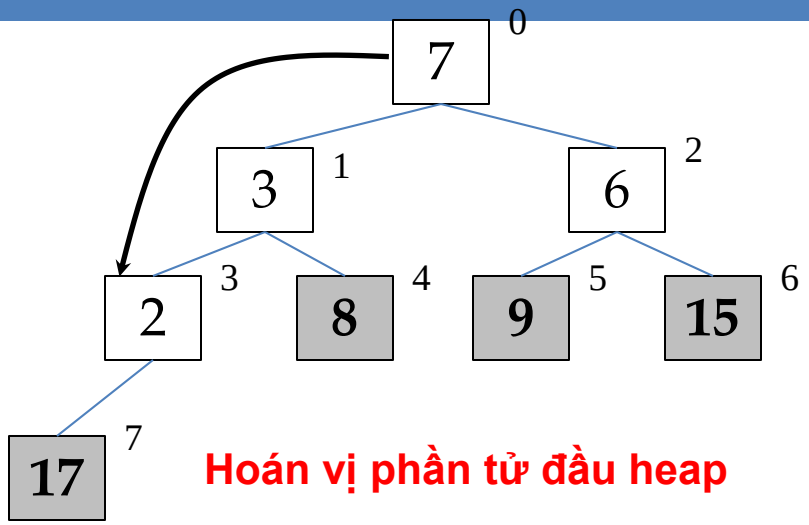


Hoán vị phần tử đầu heap

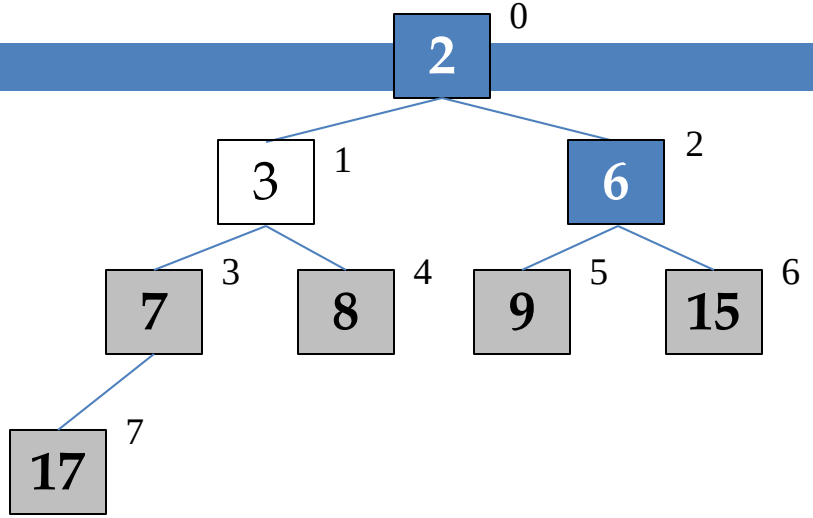


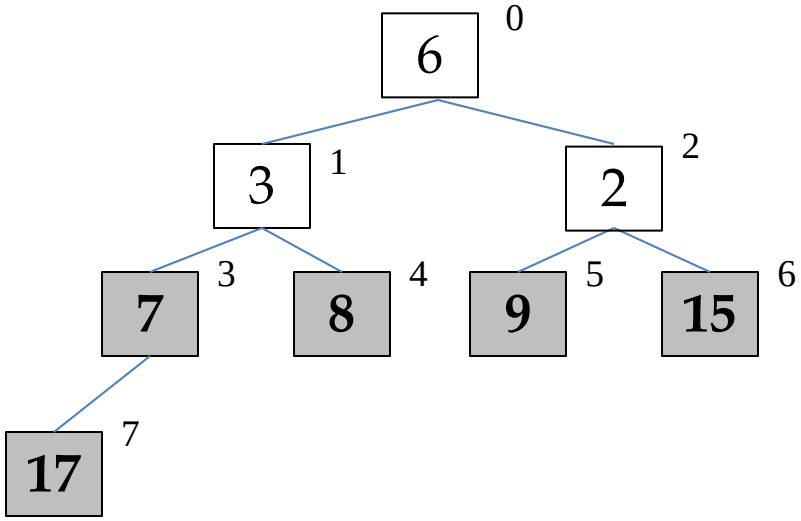


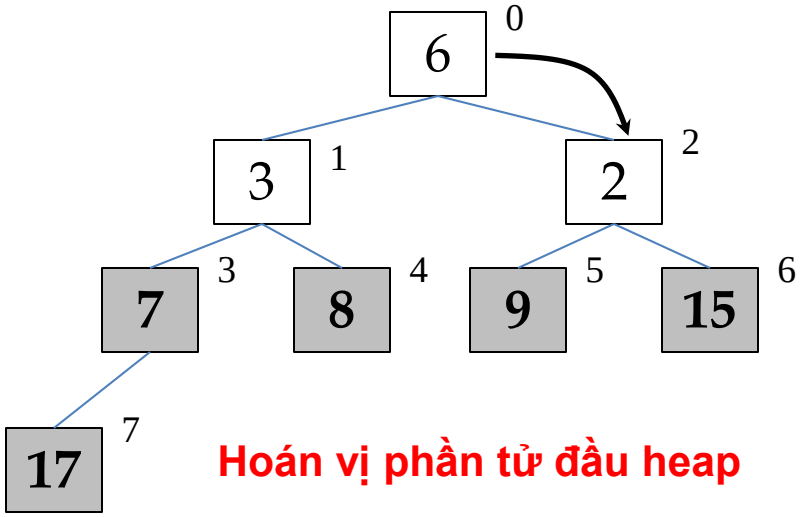
Ví dụ



Hoán vị phần tử đầu heap

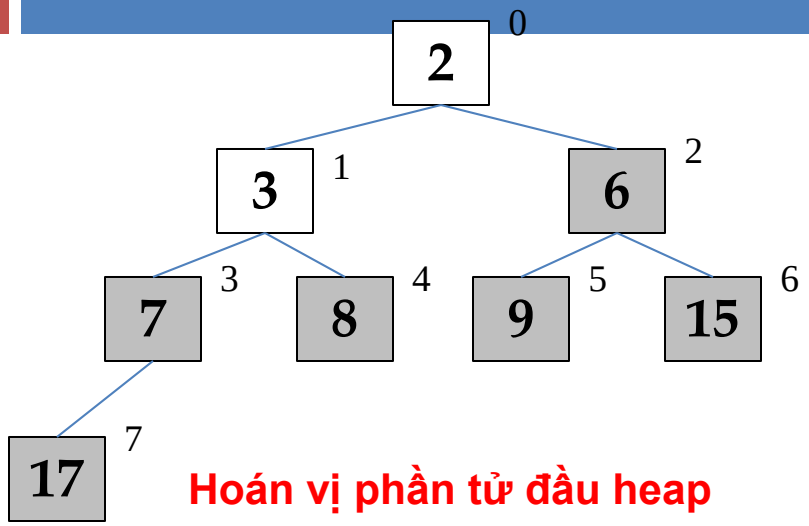




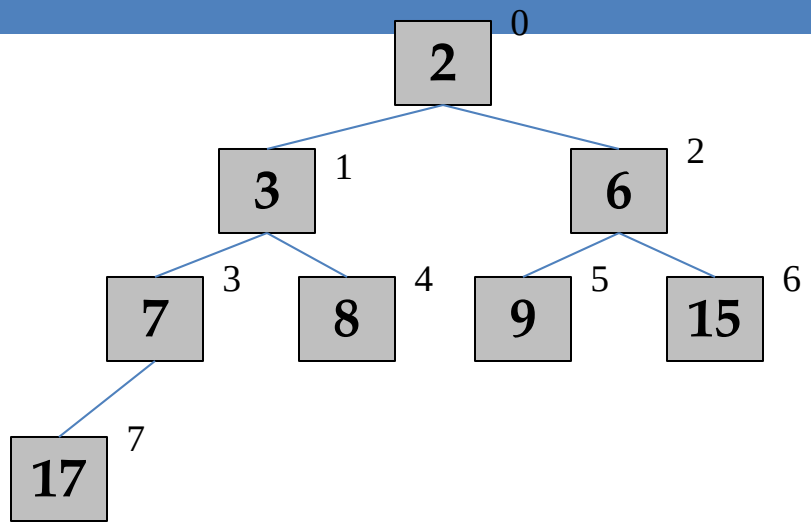


Hoán vị phần tử đầu heap

Ví dụ



Hoán vị phần tử đầu heap



Mảng sau khi sắp xếp:

2	3	6	7	8	9	15	17

Heap Sort

- Đánh giá giải thuật:
  - Độ phức tập của giải thuật trong trường hợp xấu nhất là $O(n\log_2 n)$

Sắp xếp trộn

Merge Sort

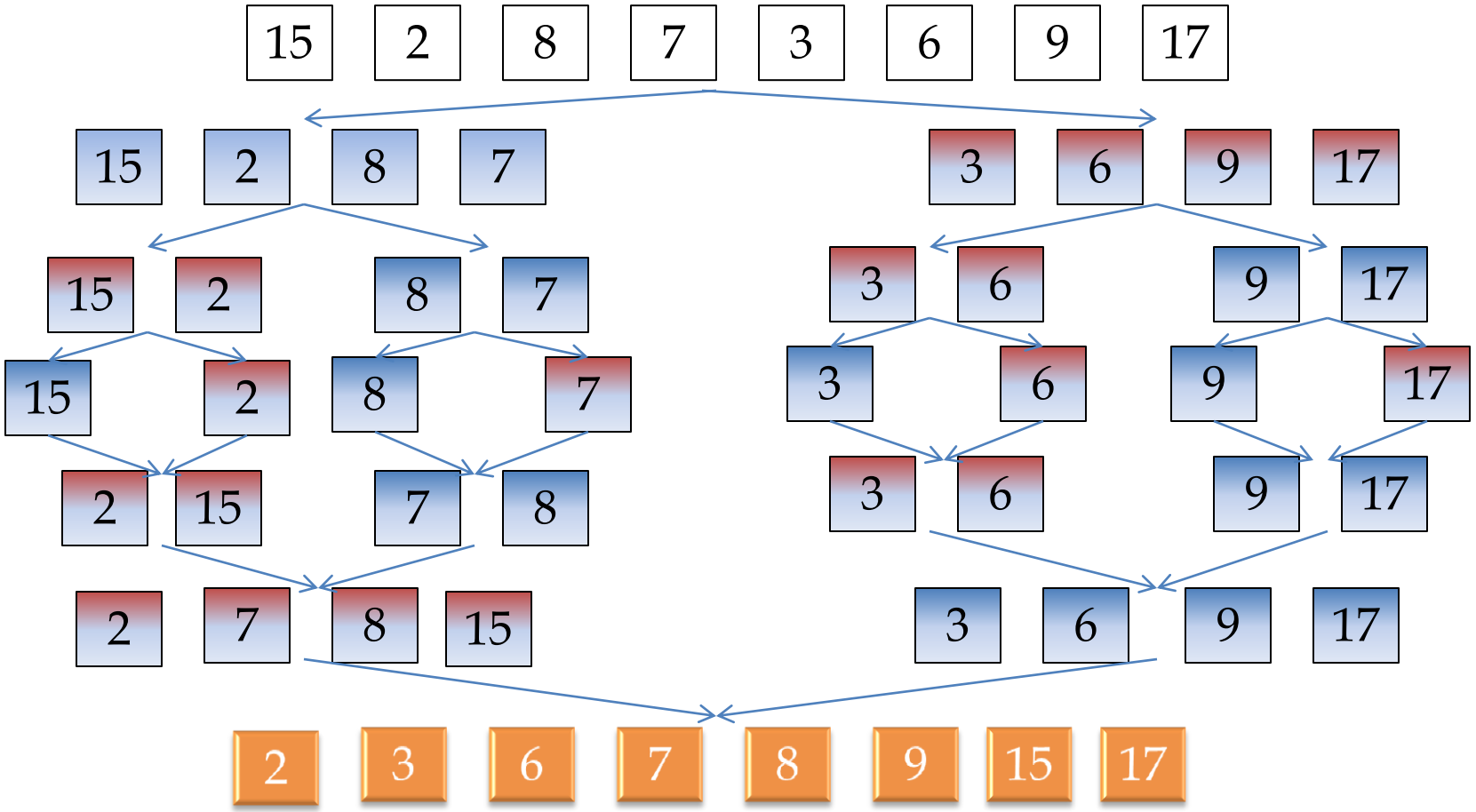
Giới thiệu

- Thực hiện theo hướng chia để trị.
- Do John von Neumann đề xuất năm 1945.

Giải thuật

- Nếu dãy có chiều dài là 0 hoặc 1: đã được sắp xếp.
- Ngược lại:
  - Chia dãy thành 2 dãy con (chiều dài tương đương nhau).
  - Sắp xếp trên từng dãy con bằng thuật toán Merge Sort.
  - Trộn 2 dãy con (đã được sắp xếp) thành một dãy mới đã được sắp xếp.

Ví dụ


15 2 8 7 3 6 9 17
15 2 8 7
3 6 9 17
15 2
8 7
3 6
9 17
15
2
8
7
3
6
9
17
2 15
7 8
3 6
9 17
2 7 8 15
3 6 9 17
2 3 6 7 8 9 15 17

Giải thuật

- Input: Dãy A và các chỉ số left, right (sắp xếp dãy A gồm các phần tử có chỉ số từ *left* đến *right*).
- Output: Dãy A đã được sắp xếp

```
MergeSort(A, left, right)
{
    if (left < right) {
        mid = (left + right)/2;
        MergeSort(A, left, mid);
        MergeSort(A, mid+1, right);
        Merge(A, left, mid, right);
    }
}
```

Thao tác trộn


```
void Merge(int a[], int Left, int Mid, int Right)
{
    // c1, c2: vị trí hiện tại trên dãy con trái, dãy con phải
    // d: vị trí hiện tại trên dãy tạm
    for(int d=Left, int c1=Left, c2=Mid+1; (c1 <= Mid) && (c2 <= Right); d++ )
    {
        if (a[c1] < a[c2]) {          // lấy phần tử trên dãy con trái
            TempArray[d] = a[c1];      c1++;
        }
        else {    // lấy phần tử trên dãy con phải
            TempArray[d] = a[c2];      c2++;
        } // end if
    } // end for
```

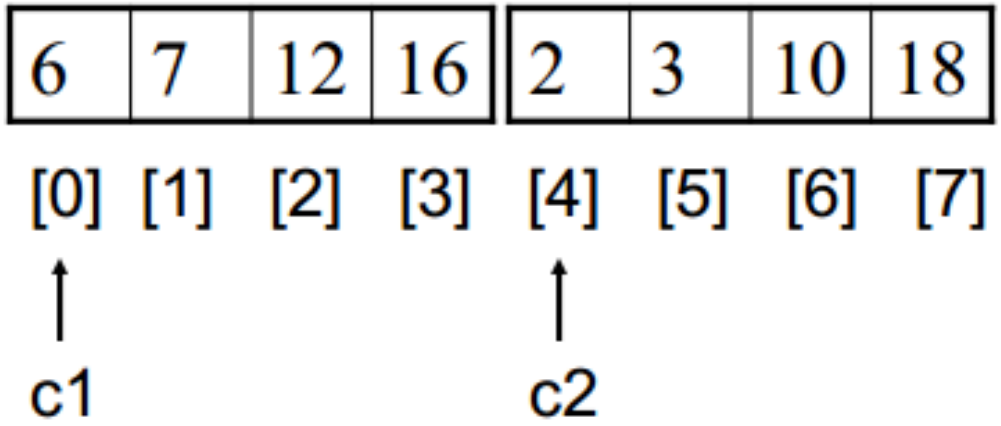
tiếp tục...

Thao tác trộn

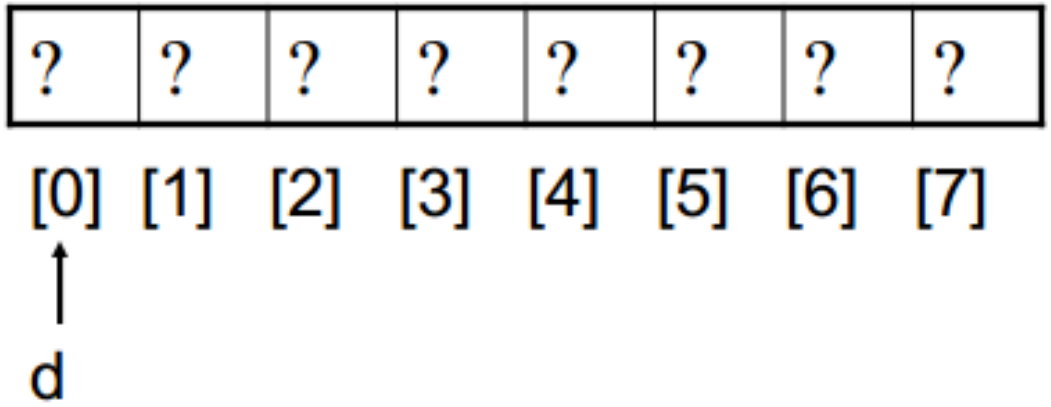

```
    // Khi 1 trong 2 dãy đã hết phần tử…
    // nếu dãy bên trái còn dư → chép vào mảng tạm
    for( ; c1 <= Mid; c1++, d++ )    TempArray[d] = a[c1];
    // nếu dãy bên phải còn dư → chép vào mảng tạm
    for( ; c2 <= Right; c2++, d++ )   TempArray[d] = a[c2];
     // Sau khi trộn, copy mảng tạm trở lại mảng gốc
    for (d=Left; d<=Right; d++) {
            a[d] = TempArray[d];
    }
} // end of Merge
```

Thao tác trộn (ví dụ)

Hai dãy con đã sắp

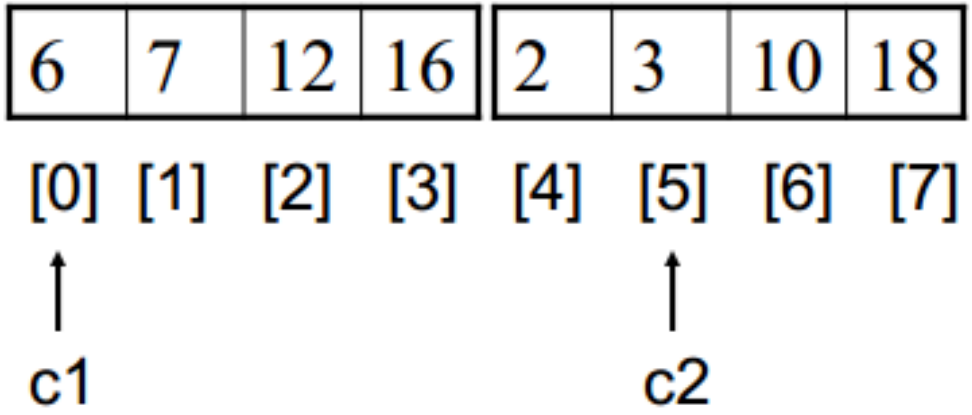


Bảng tạm

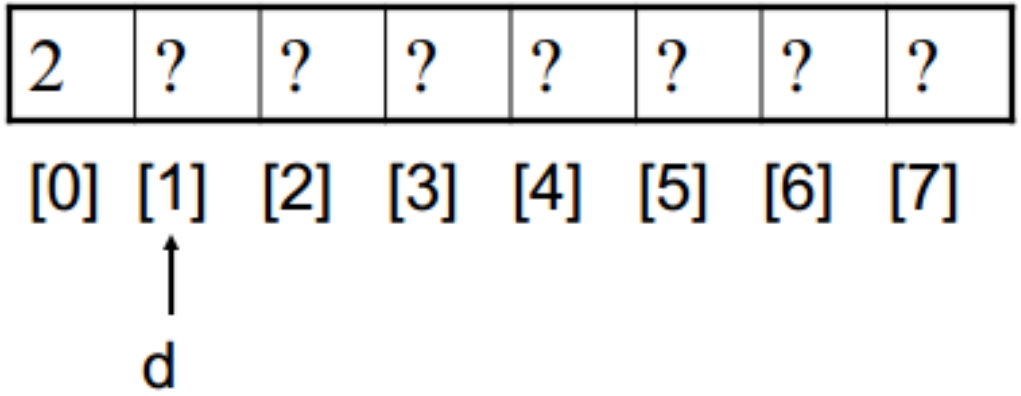


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

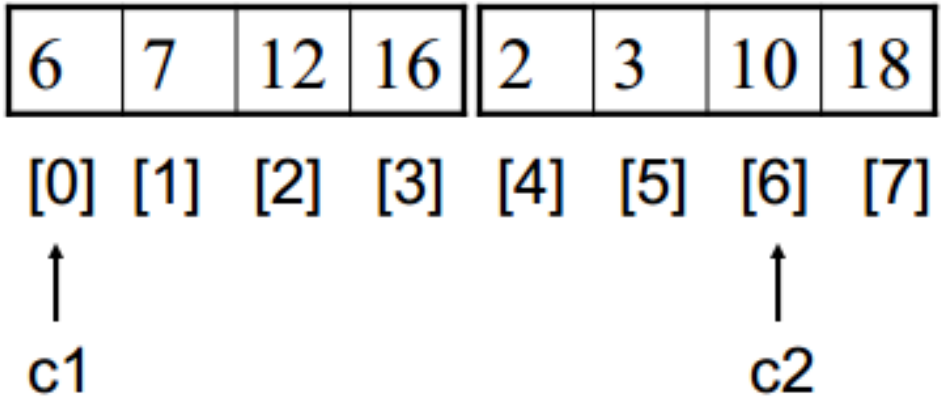


Bảng tạm

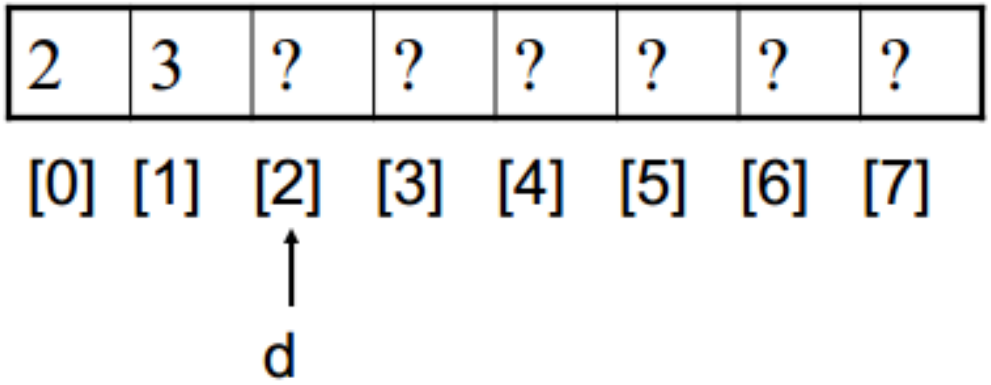


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

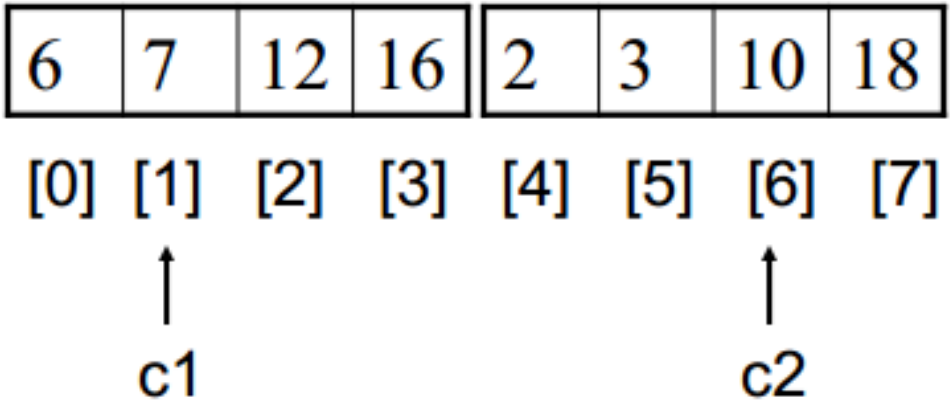


Bảng tạm

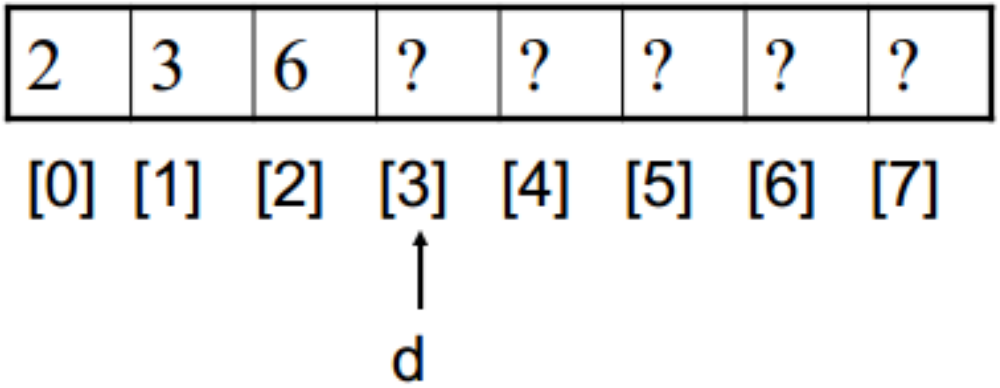


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

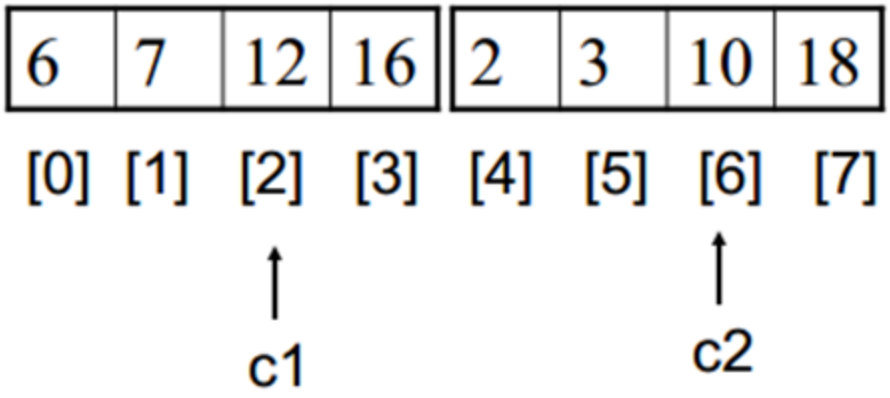


Bảng tạm

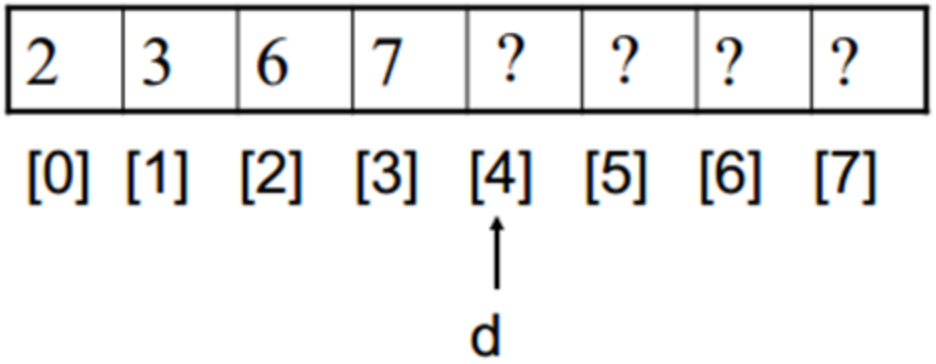


Thao tác trộn (ví dụ)

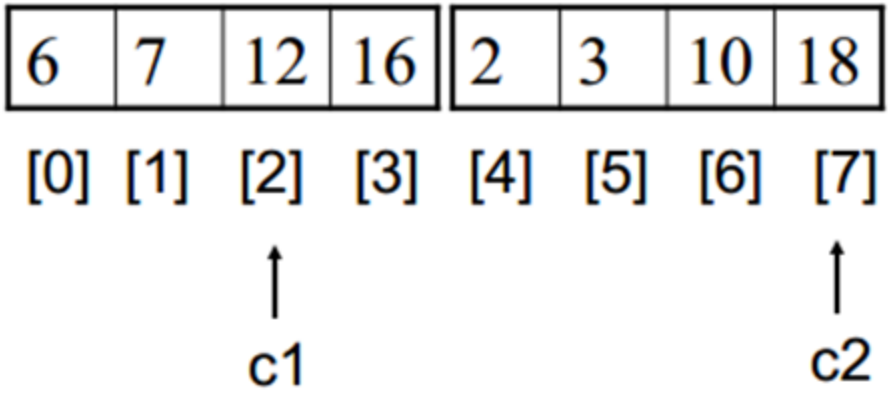
Hai dãy con đã sắp



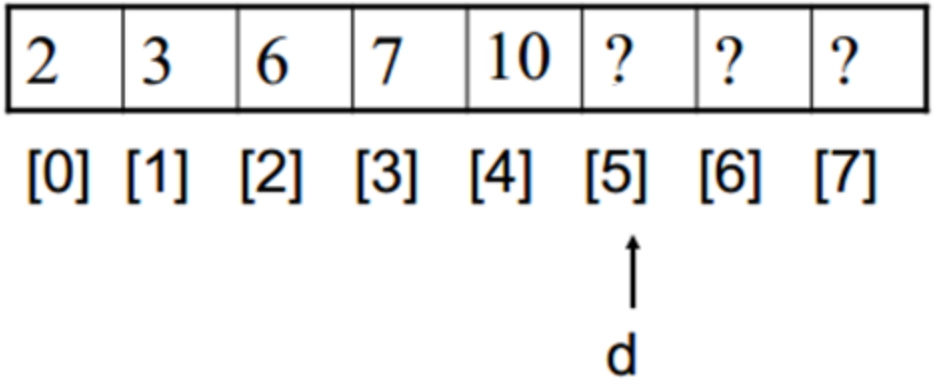
Bảng tạm



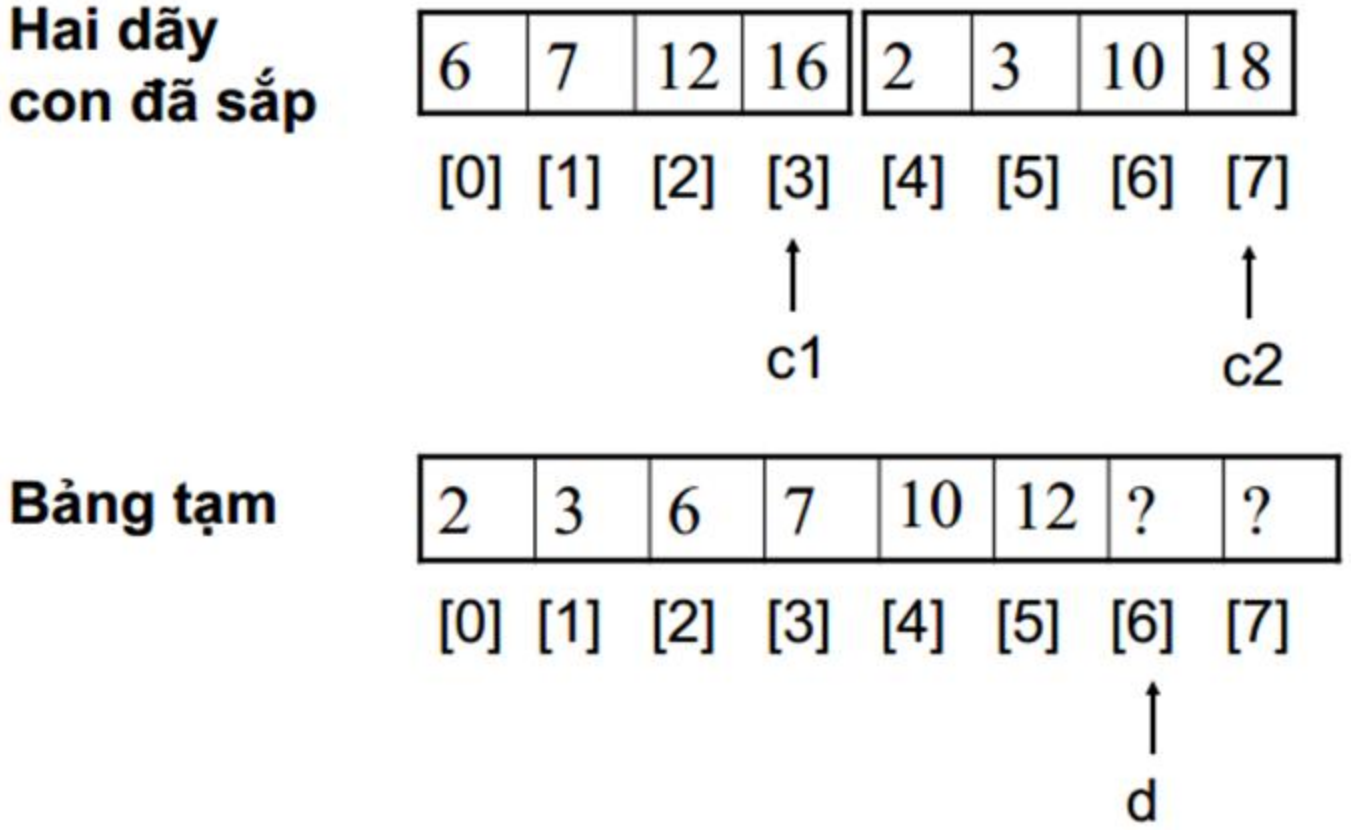
Hai dãy con đã sắp



Bảng tạm

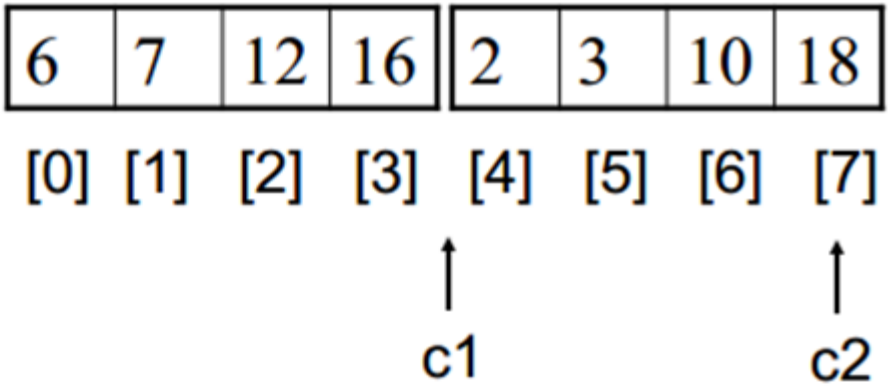


Thao tác trộn (ví dụ)

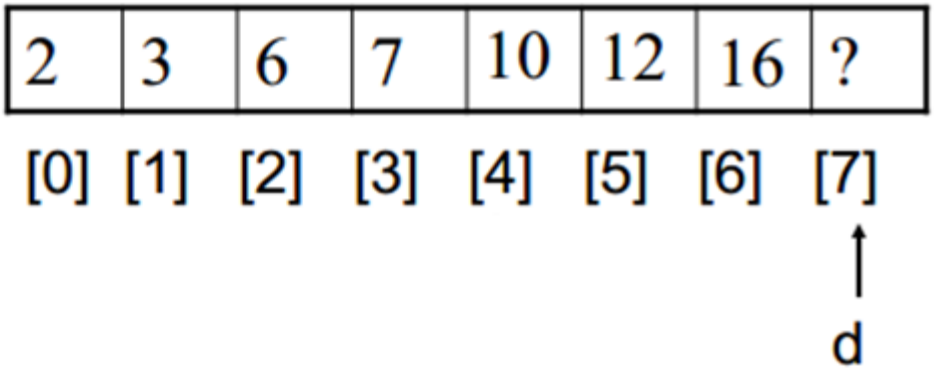


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

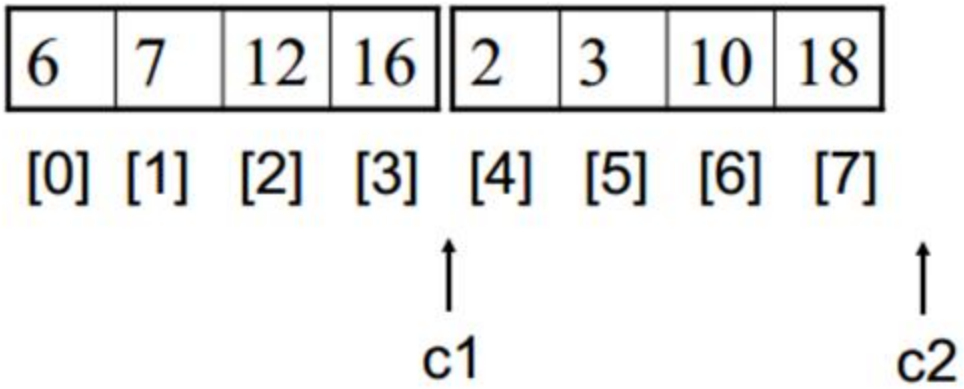


Bảng tạm

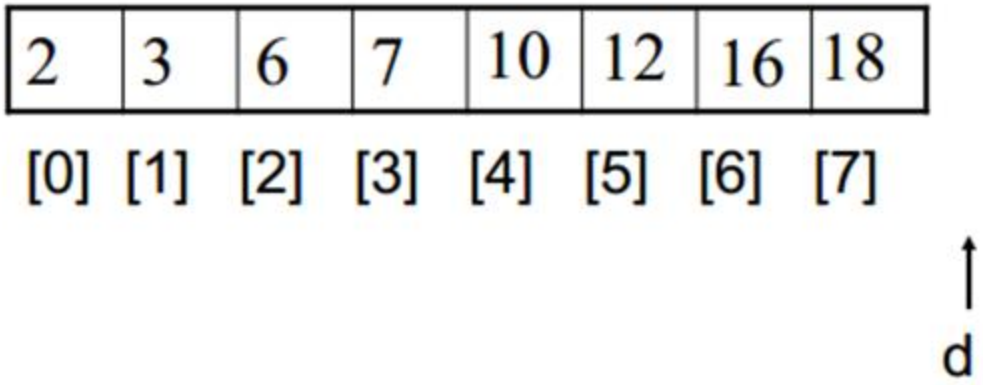


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

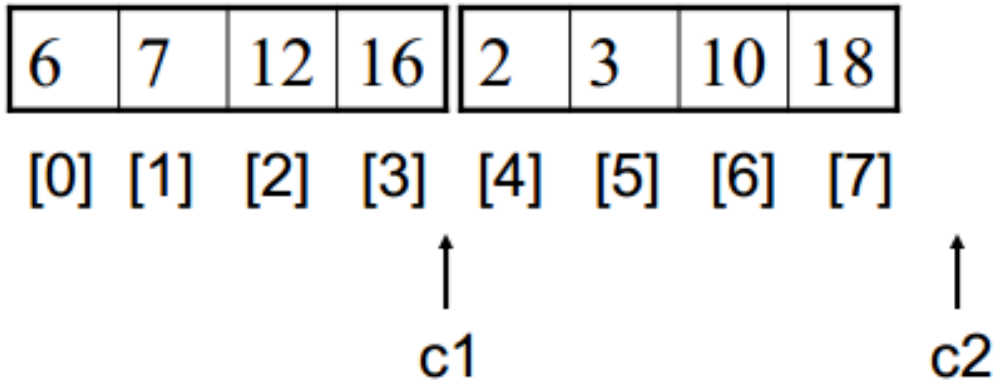


Bảng tạm

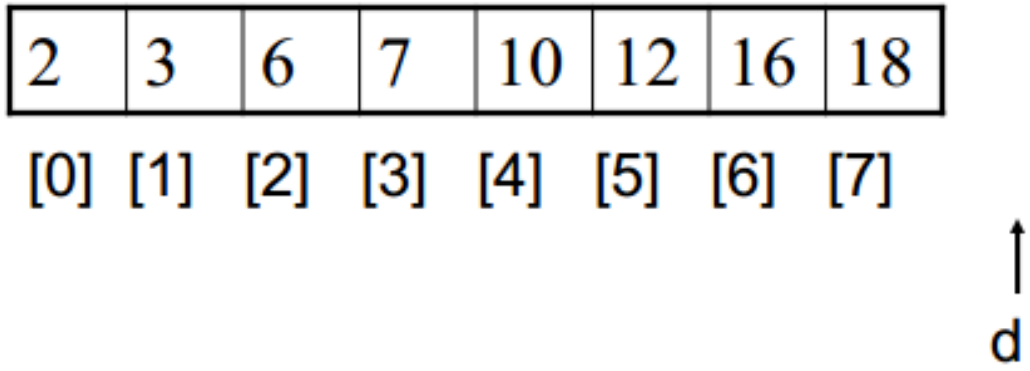


Thao tác trộn (ví dụ)

Hai dãy con đã sắp

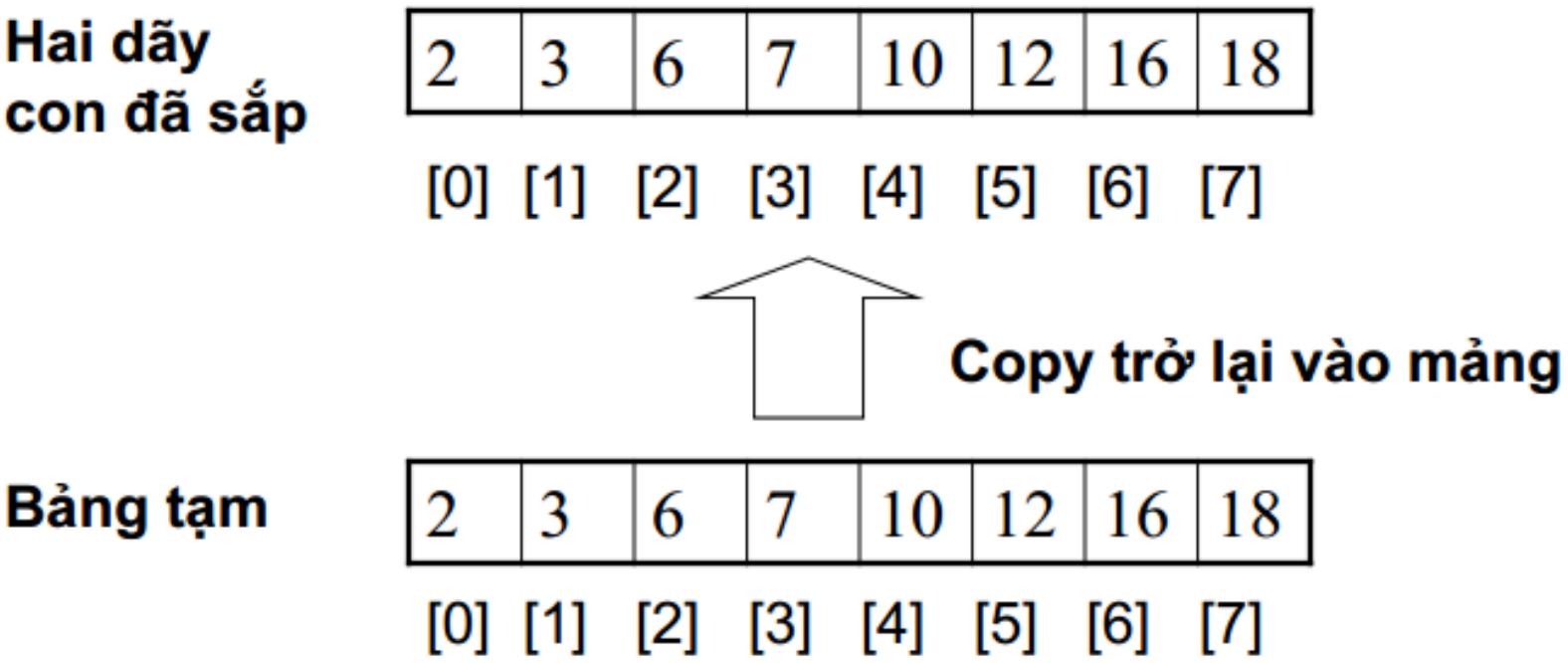


Bảng tạm



Hoàn tất !

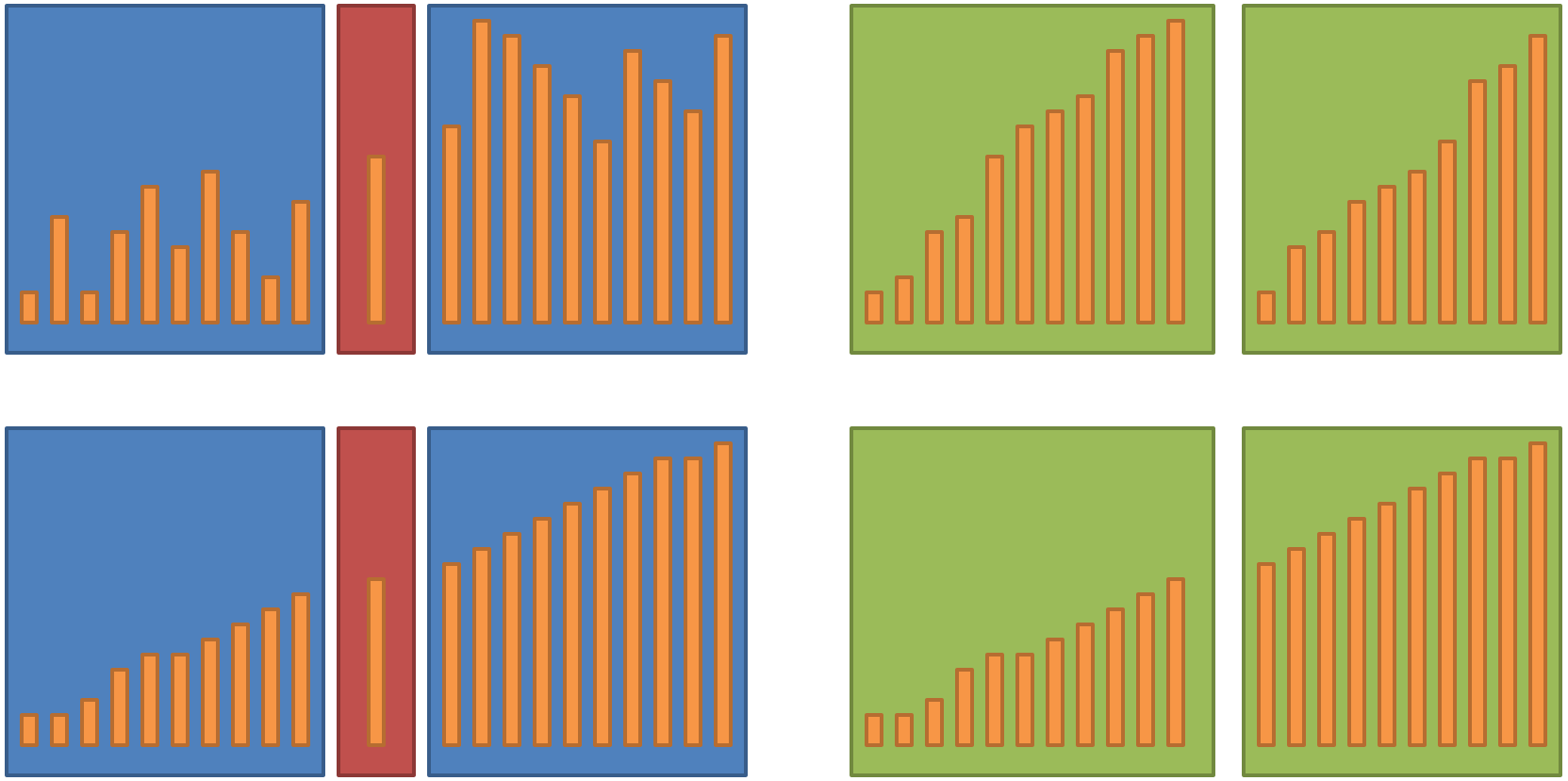
Thao tác trộn (ví dụ)



Đánh giá

- Số lần chia các dãy con: $\log_2 n$
- Chi phí thực hiện việc trộn hai dãy con đã sắp xếp tỷ lệ thuận với n.
- Chi phí của Merge Sort là $O(n\log_2 n)$
- Thuật toán không sử dụng thông tin nào về đặc tính của dãy cần sắp xếp => chi phí thuật toán là không đổi trong mọi trường hợp

So sánh tư tưởng sắp xếp giữa Quick sort và Merge sort



Kết luận

Kết luận

- Các thuật toán Bubble sort, Selection sort, Insertion sort
  - Cài đặt thuật toán đơn giản.
  - Chi phí của thuật toán cao: $O(n^2)$.
- Heap sort được cải tiến từ Selection sort nhưng chi phí thuật toán thấp hơn hẳn ($O(n\log_2 n)$)

Kết luận

- Các thuật toán Quick sort, Merge sort là những thuật toán theo chiến lược chia để trị.
  - Cài đặt thuật toán phức tạp
  - Chi phí thuật toán thấp: $O(n\log_2 n)$
  - Rất hiệu quả khi dùng danh sách liên kết.
  - Trong thực tế, Quick sort chạy nhanh hơn hẳn Merge sort và Heap sort.

Kết luận

- Người ta chứng minh $O(n\log_2 n)$ là ngưỡng chặn dưới của các thuật toán sắp xếp dựa trên việc so sánh giá trị của các phần tử.

Hỏi và Đáp