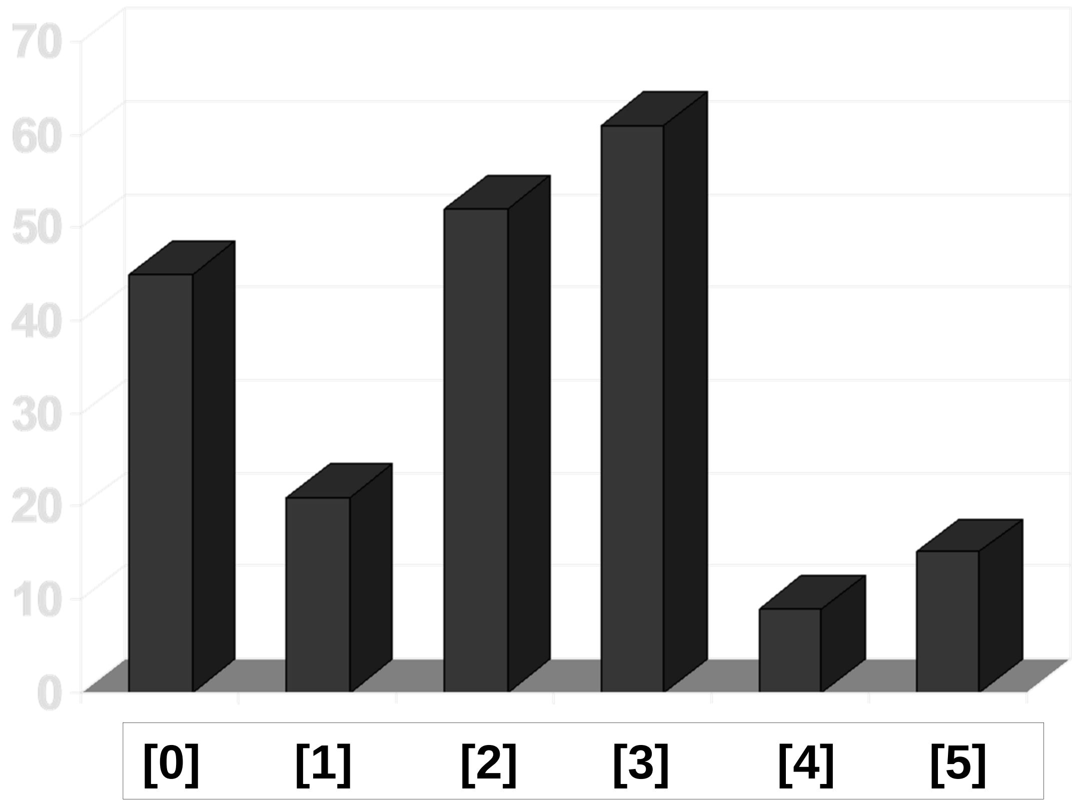
Các thuật toán sắp xếp
(Sorting algorithms)

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@fit.hcmus.edu.vn

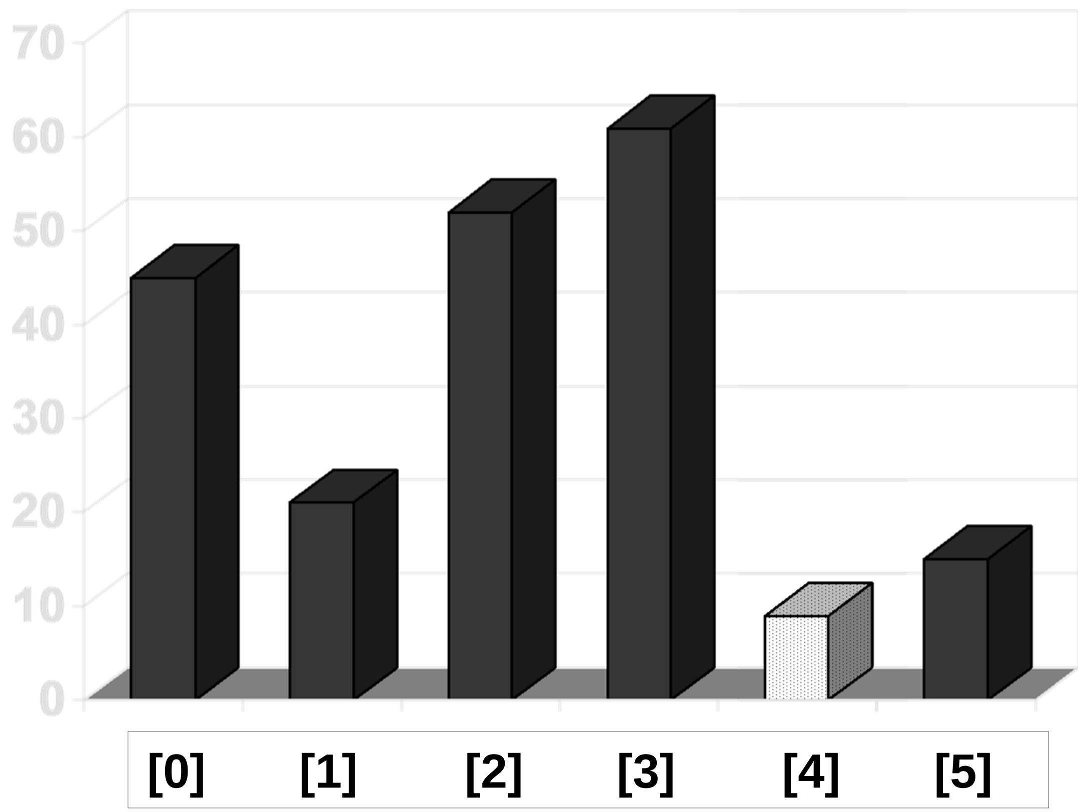
Sắp xếp 1 mảng các số nguyên

- Giả sử có 1 mảng gồm 6 số nguyên. Ta cần sắp xếp các phần tử của mảng theo thứ tự tăng dần



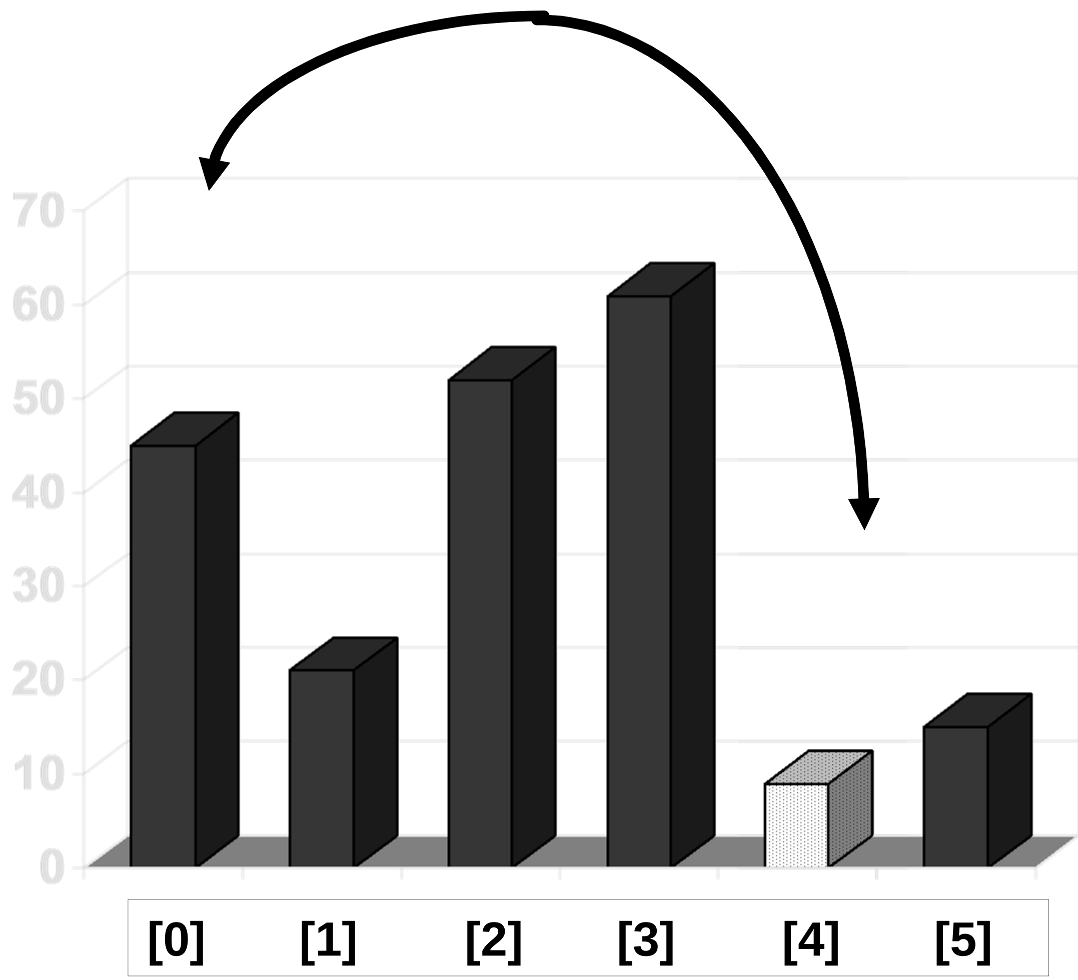
Thuật toán "Chọn trực tiếp" (Selection sort Algorithm)

- Bắt đầu bằng cách tìm **<u>phần tử nhỏ nhất</u>**



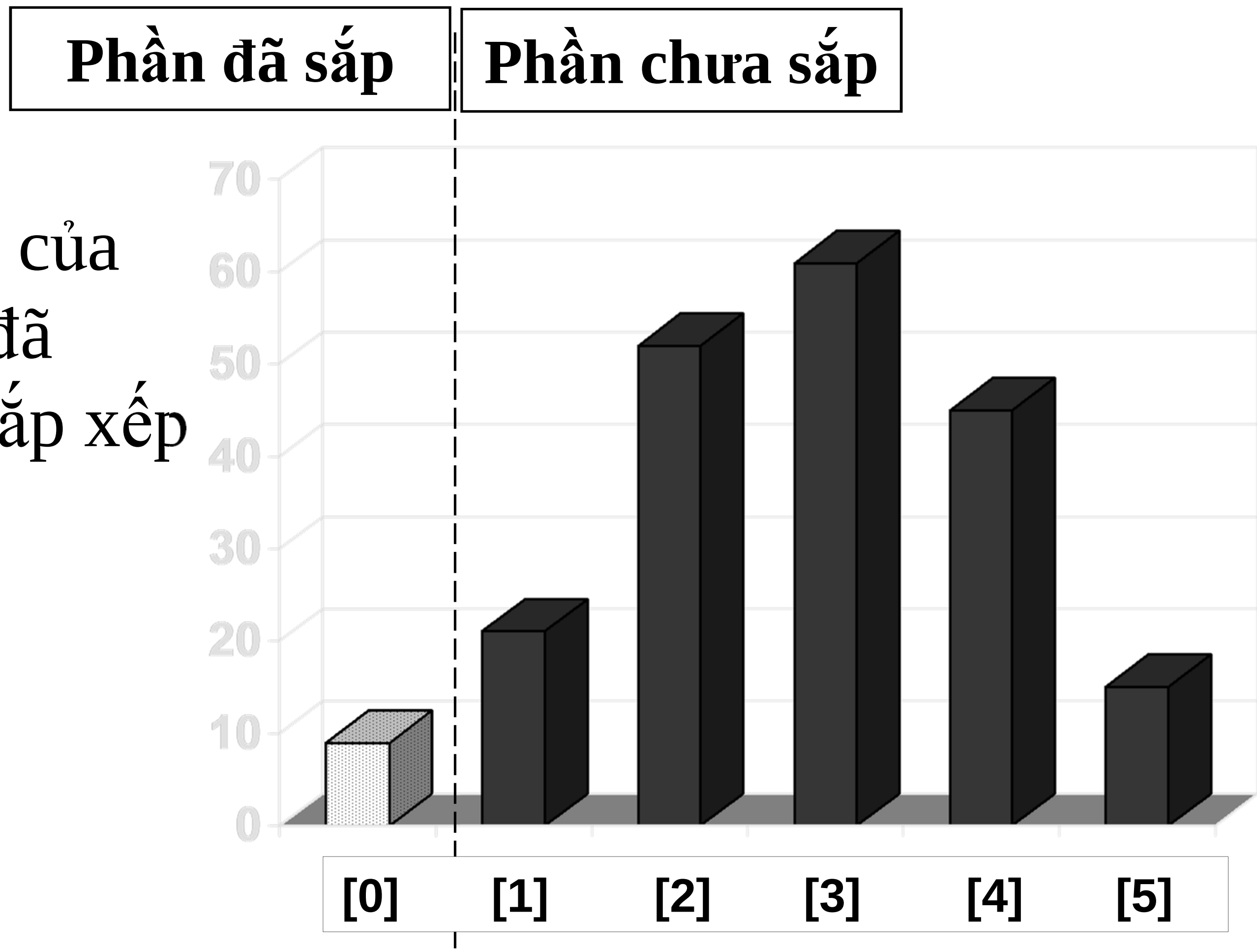
Selection sort Algorithm

- Hoán vị **phần tử nhỏ nhất** tìm được với **phần tử đầu tiên** của mảng



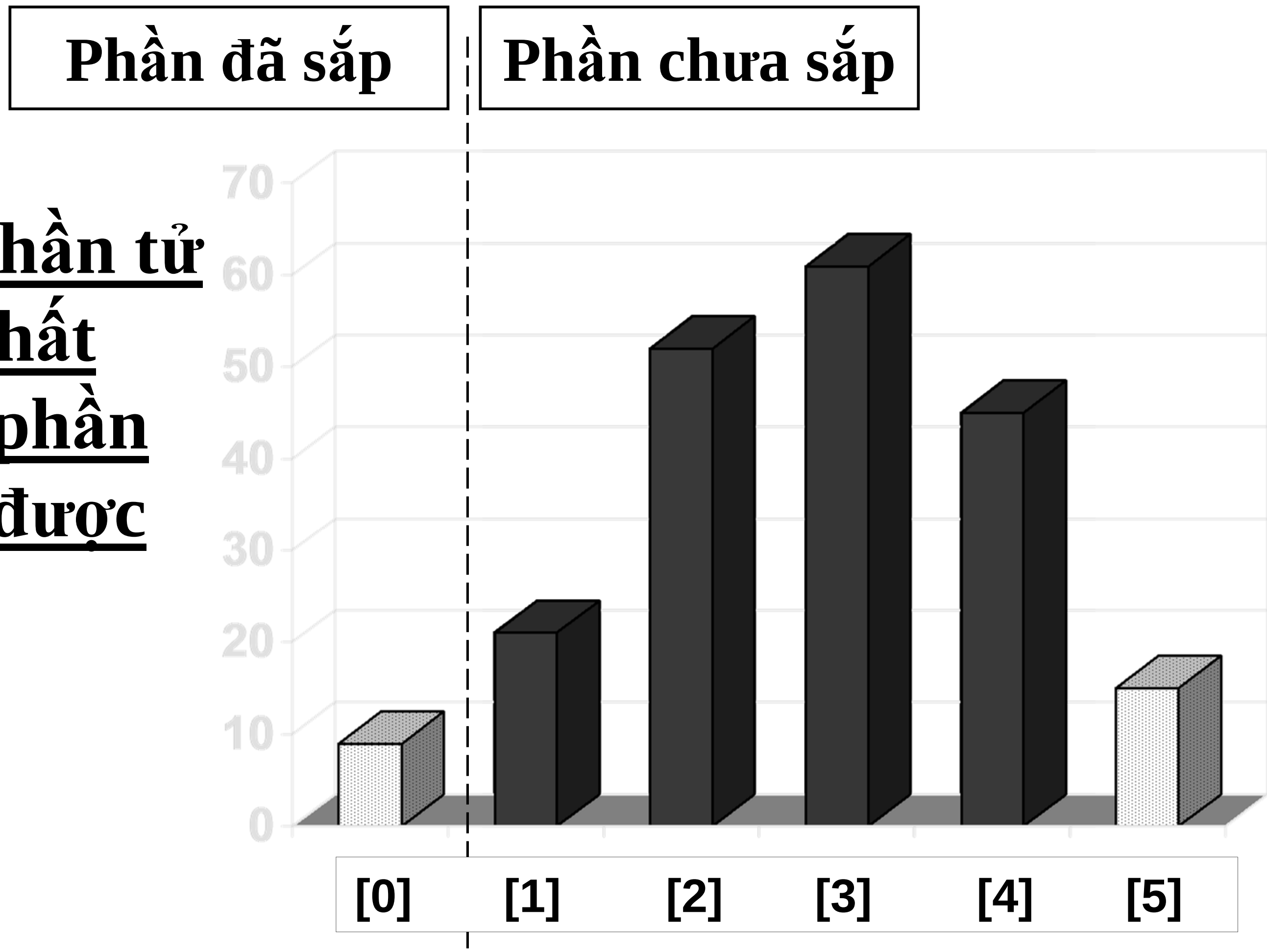
Selection sort Algorithm

- 1 phần của mảng đã được sắp xếp



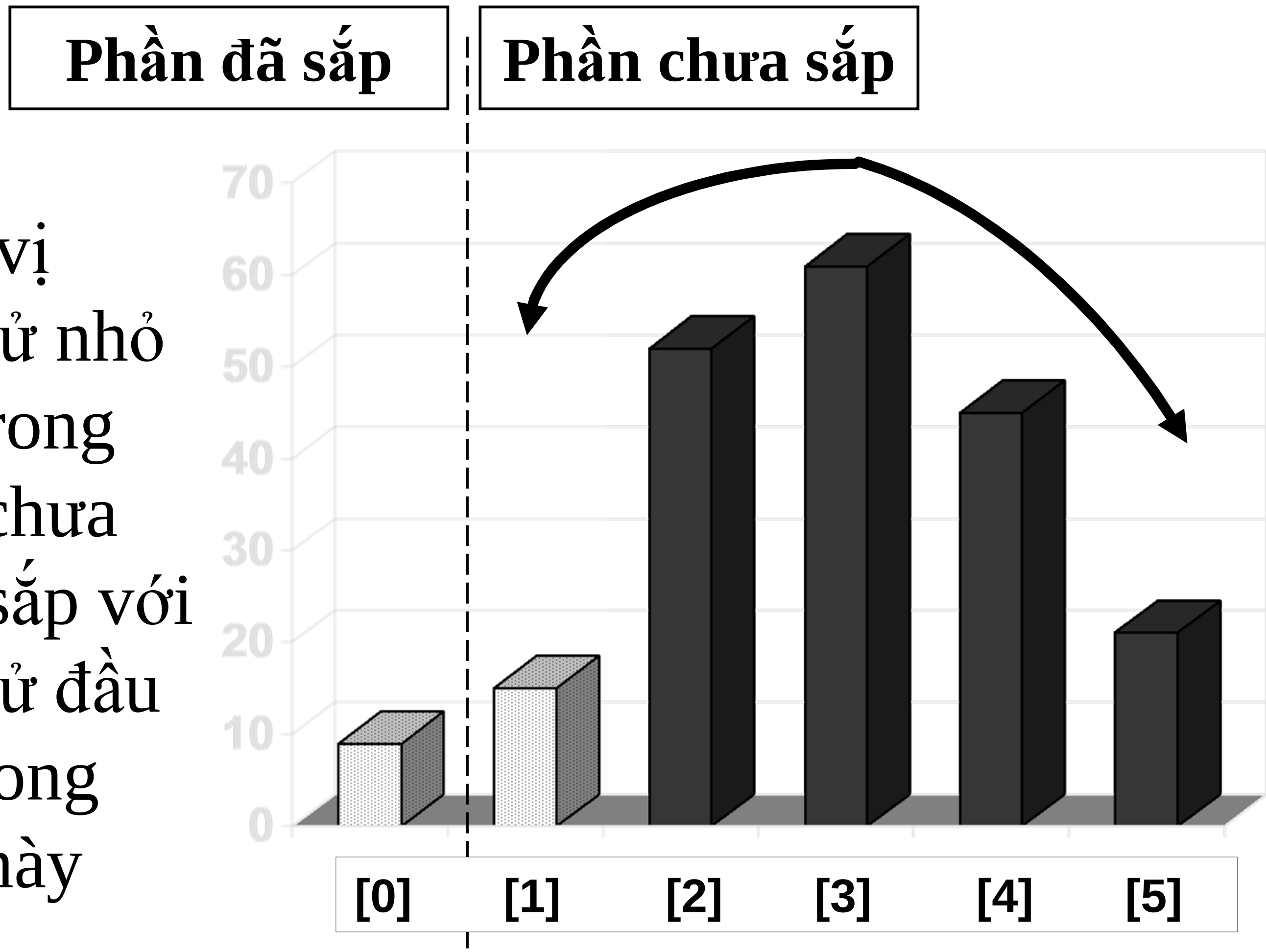
Selection sort Algorithm

- Tìm **phần tử nhỏ nhất** trong **phần chưa được sắp**

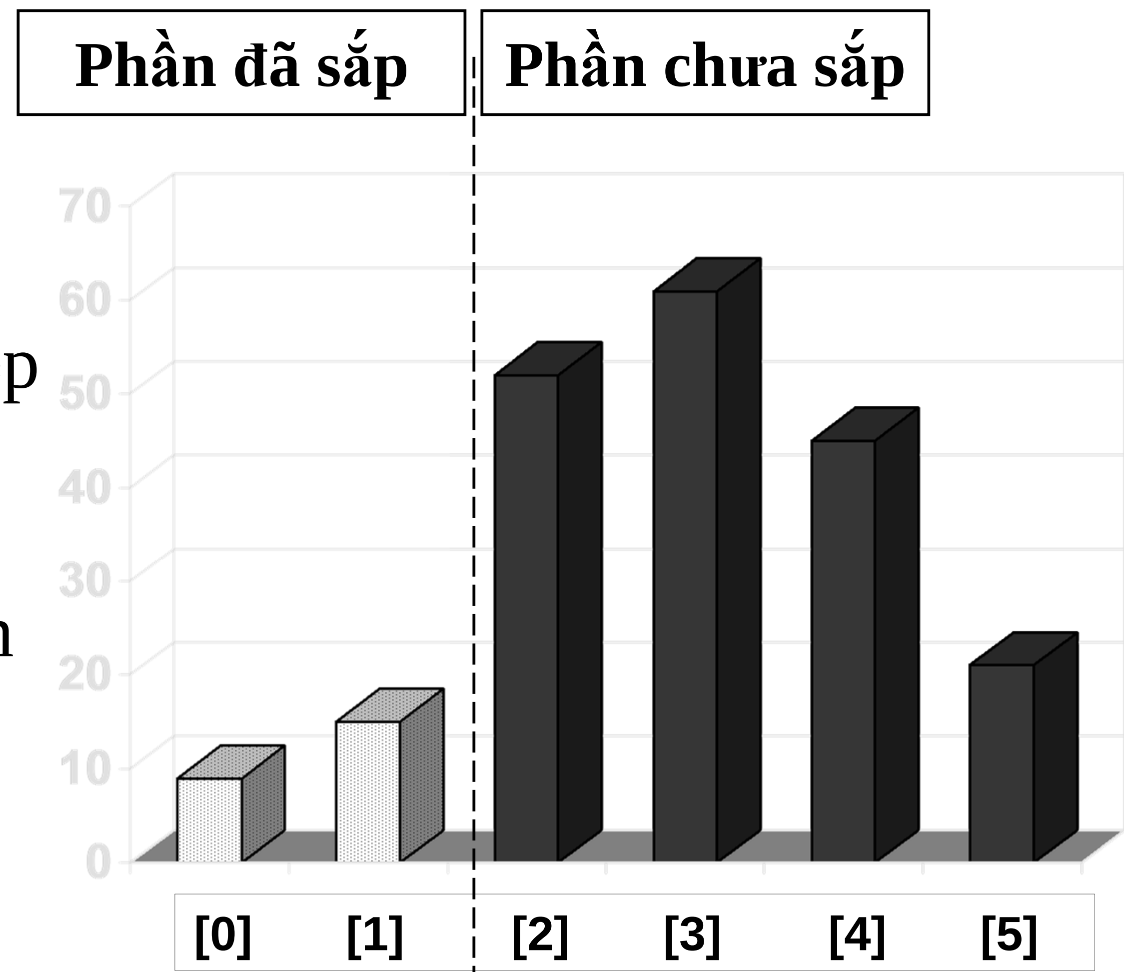


Selection sort Algorithm

- Hoán vị phần tử nhỏ nhất trong phần chưa được sắp với phần tử đầu tiên trong phần này



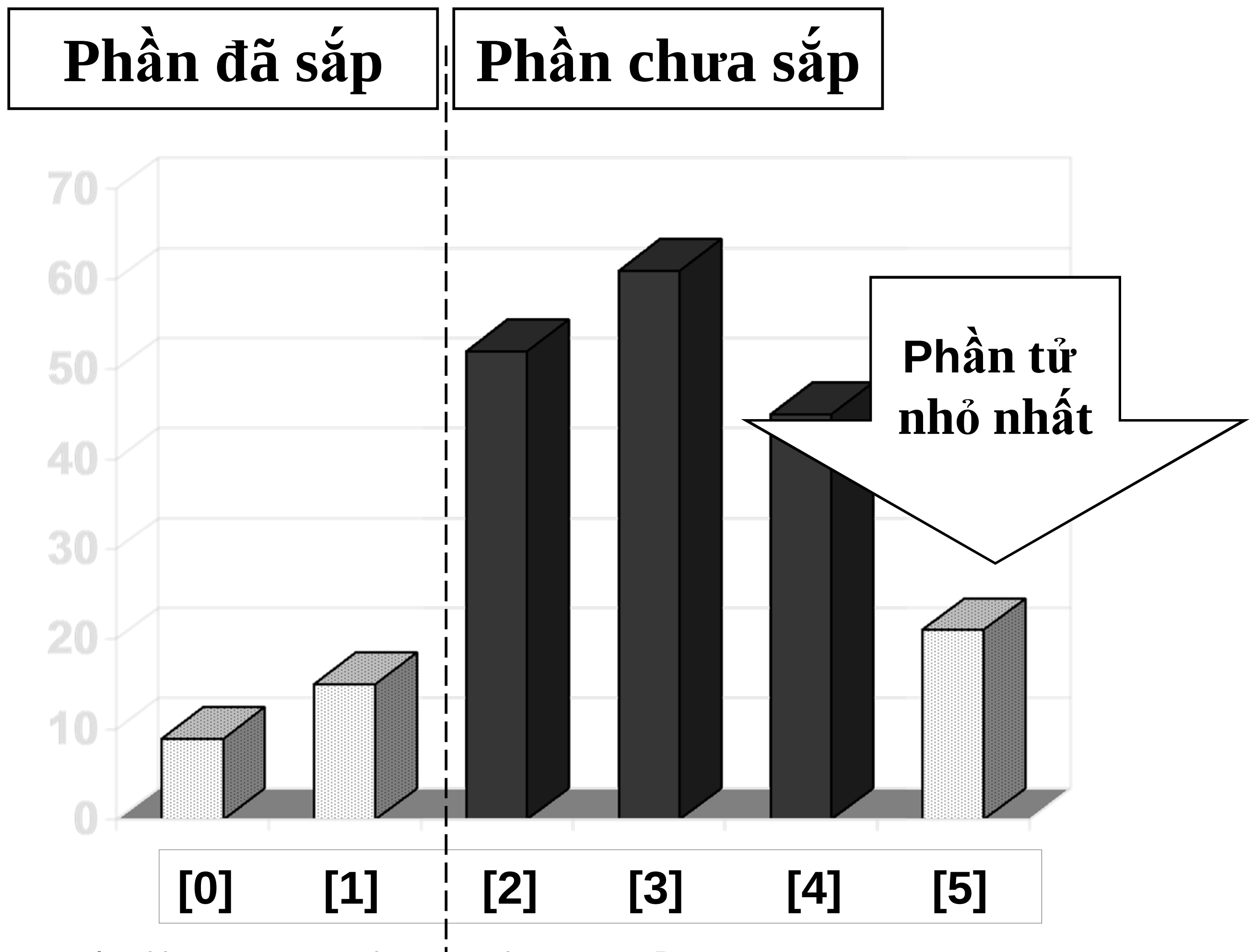
Selection sort Algorithm



- Phần đã được sắp xếp của mảng được tăng thêm 1 phần tử

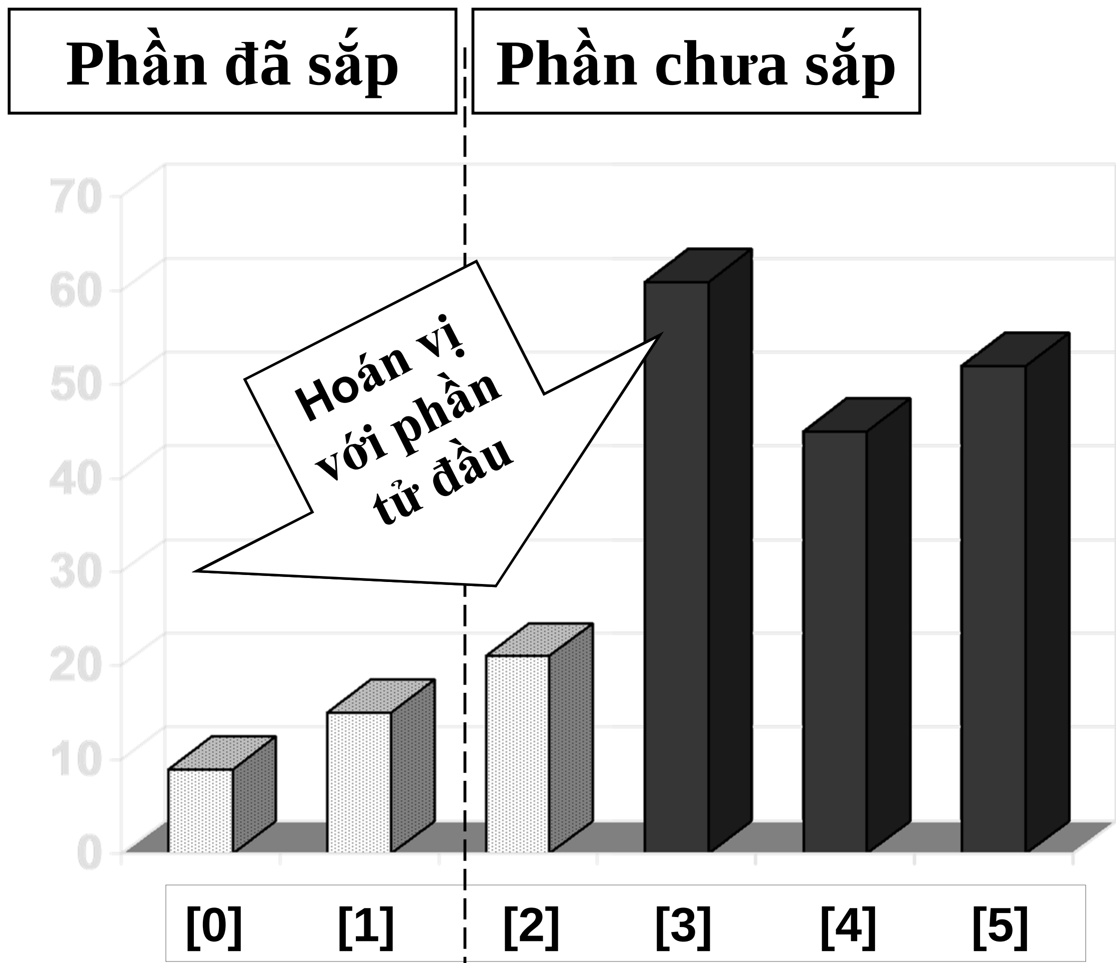
Selection sort Algorithm

- Tiếp tục tương tự ...



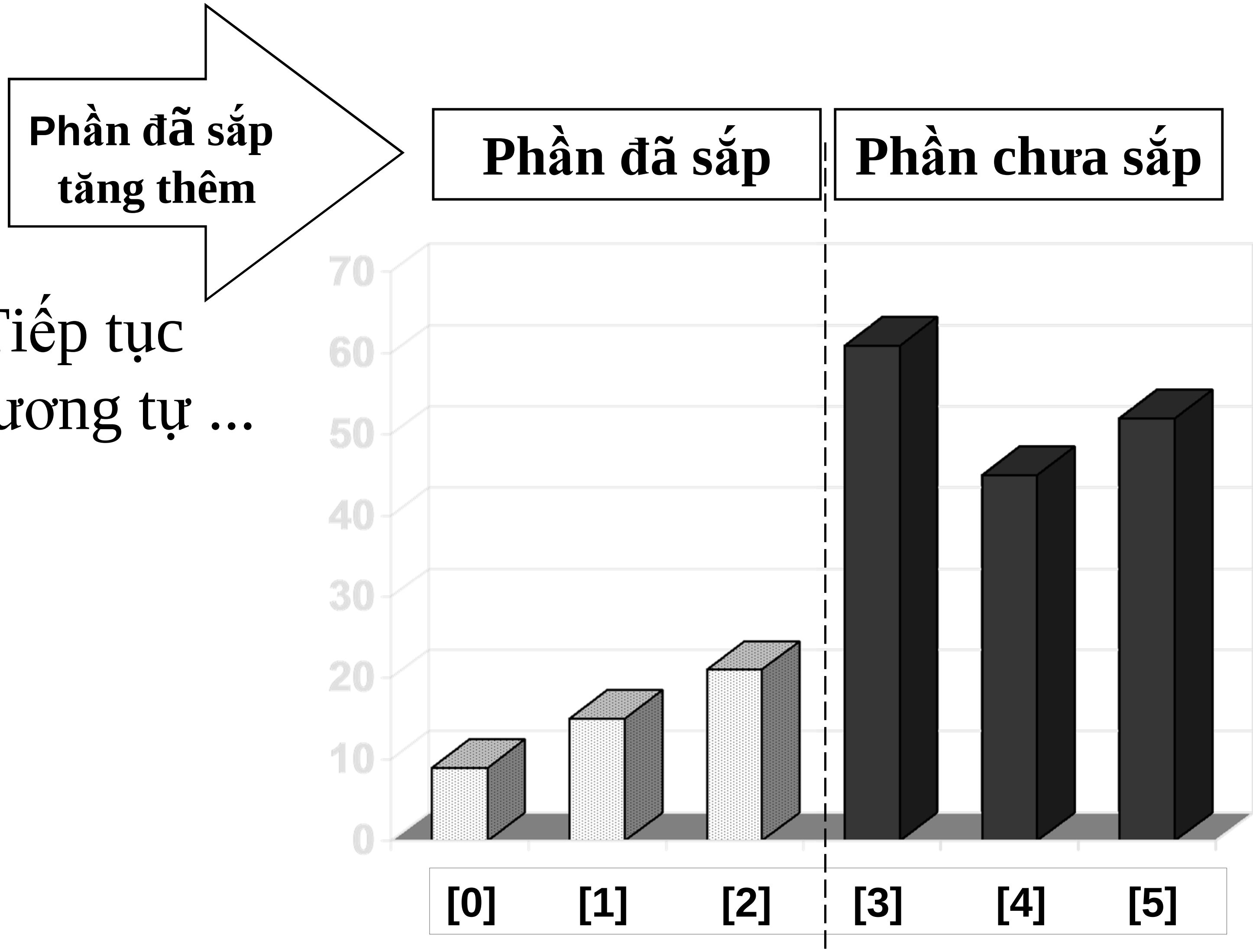
Selection sort Algorithm

- Tiếp tục tương tự ...



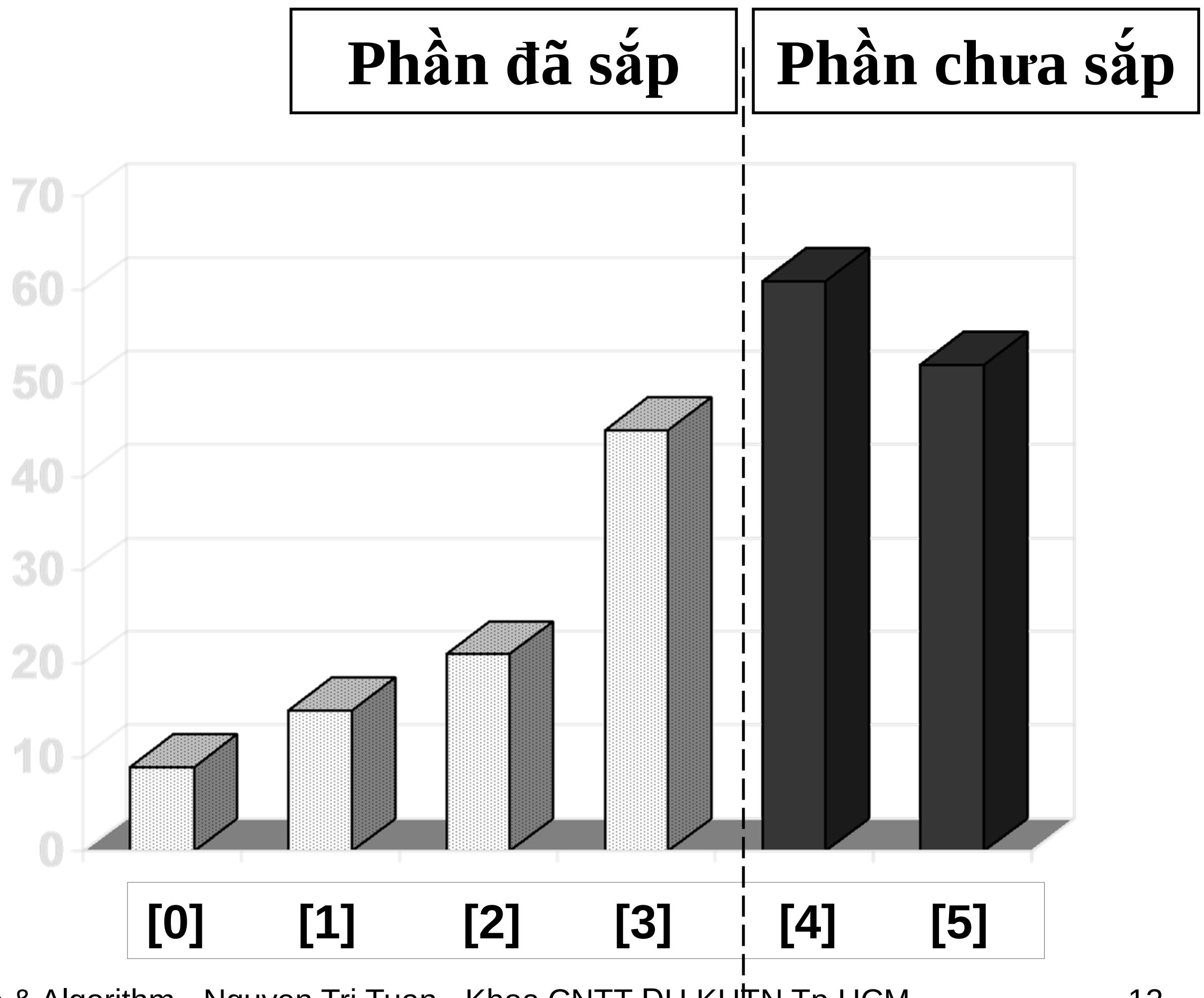
Selection sort Algorithm

- Tiếp tục tương tự ...



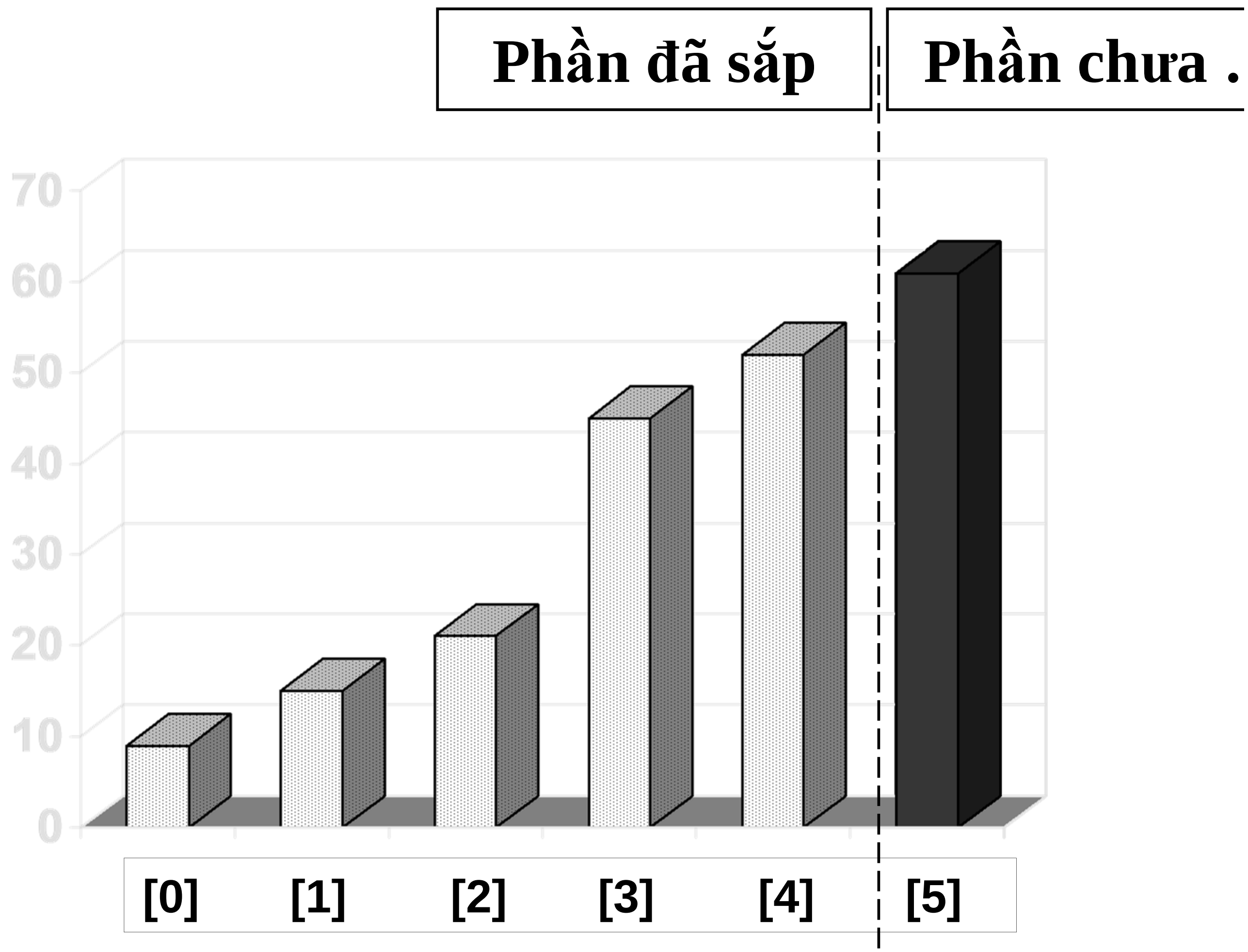
Selection sort Algorithm

- Quá trình lần lượt thêm từng phần tử vào phần đã sắp…
- Phần đã sắp chứa các phần tử nhỏ nhất, sắp tăng dần



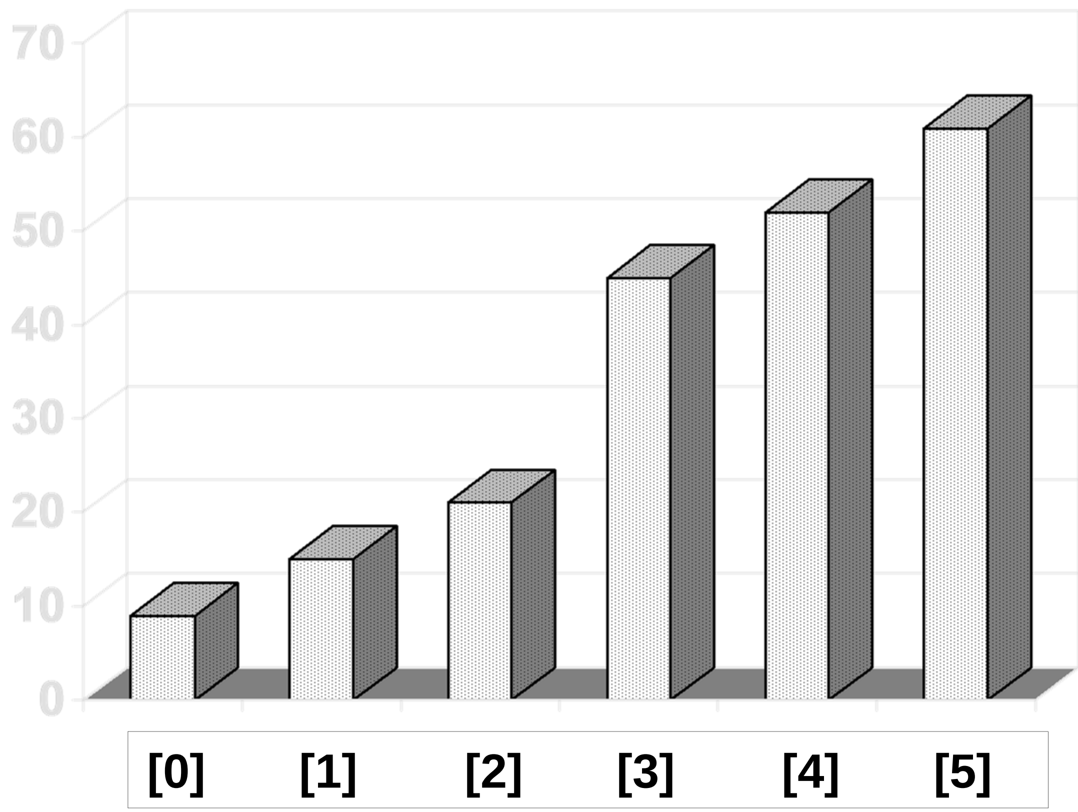
Selection sort Algorithm

- Thuật toán **<u>dừng khi chỉ còn 1 phần tử</u>** (đó là phần tử lớn nhất).



Selection sort Algorithm

- Toàn bộ mảng đã được sắp thứ tự.
- Tổng quát: **<u>chọn</u>** phần tử nhỏ nhất và đưa nó về vị trí đầu của phần chưa được sắp trong mảng.



Selection sort Algorithm (Minh họa chương trình)

```
void SelectionSort (int a[ ], int n )
{
    int min;        // vị trí của phần tử nhỏ nhất (trong phần chưa sắp)
    int tmp;        // biến tạm dùng khi hoán vị
    for (int i = 0; i < n; i++ ) {
        // tìm phần tử nhỏ nhất trong phần chưa sắp
        min = i;
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[min] )         min = j;
        // hoán vị phần tử nhỏ nhất được tìm thấy với phần tử đầu
        if (a[min] < a[i]) { tmp = a[i];    a[i] = a[min]; a[min] = tmp; }
    } // end of  for i
}
```

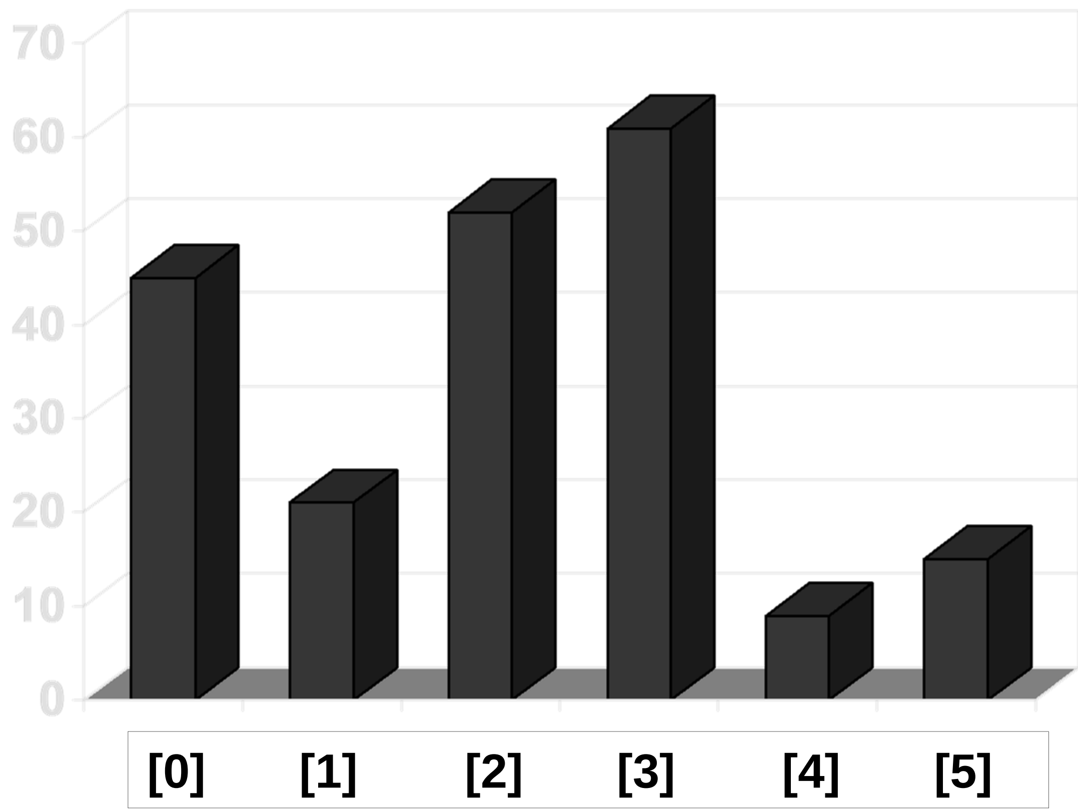
Đánh giá thuật toán (Selection sort Algorithm)

- Trong mọi trường hợp, số phép so sánh là:
  $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 = \mathbf{O(n^2)}$

- Số phép hoán vị:
  - Trường hợp xấu nhất: **O(n)**
  - Trường hợp tốt nhất (mảng đã sắp tứ tự tăng dần): **0**

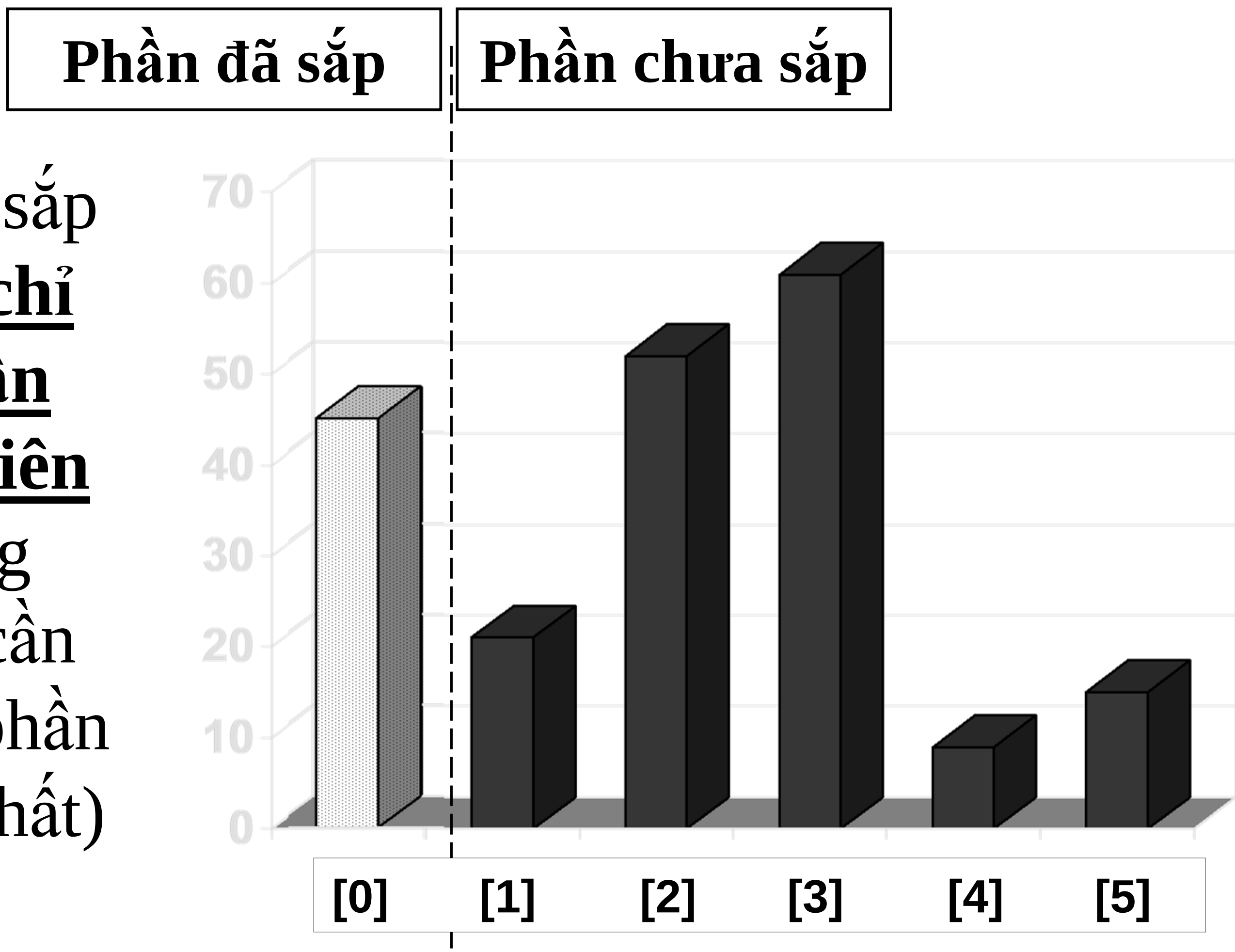
Thuật toán “Chèn trực tiếp” (Insertion sort Algorithm)

- Thuật toán “Chèn trực tiếp” cũng chia mảng thành 2 phần: **<u>phần đã được sắp</u>** và **<u>phần chưa được sắp</u>**

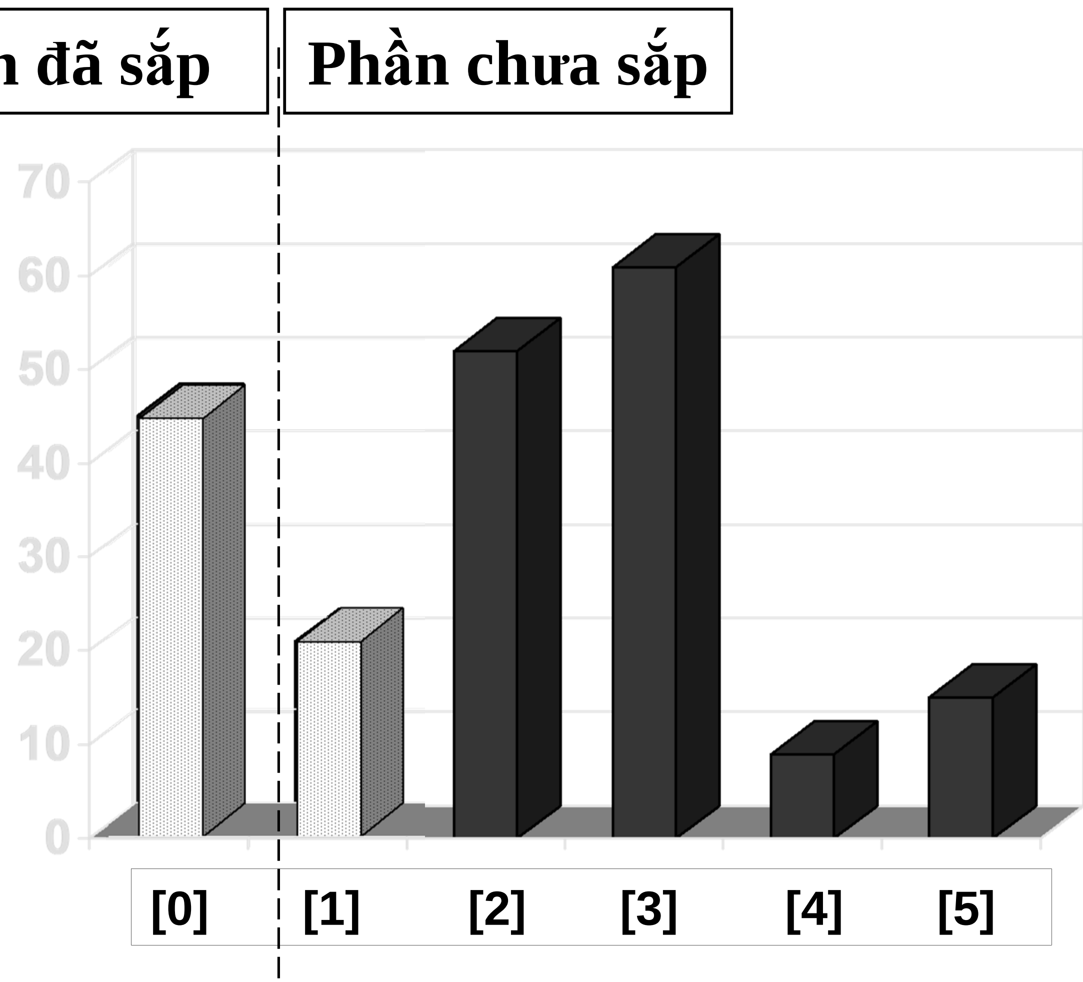


Insertion sort Algorithm

- Phần đã sắp lúc đầu <u>**chỉ có 1 phần tử đầu tiên**</u> của mảng (không cần thiết là phần tử nhỏ nhất)



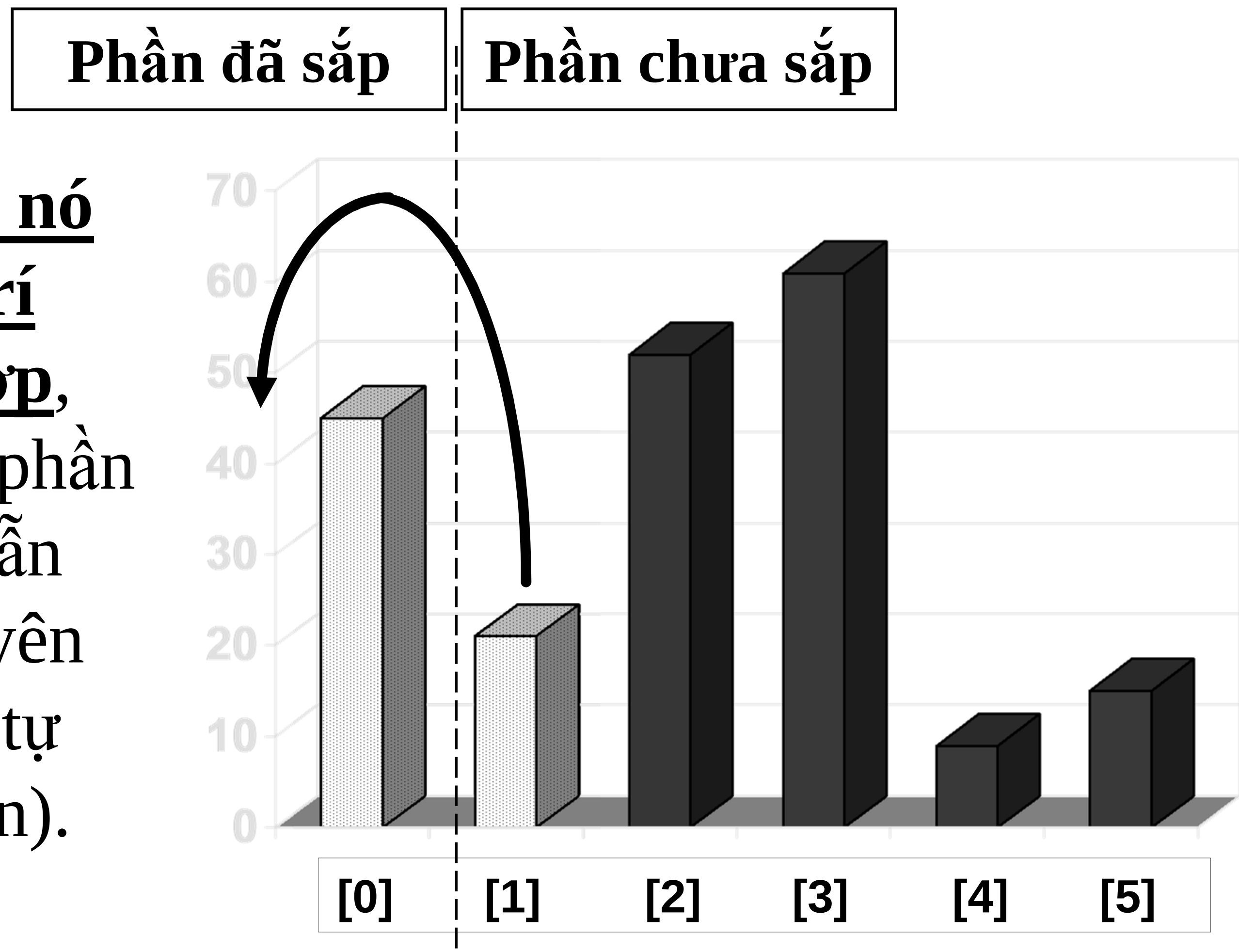
Insertion sort Algorithm



- Mở rộng phần đã sắp bằng cách **<u>thêm vào phần tử đầu tiên</u>** trong phần chưa được sắp…

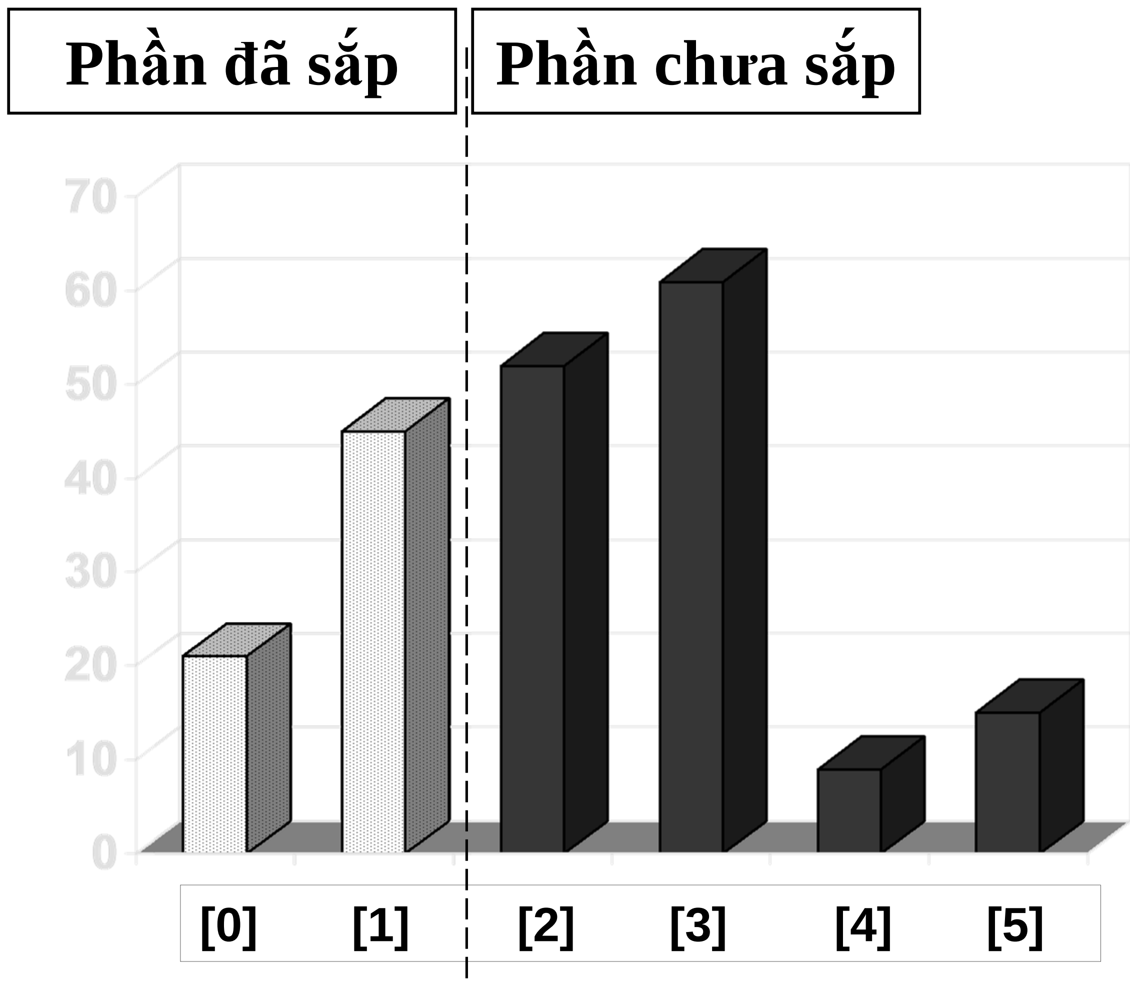
Insertion sort Algorithm

- ...và **<u>đặt nó vào vị trí thích hợp</u>**, sao cho phần đã sắp vẫn giữ nguyên tính thứ tự (tăng dần).



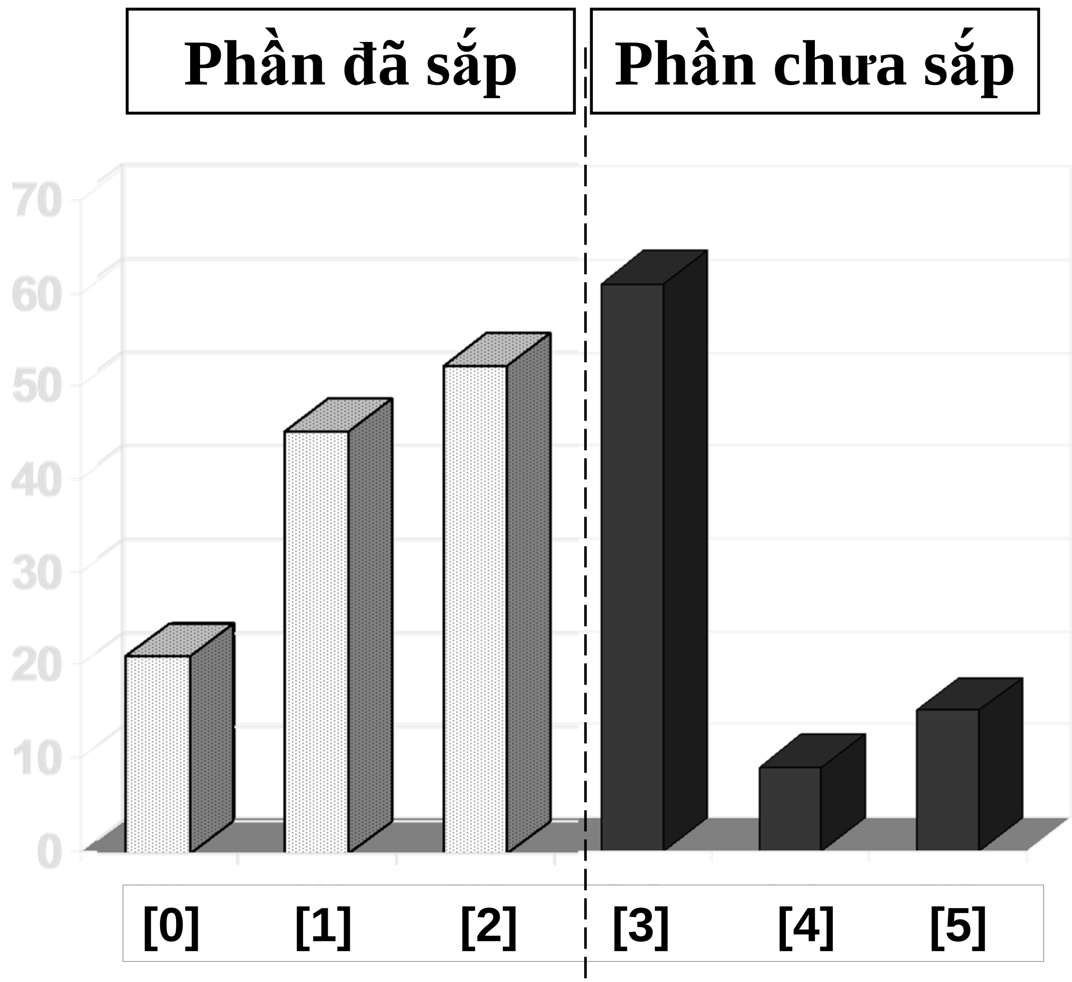
Insertion sort Algorithm

- Trong ví dụ này, phần tử mới được đặt vào vị trí đầu của phần đã sắp.



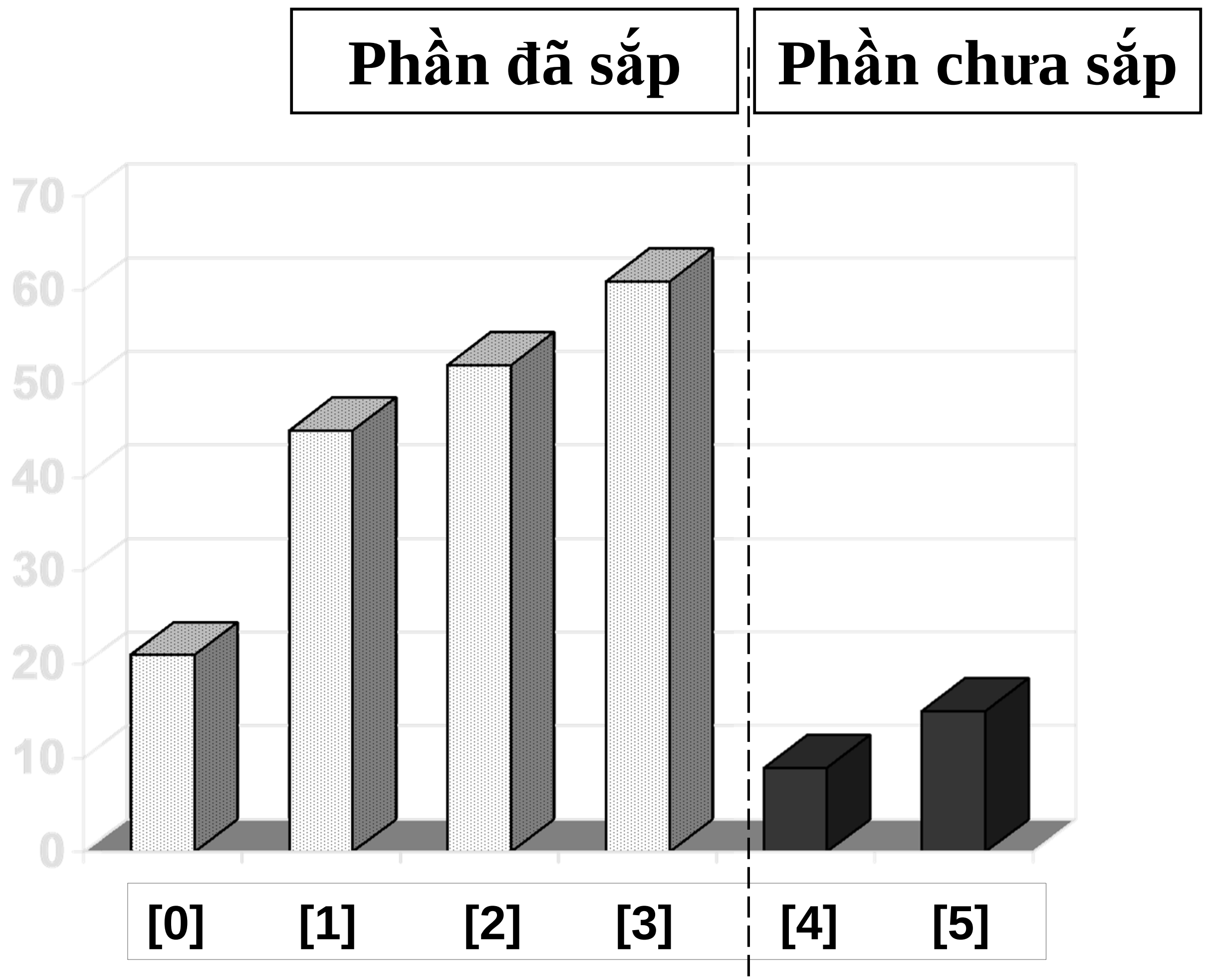
Insertion sort Algorithm

- Đôi khi chúng ta “gặp may”, phần tử mới không cần phải di chuyển.



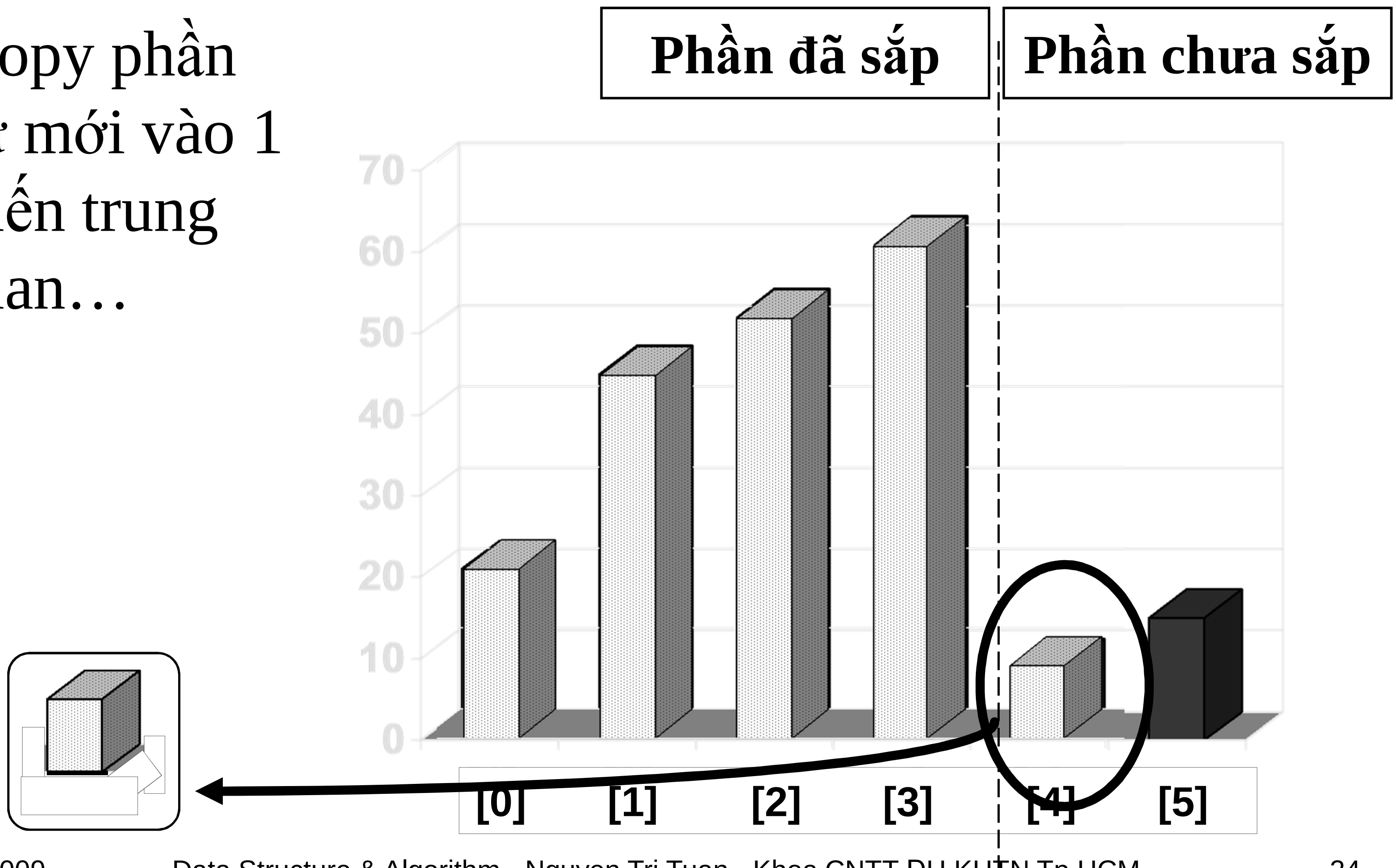
Insertion sort Algorithm

- … và lại "gặp may" thêm 1 lần nữa..



Insertion sort Algorithm
Làm sao để chèn 1 phần tử ?

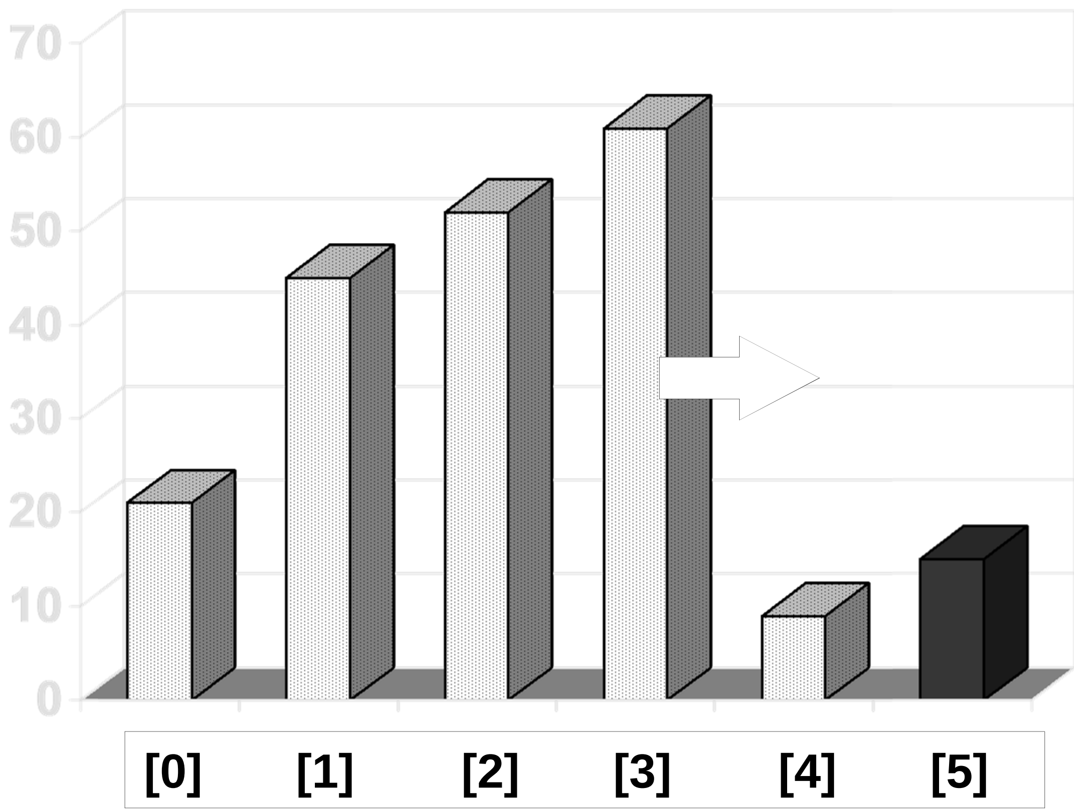
- ☆ Copy phần tử mới vào 1 biến trung gian…

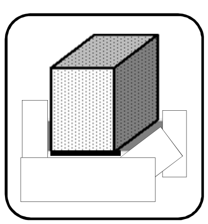


Insertion sort Algorithm

Làm sao để chèn 1 phần tử ?

- …Dịch chuyển các phần tử trong phần đã sắp sang phải…

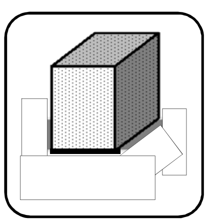


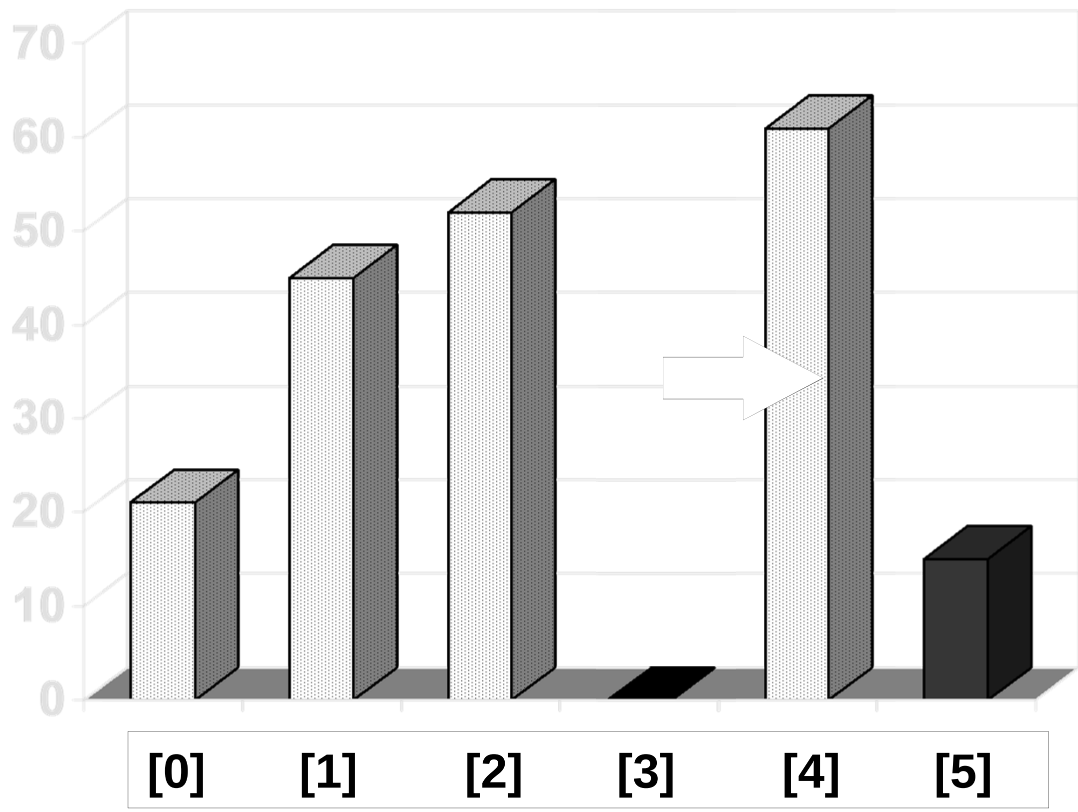


Insertion sort Algorithm

Làm sao để chèn 1 phần tử ?

- …để tạo 1 chỗ trống cho phần tử mới…

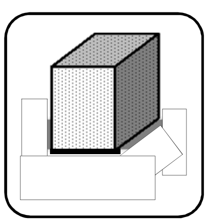


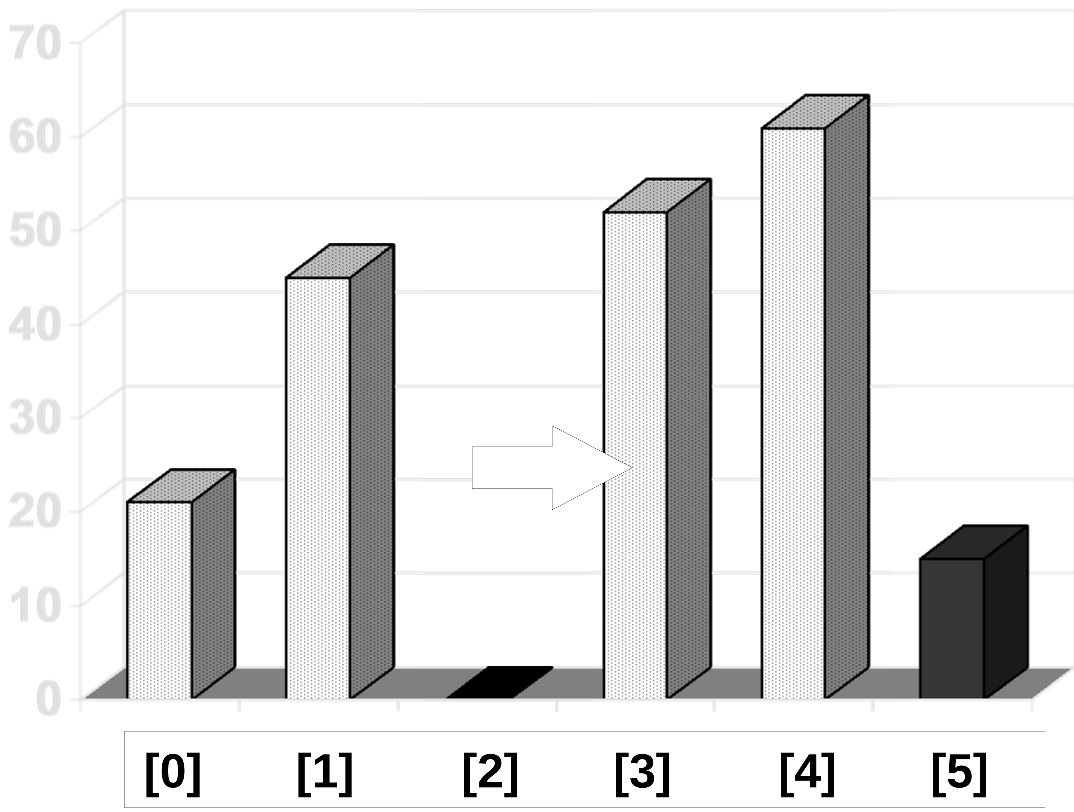


Insertion sort Algorithm

Làm sao để chèn 1 phần tử ?

- …tiếp tục dịch chuyển các phần tử...

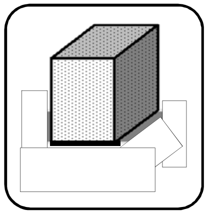


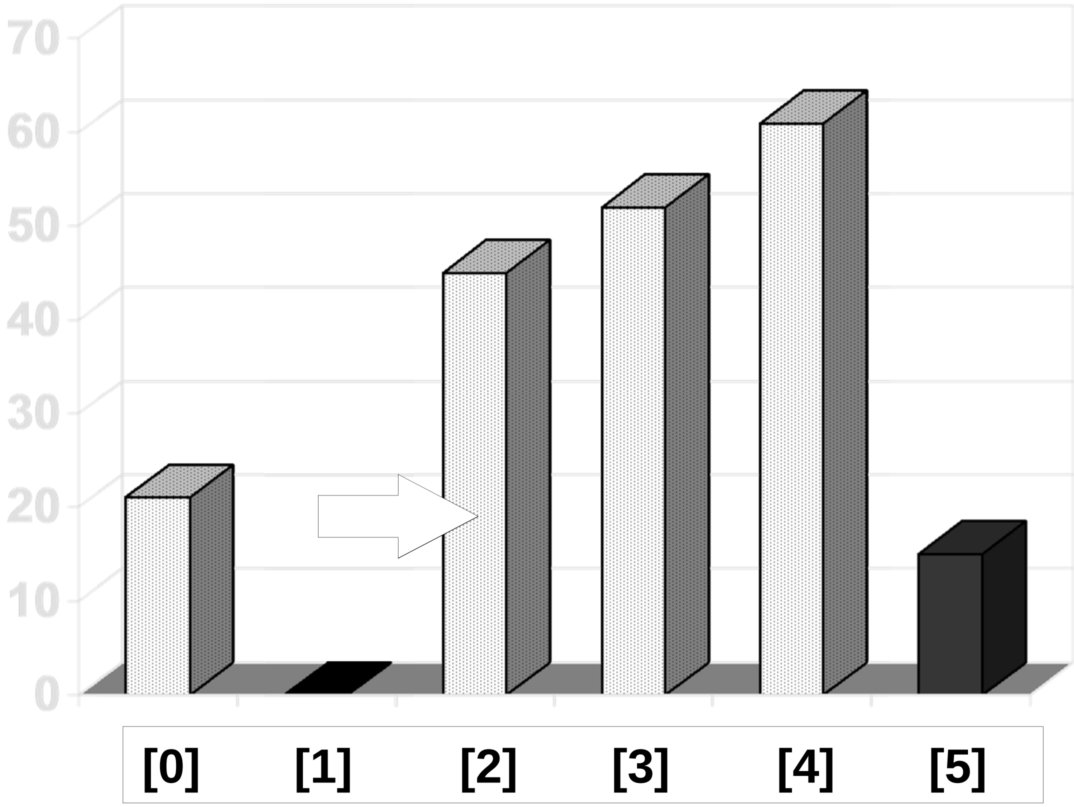


Insertion sort Algorithm

Làm sao để chèn 1 phần tử ?

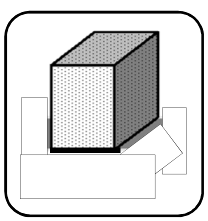
- …tiếp tục dịch chuyển các phần tử...

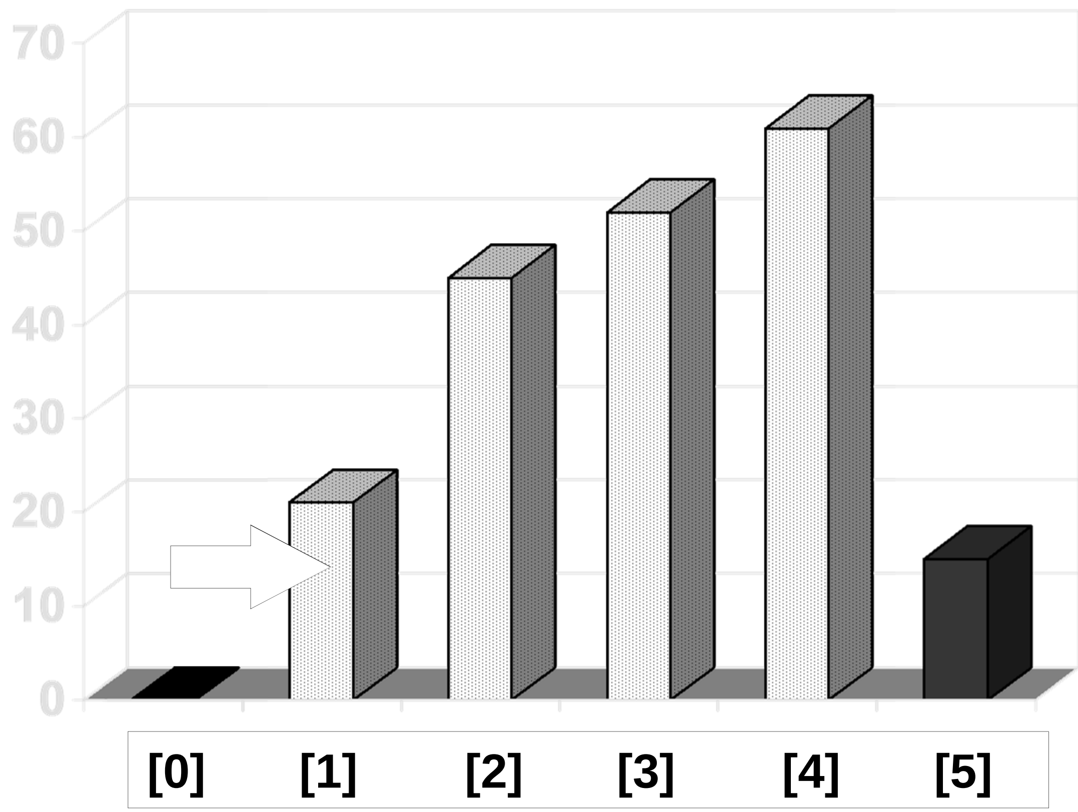




Insertion sort Algorithm
Làm sao để chèn 1 phần tử ?

- ...cho đến khi tìm thấy vị trí thích hợp cho phần tử mới...

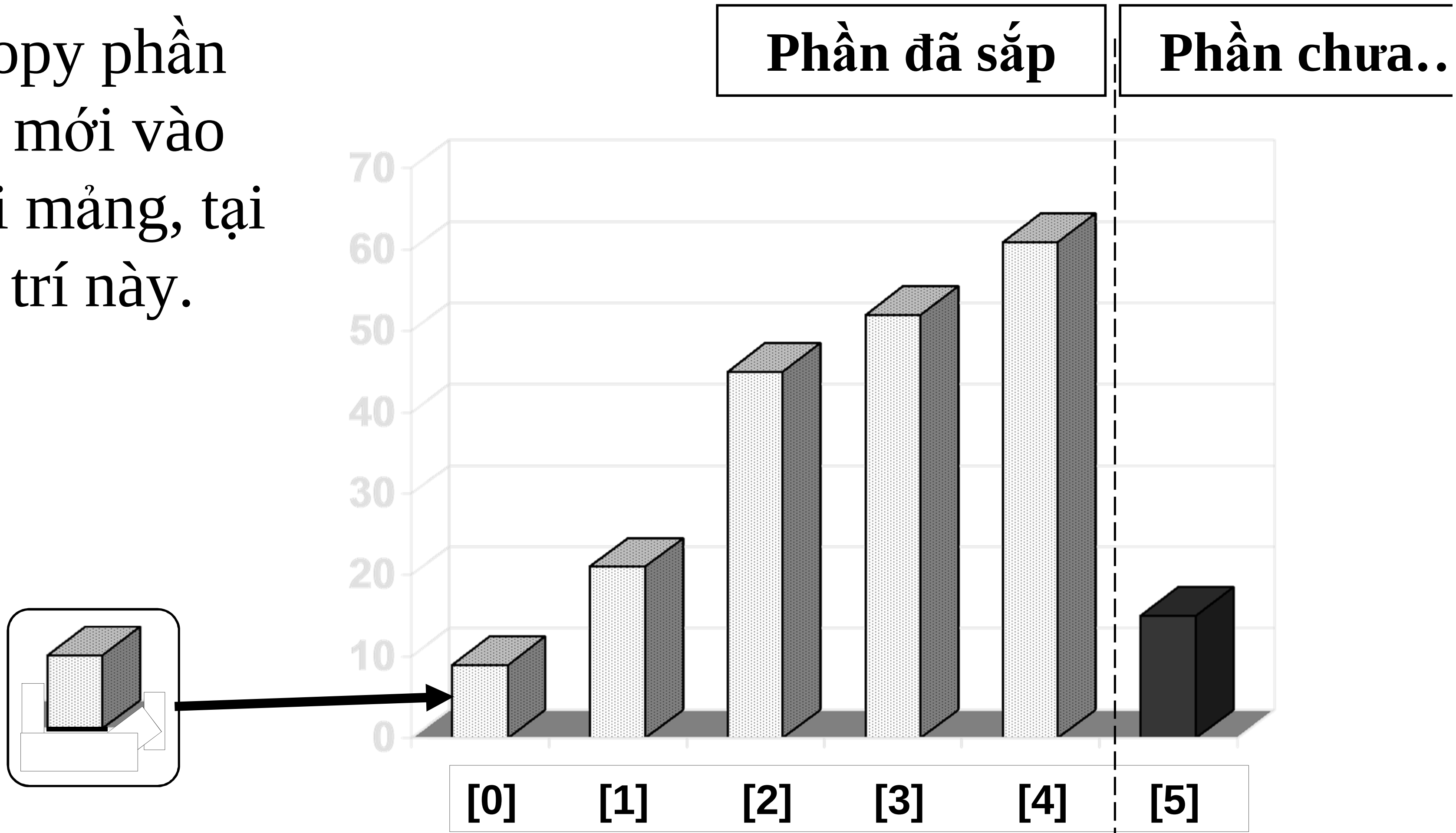




Insertion sort Algorithm

Làm sao để chèn 1 phần tử ?

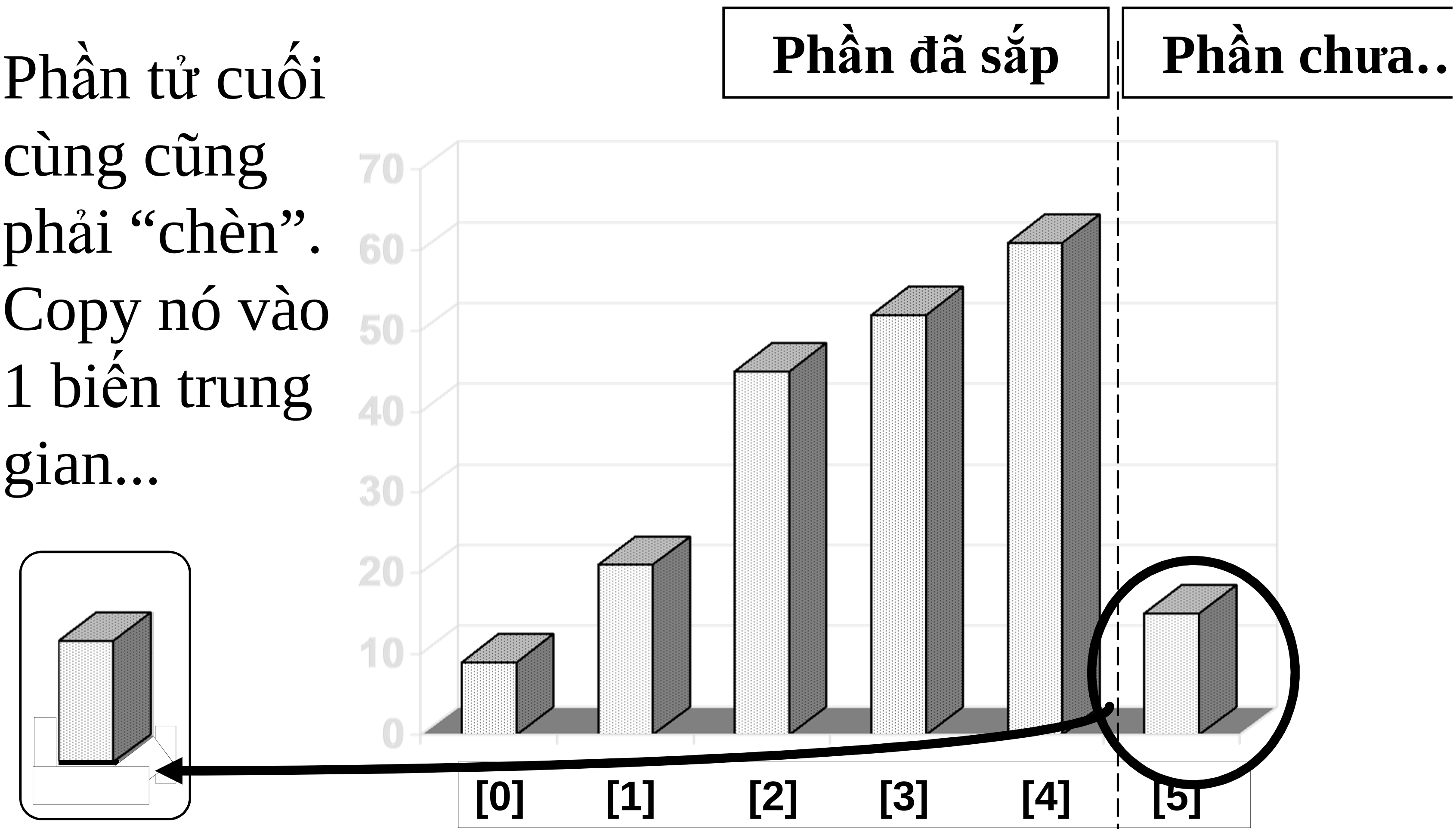
- Copy phần tử mới vào lại mảng, tại vị trí này.



Insertion sort Algorithm

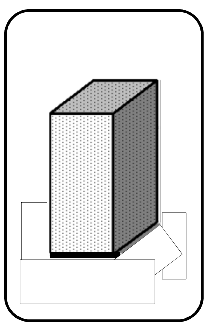
Làm sao để chèn 1 phần tử ?

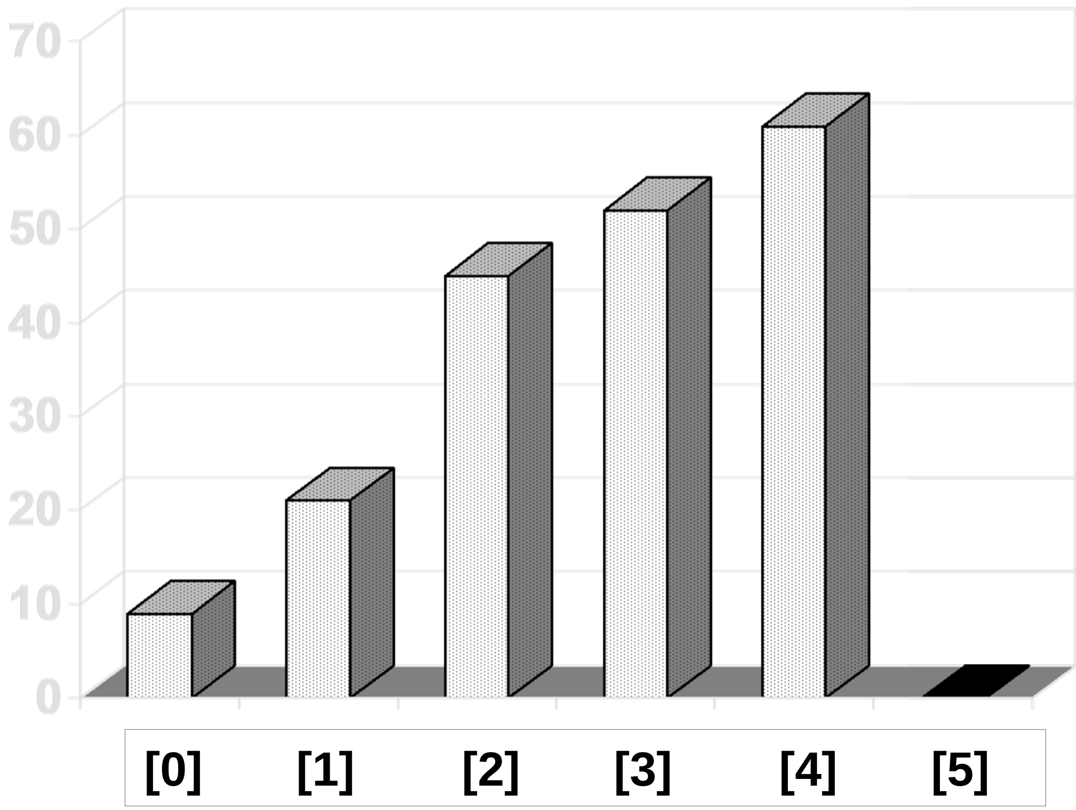
- Phần tử cuối cùng cũng phải “chèn”. Copy nó vào 1 biến trung gian...



Insertion sort Algorithm
Câu hỏi ?

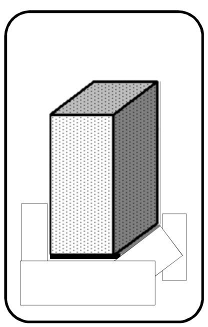
Có bao nhiêu phép dịch chuyển xảy ra ?

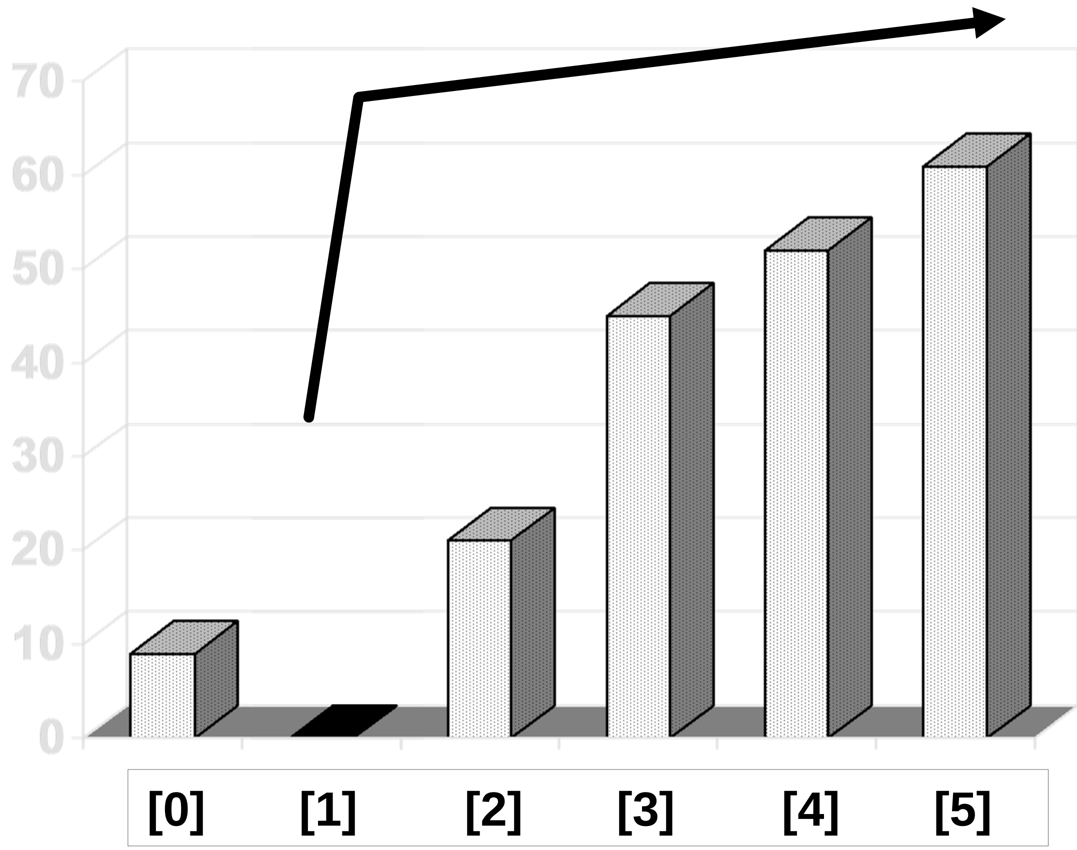




Insertion sort Algorithm
Câu hỏi ?

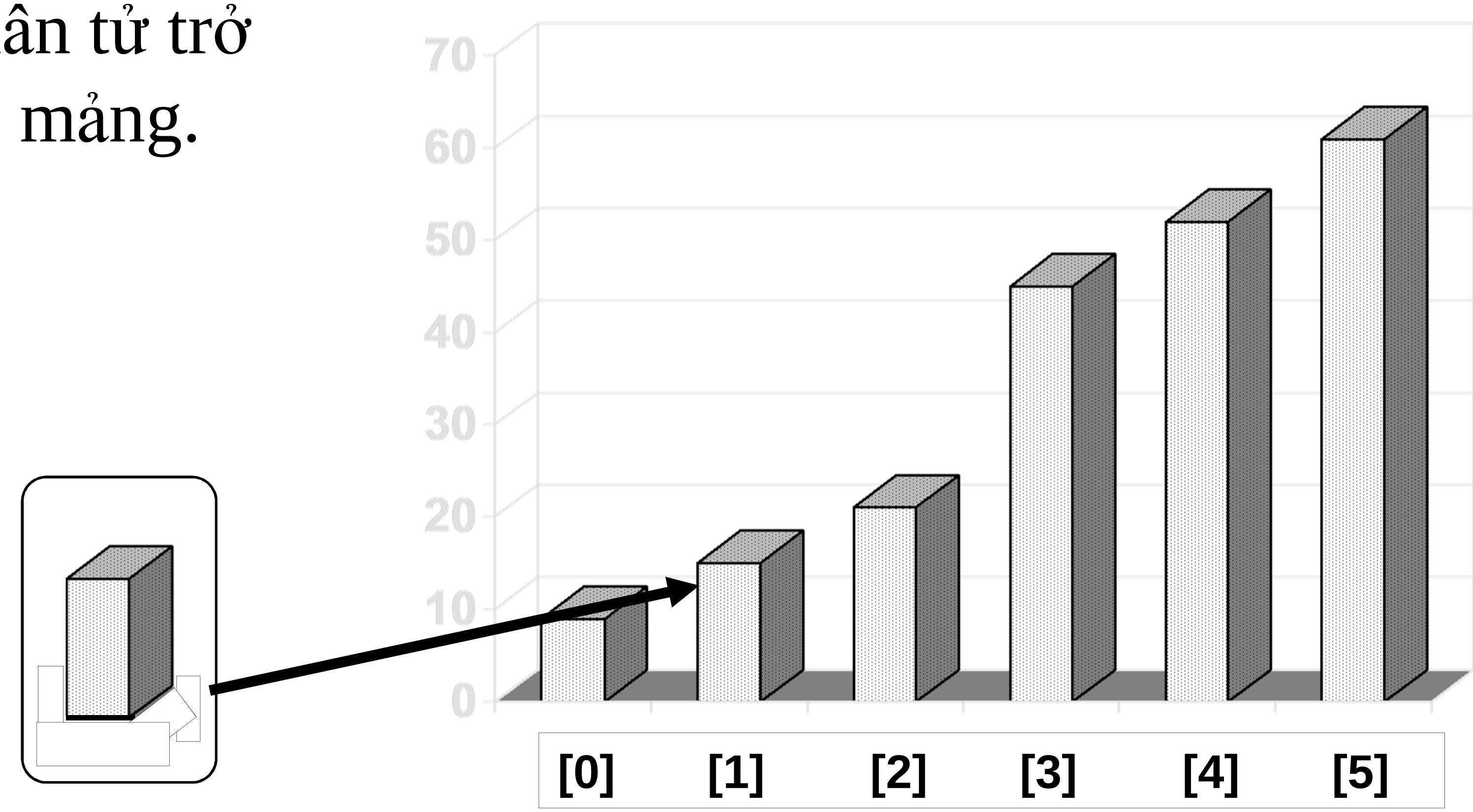
- Có 4 phép dịch chuyển …





Insertion sort Algorithm

- … Copy phần tử trở lại mảng.



Insertion sort Algorithm (Minh họa chương trình)

```
void InsertionSort (int a[ ], int n)
{
    int saved;  // biến trung gian lưu lại giá trị phần tử cần chèn
    for (int i = 1; i < n; i++ ) {
        saved = a[i];        // lưu lại giá trị phần tử cần chèn
        // dịch chuyển các phần tử trong phần đã sắp sang phải
        for (int j = i; j > 0 && saved < a[j-1]; j--)
            a[j] = a[j-1];
        a[j] = saved;        // chèn phần tử vào đúng vị trí
    }   // end of for i
}
```

Đánh giá thuật toán (Insertion sort Algorithm)

- Trường hợp xấu nhất có:

  $1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2 = \mathbf{O(n^2)}$

  phép so sánh và dịch chuyển

- Trường hợp tốt nhất (mảng đã có thứ tự tăng dần): **O(n)** phép so sánh và **0** phép dịch chuyển

Nhận xét chung (Selection & Insertion sort)

- “Chèn trực tiếp” và “Chọn trực tiếp” đều có chi phí cho trường hợp xấu nhất là **<u>O(n²)</u>**
- Do đó, không thích hợp cho việc sắp xếp các mảng lớn
- Dễ cài đặt, dễ kiểm lỗi
- “Chèn trực tiếp” tốt hơn “Chọn trực tiếp”, nhất là khi mảng đã có thứ tự sẵn
- Cần có những thuật toán hiệu quả hơn cho việc sắp xếp các mảng lớn

Thuật toán “Shell sort” (Shell sort Algorithm)

- Được đề xuất vào năm 1959 bởi **Donald L. Shell** trên tạp chí *Communication of the ACM*
- Thuật toán này cải tiến hiệu quả của thuật toán “Chèn trực tiếp”
- Phá vỡ rào cản chi phí **O(n^2)** của những thuật toán sắp xếp trước đó

Shell sort Algorithm

- <u>Ý tưởng:</u>
  - Chia dãy ban đầu thành **h** dãy con

    a_0, a_{0+h}, a_{0+2h}, …

    a_1, a_{1+h}, a_{1+2h}, …

    a_2, a_{2+h}, a_{2+2h}, …

    …
  - Sắp xếp từng dãy con bằng cách sử dụng phương pháp “Chèn trực tiếp”

Shell sort Algorithm

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Ban đầu	81	94	11	96	12	35	17	95	28	58	41	75	15

- Chia dãy thành **h=5** dãy con

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=5	81	94	11	96	12	35	17	95	28	58	41	75	15

- Sắp xếp **5** dãy con bằng phương pháp “Chèn trực tiếp”

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=5	35	17	11	28	12	41	75	15	96	58	81	94	95

Shell sort Algorithm

- Chia dãy thành **h=3** dãy con

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=3	35	17	11	28	12	41	75	15	96	58	81	94	95

- Sắp xếp **3** dãy con bằng phương pháp “Chèn trực tiếp”

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=3	28	12	11	35	15	41	58	17	94	75	81	96	95

Shell sort Algorithm

- Chia dãy thành **h=1** dãy con

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=1	28	12	11	35	15	41	58	17	94	75	81	96	95

- Sắp xếp **1** dãy con bằng phương pháp "Chèn trực tiếp"

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
h=1	11	12	15	17	28	35	41	58	75	81	94	95	96

- Kết thúc !

Shell sort Algorithm

- Thuật toán sử dụng 1 **dãy h_k**:

  $h_1, h_2, h_3, \ldots, h_t$
- (*) Tính chất dãy h_k:
  - $h_i > h_{i+1}$ (dãy giảm dần)
  - $h_t = 1$
- Dãy h_k gọi là **dãy "gia số"** (Increment sequence), dùng để tạo lập các dãy con trong mảng ban đầu
- Trong ví dụ: $h_1 = 5$, $h_2 = 3$, $h_3 = 1$

Shell sort Algorithm

- Vấn đề: Lựa chọn dãy gia số h_k như thế nào ?
  - Mọi dãy h_k thoả mãn tính chất (*) đều chấp nhận được;
  - Tuy nhiên, cho đến nay, người ta chỉ có thể chỉ ra rằng dãy h_k này tốt hơn dãy h_k kia, chứ không thể xác định được dãy nào là tốt nhất
- Chi phí của thuật toán Shell sort phụ thuộc vào 2 vấn đề chính là:
  - Cách thức xây dựng dãy h_k
  - Dữ liệu nhập

Shell sort Algorithm

- Các chiến lược xây dựng dãy h_k đã được khảo sát:
  - D.Shell (1959):

    $h_1 = [n/2]$, $h_{i+1} = [h_i/2]$, $h_t = 1$
  - T.H.Hibbard (1963):

    1, 3, 7, 15, …. , 2^k-1 ($k \in N^*$)

    $N^* = N \setminus \{0\} = \{ 1, 2, 3, 4, ….\}$
  - Knuth:

    $h_1 = 1$, $h_i = h_{i-1} * 3 + 1$, và dừng tai $i = \log_2 n - 1$

    1, 4, 13, 40, 121, ...
  - Pratt (1979):

    1, 2, 3, 4, 6, 8, 9, 12, 16, ..., 2^p3^q, (với $p, q \in N$)

Shell sort Algorithm (Minh họa chương trình)

```
void ShellSort(int h[], int a[], int t, int n)
{
    for (int k=0; k<t; k++) {
            int increment = h[k];
            for (int i=increment; i<n; i++) {
                    int saved = a[i];
                    for (int j=i; j>=increment && saved<a[j-increment];
                          j-=increment)
                            a[j] = a[j-increment];
                    a[j] = saved;
            }
    }
}
```

Đánh giá thuật toán (Shell sort Algorithm)

- Việc phân tích giải thuật này đặt ra những vấn đề toán học hết sức phức tạp mà trong đó có 1 số vấn đề đến nay vẫn chưa được giải quyết
- Người ta vẫn chưa biết chọn dãy h_k như thế nào là phù hợp để cho ra kết quả tốt nhất
- Một số kết quả đã chứng minh:
  - Shell sort với dãy h_k của Donald Shell có số phép gán trong trường hợp xấu nhất là **$O(n^2)$**
  - Sử dụng dãy h_k của Hibbard cần dùng **$O(n^{3/2})$** phép gán
  - Chi phí khi dùng dãy h_k của Pratt là **$O(n(\log_2 n)^2)$**

Thuật toán "Sắp xếp cây" (Heap sort Algorithm)

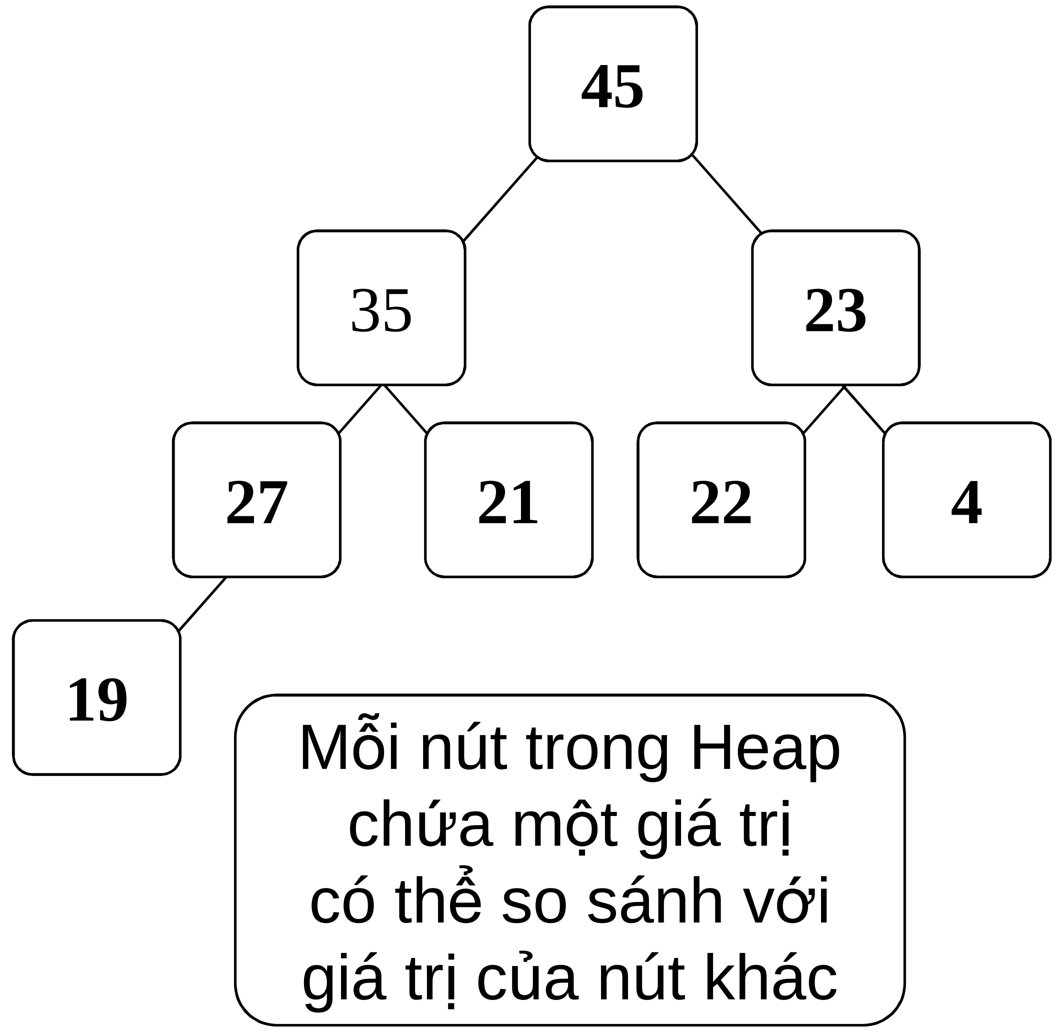
- Được đề xuất vào năm 1964 bởi **J.W.J. Williams** trên tạp chí *Communication of the ACM*
- Đây là thuật toán sắp xếp chậm nhất trong số các thuật toán có độ phức tạp **O(n*log$_2$n)**
- Nhưng nó lại đạt được ưu điểm vì tính đơn giản của cài đặt không đòi hỏi vòng đệ qui phức tạp như của **Quicksort** và không sử dụng mảng phụ như **Mergesort**

Heap sort Algorithm
Nội dung

- Định nghĩa Heap
- Biểu diễn Heap bằng mảng (array)
- Thao tác cơ bản trên Heap
- Thuật toán Heap sort
- Đánh giá thuật toán

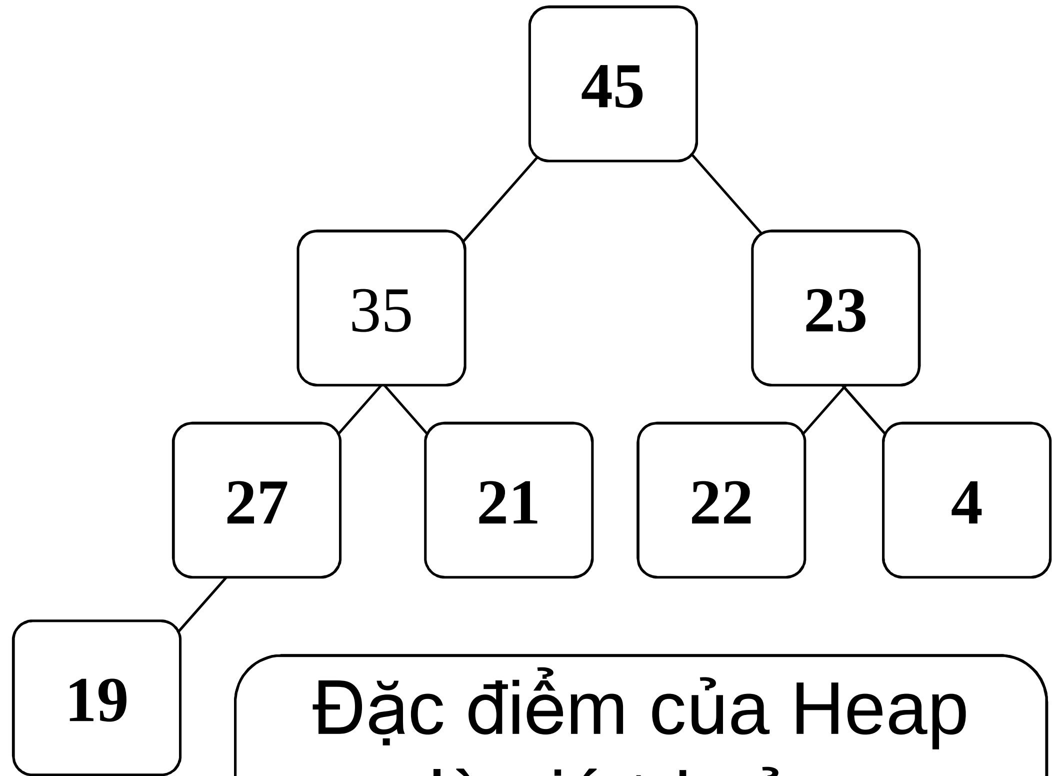
Heap sort Algorithm
Định nghĩa Heap

- **Heap** là một cây nhị phân đầy đủ



Heap sort Algorithm
Định nghĩa Heap

- **Heap** là một cây nhị phân đầy đủ



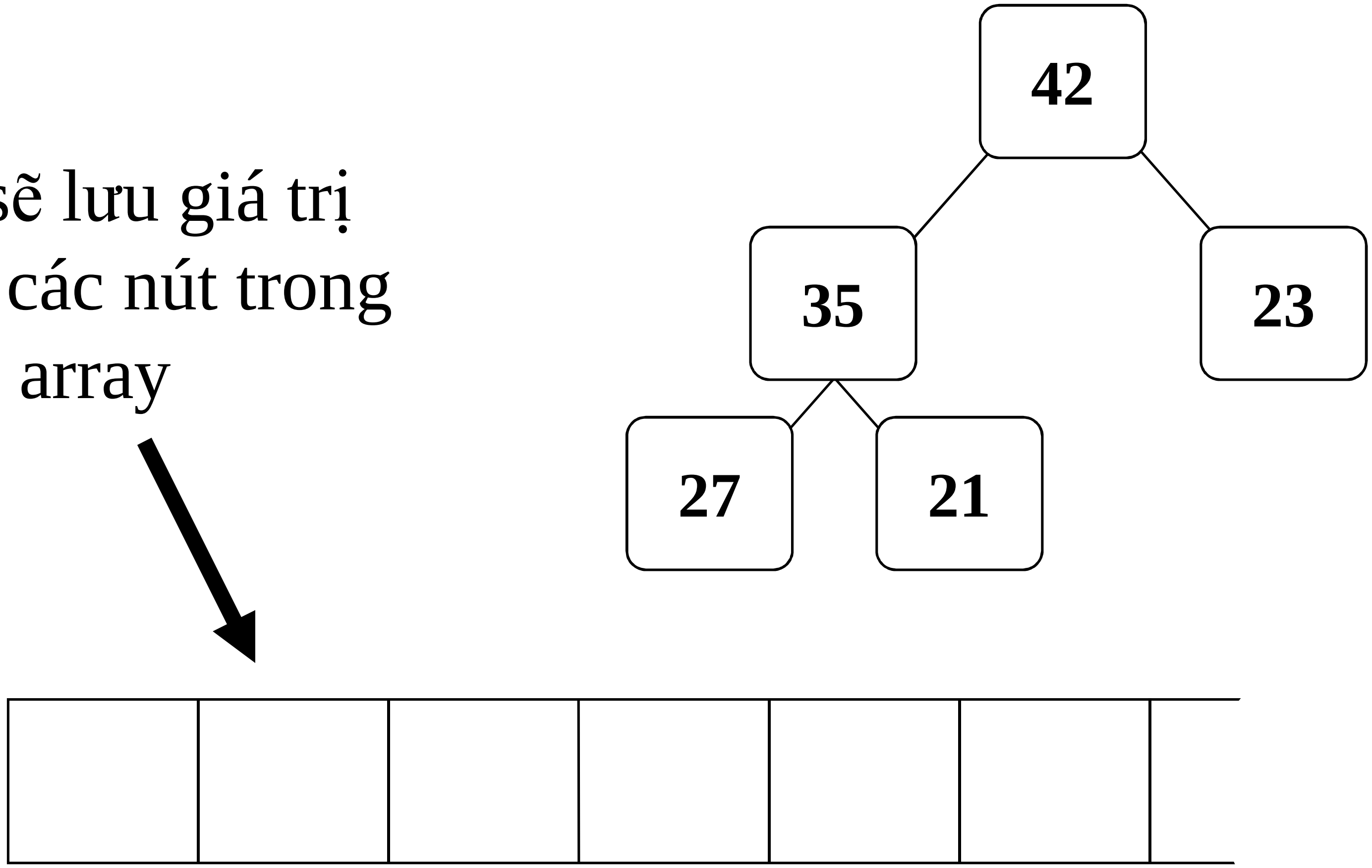
Đặc điểm của Heap là giá trị của mỗi nút >= giá trị của các nút con của nó

Heap sort Algorithm
Định nghĩa Heap

- Heap là một cây nhị phân thỏa các tính chất sau sau:
  - Là một cây đầy đủ;
  - Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con
- Hệ quả:
  - Nút lớn nhất là … ?

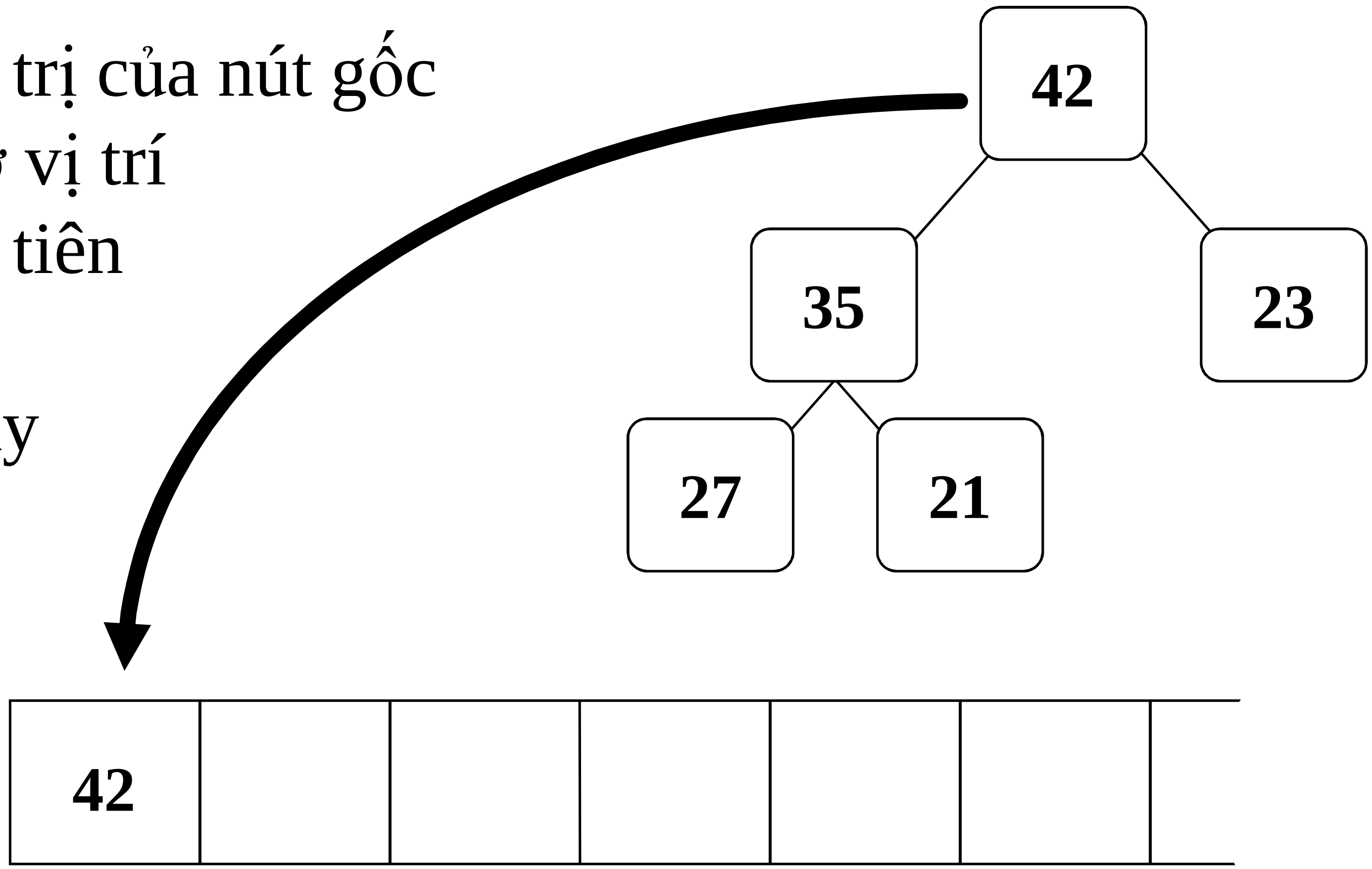
Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Ta sẽ lưu giá trị của các nút trong một array



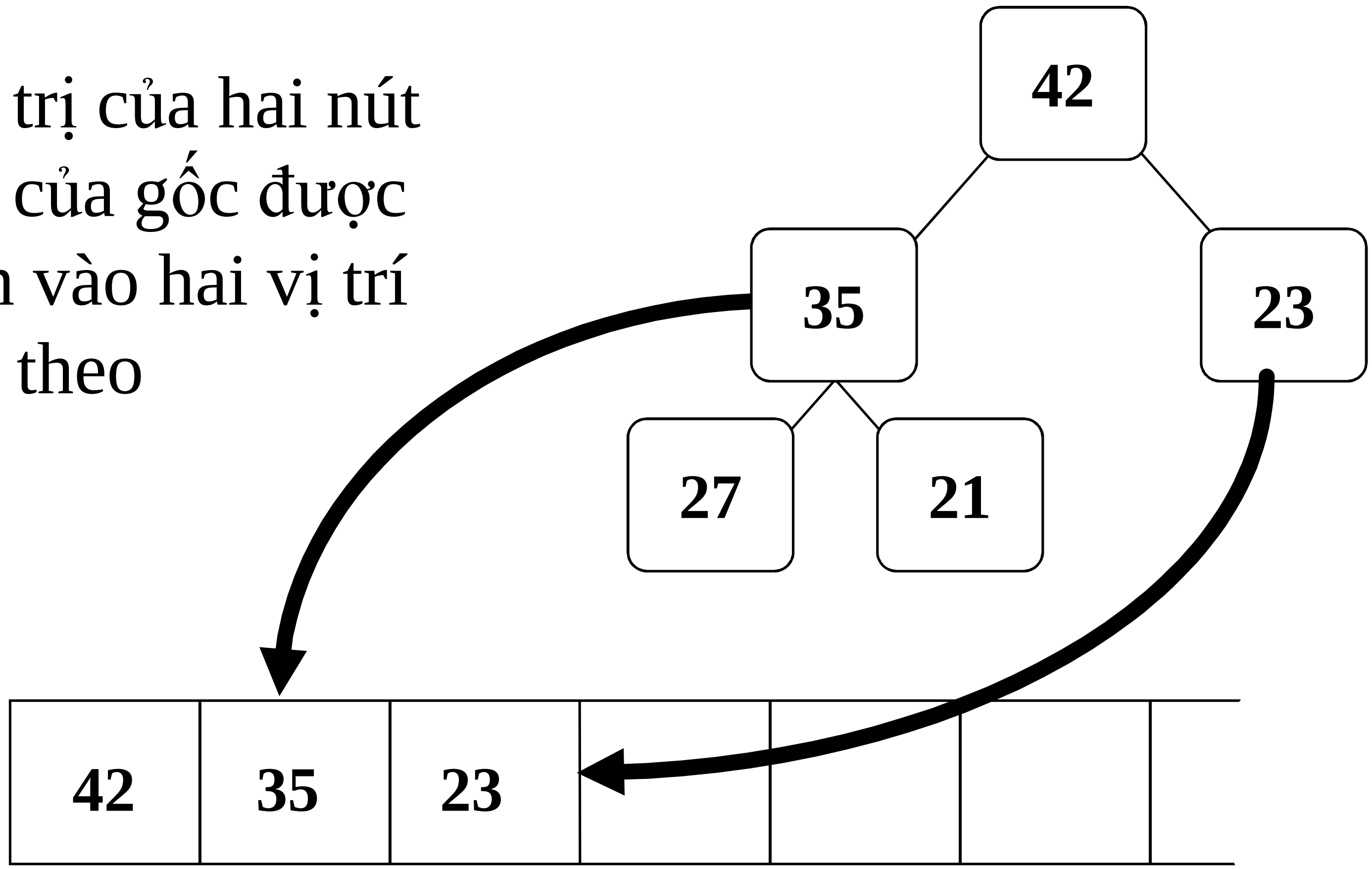
Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Giá trị của nút gốc sẽ ở vị trí đầu tiên của array



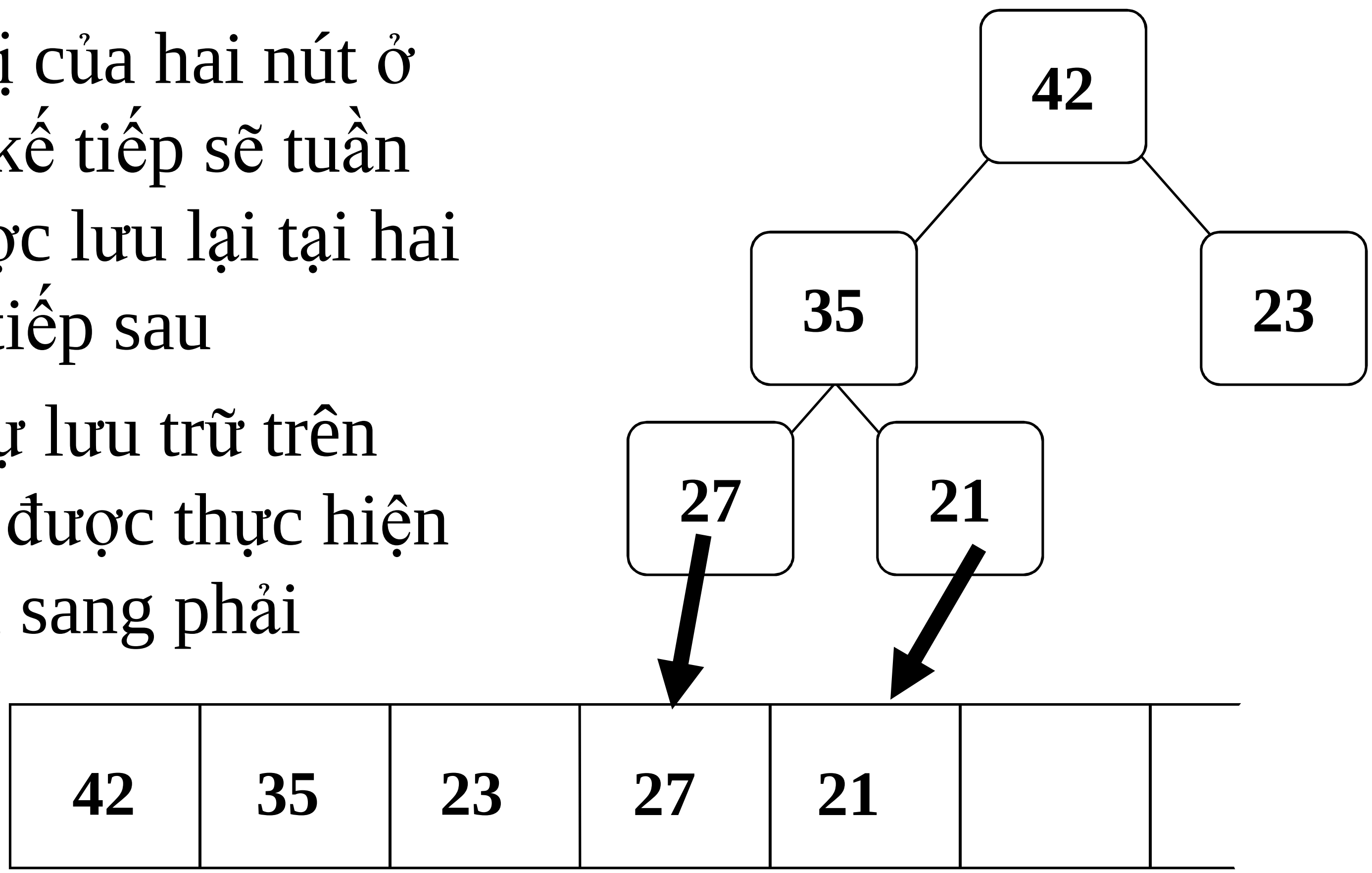
Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Giá trị của hai nút con của gốc được điền vào hai vị trí tiếp theo



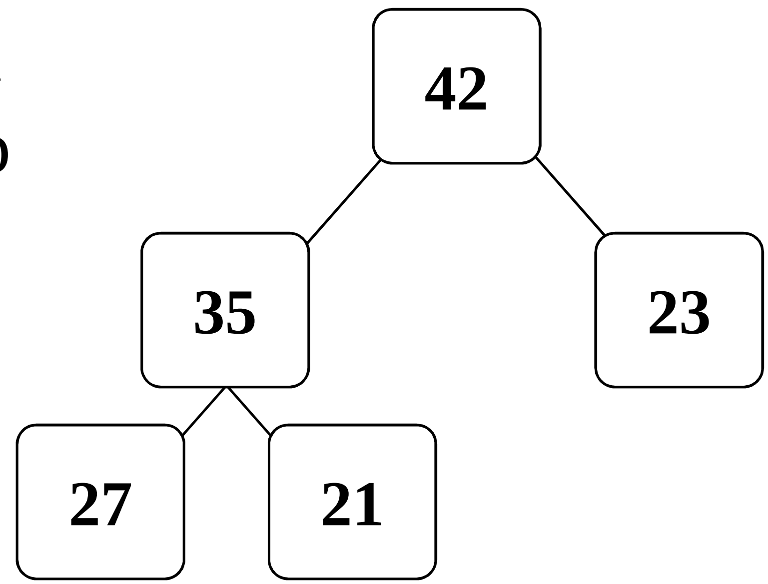
Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Giá trị của hai nút ở hàng kế tiếp sẽ tuần tự được lưu lại tại hai vị trí tiếp sau
- Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải



Heap sort Algorithm
Biểu diễn Heap bằng mảng

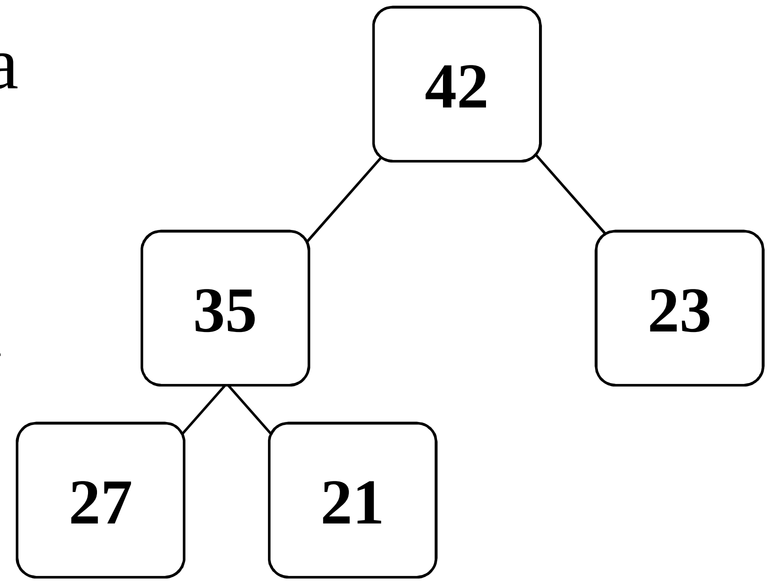
- Liên kết giữa các nút được hiểu ngầm, không trực tiếp dùng con trỏ.
- Array được xem là cây chỉ do cách ta xử lý dữ liệu trên đó



42	**35**	**23**	**27**	**21**		

Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Nếu ta biết được chỉ số của 1 phần tử trên mảng, ta sẽ dễ dàng xác định được chỉ số của nút cha và (các) nút con của nó.



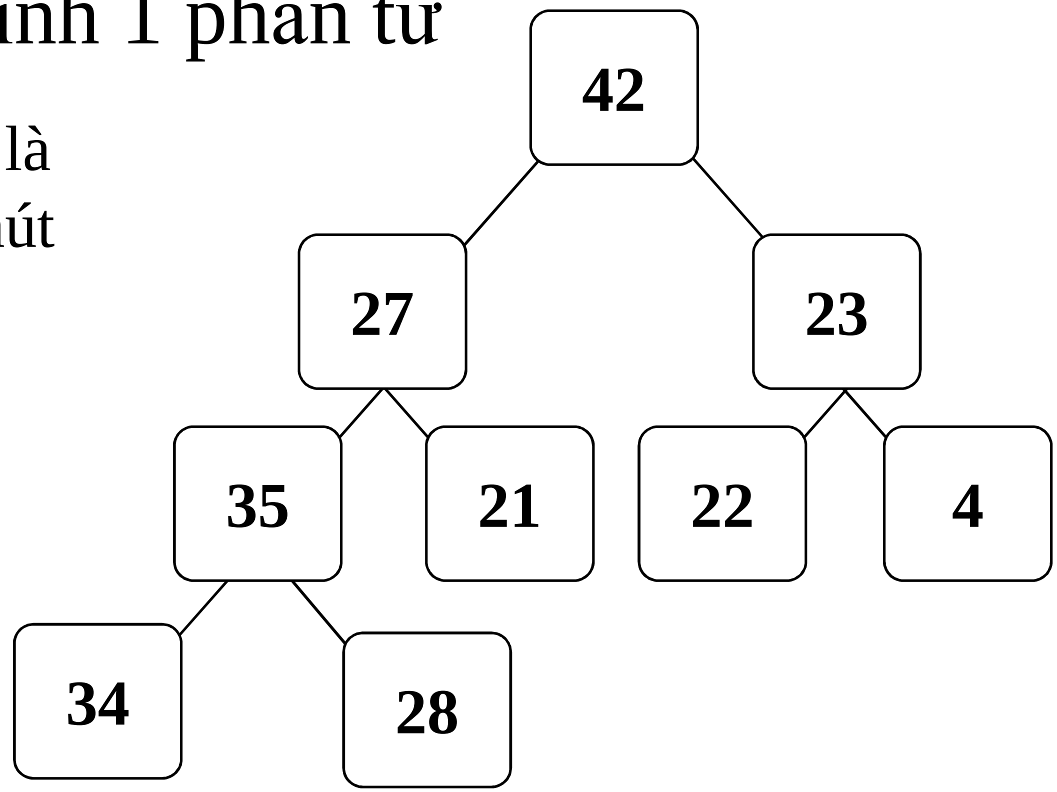
42	**35**	**23**	**27**	**21**		
[0]	[1]	[2]	[3]	[4]		

Heap sort Algorithm
Biểu diễn Heap bằng mảng

- Nút gốc ở chỉ số [0]
- Nút cha của nút [i] có chỉ số là [(i-1)/2]
- Các nút con của nút [i] (nếu có) có chỉ số [2i+1] và [2i+2]
- Ví dụ:
  - Nút con của nút [0] là nút [1] và nút [2]
  - Nút cha của nút [7] là nút [3]
  - Nút cha của nút [8] là nút [3]

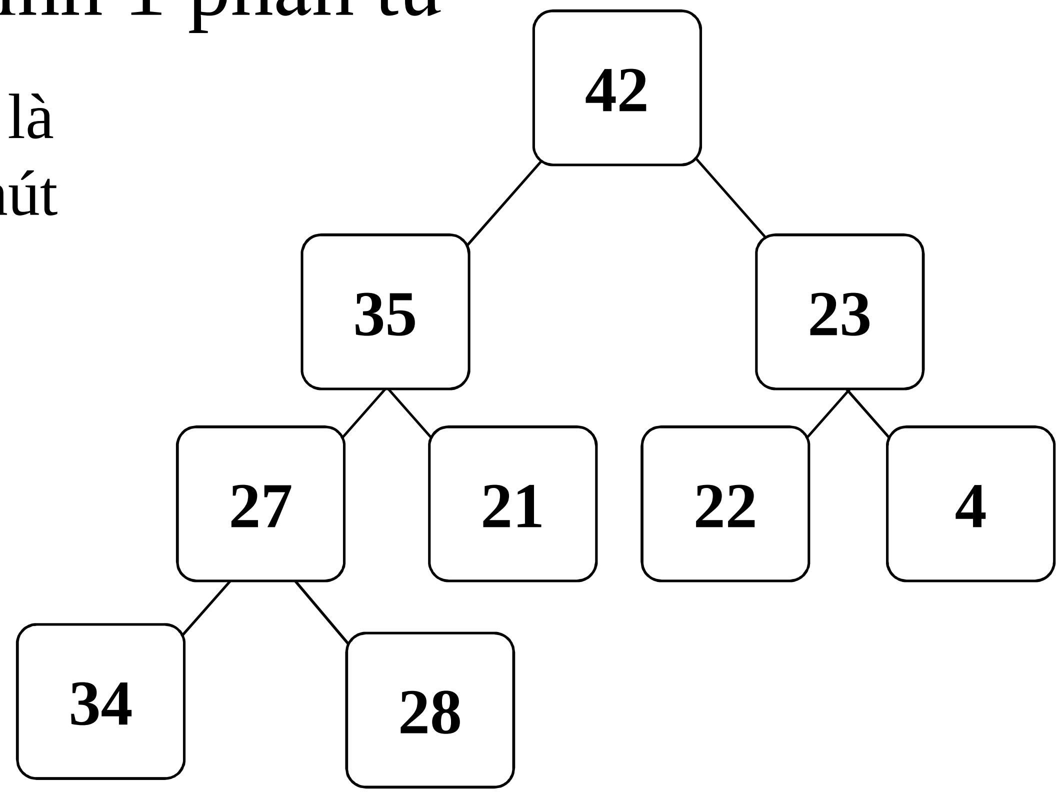
Heap sort Algorithm
Thao tác cơ bản trên Heap

- **Heapify** - Điều chỉnh 1 phần tử
  - Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó
  - Tiến hành đổi chỗ với nút con có giá trị lớn nhất



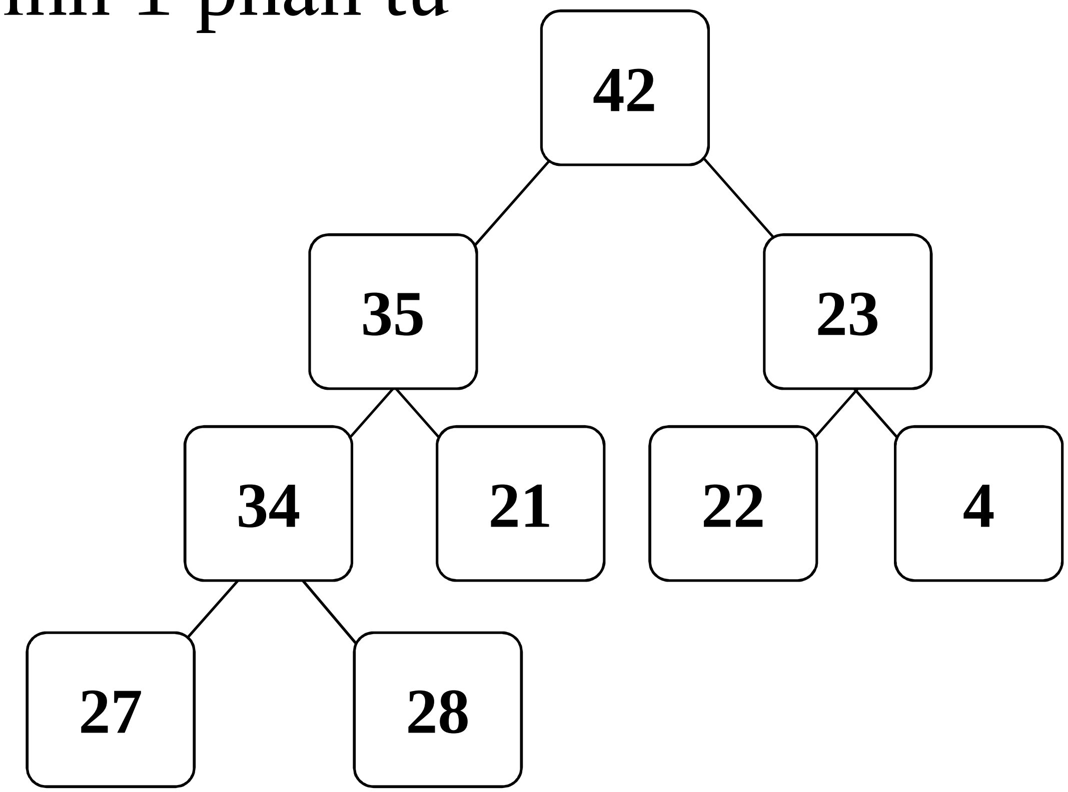
Heap sort Algorithm
Thao tác cơ bản trên Heap

- **Heapify** - Điều chỉnh 1 phần tử
  - Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó
  - Tiến hành đổi chỗ với nút con có giá trị lớn nhất



Heap sort Algorithm
Thao tác cơ bản trên Heap

- **Heapify** - Điều chỉnh 1 phần tử
  - Hoàn tất !



Heap sort Algorithm
Thao tác cơ bản trên Heap

```
void Heapify(int a[], int n, int i)  // Điều chỉnh phần tử a[i]
{
    int saved = a[i];                // lưu lại giá trị của nút i
    while (i < n/2) {                // a[i] không phải là nút lá
        int child = 2*i + 1;         // nút con bên trái của a[i]
        if (child < n-1)
            if (a[child] < a[child+1])   child ++;
        if (saved >= a[child])  break;
        a[i] = a[child];
        i = child;
    }
    a[i] = saved;                    // Gán giá trị của nút i vào vị trí mới
}
```

Heap sort Algorithm
Thuật toán Heap sort

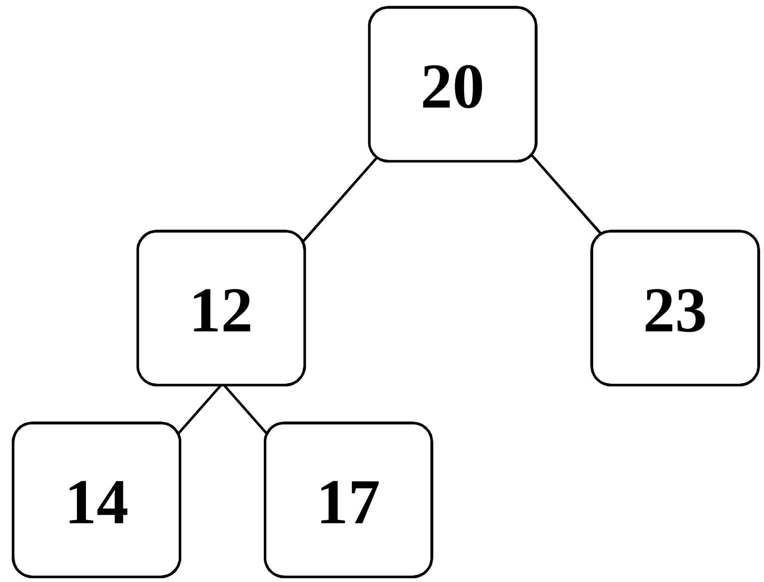
- **Xây dựng Heap**: Sử dụng thao tác Heapify để chuyển đổi một mảng bình thường thành Heap
- **Sắp xếp**:
  - Hoán vị phần tử cuối cùng của Heap với phần tử đầu tiên của Heap (có giá trị lớn nhất)
  - Loại bỏ phần tử cuối cùng
  - Thực hiện thao tác Heapify để điều chỉnh phần tử đầu tiên

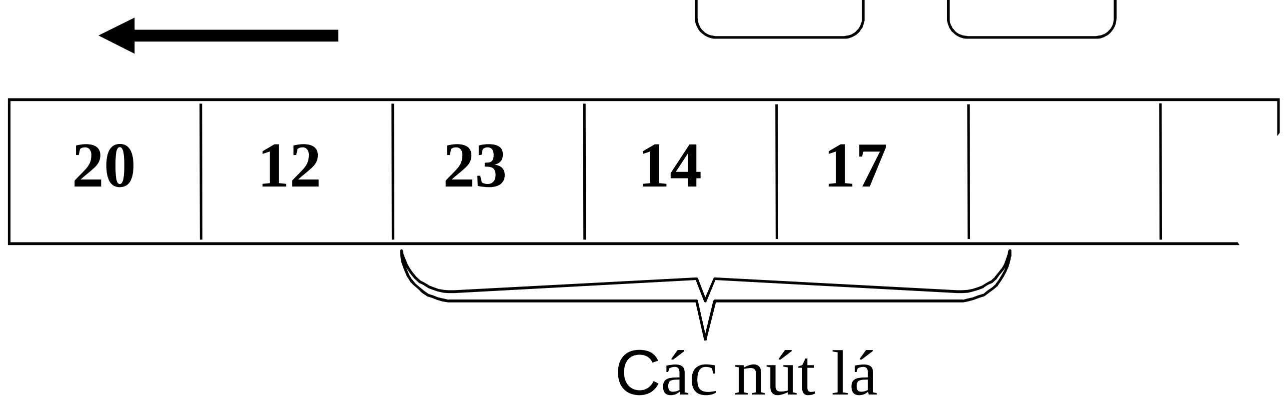
Heap sort Algorithm
Thuật toán Heap sort - Xây dựng Heap

- Tất cả các phần tử trên mảng có chỉ số [n/2] đến [n-1] đều là nút lá
- Mỗi nút lá được xem là Heap có một phần tử
- Thực hiện thao tác Heapify trên các phần tử có chỉ số từ [n/2]-1 đến [0]

Heap sort Algorithm
Thuật toán Heap sort - Xây dựng Heap

- Thực hiện Heapify với tất cả các nút không phải là lá
- Theo thứ tự từ trái sang phải





Heap sort Algorithm
Thuật toán Heap sort - Xây dựng Heap

```
// Xây dựng mảng bình thường a trở thành
// một Heap
void BuildHeap(int a[], int n)
{
        for (int i = n/2 - 1; i >= 0; i--)
                Heapify(a, n, i);
}
```

Heap sort Algorithm
Thuật toán Heap sort - Sắp xếp

```
void  HeapSort(int a[], int n)
{
        BuildHeap(a, n);
        for (int i=n-1; i>=0; i--) {
                Hoán vị a[0] với a[i]
                Heapify(a, i, 0);
        }
}
```

Đánh giá thuật toán (Heap sort Algorithm)

- Heap sort luôn có độ phức tạp là **O(n* log₂n)**
- Quick sort thường có độ phức tạp là **O(n* log₂n)** nhưng trường hợp xấu nhất lại có độ phức tạp **O(n²)**
- Nhìn chung Quick sort nhanh hơn Heap sort 2 lần nhưng Heap sort lại có độ ổn định cao trong mọi trường hợp

Thuật toán “Sắp xếp trộn” (Merge sort Algorithm)

- Là một phươhg pháp sắp xếp dạng **“Chia để trị”** (Divide and Conquer)
- Nguyên tắc “Chia để trị”:
  - Nếu vấn đề nhỏ thì xử lý ngay
  - Nếu vấn đề lớn: chia thành 2 vấn đề nhỏ, mỗi phần bằng ½
  - Giải quyết từng vấn đề nhỏ
  - Kết hợp kết quả của những vấn đề nhỏ lại với nhau

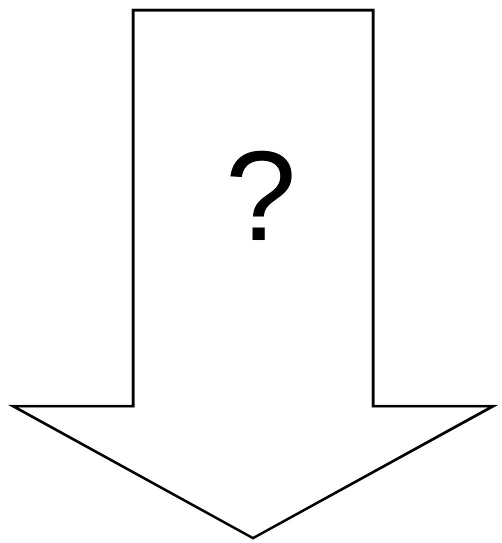
Merge sort Algorithm

- Ý tưởng:
  - Chia dãy cần sắp thành 2 phần, ở vị trí giữa
  - Nếu số phần tử của mỗi phần > 1 thì
    - Sắp sếp mỗi phần bằng Merge sort
  - Trộn 2 phần đã được sắp lại với nhau
- Thuật toán Merge sort có thể cài đặt bằng **đệ qui**

Merge sort Algorithm

- Ví dụ:

16	12	7	6	3	2	18	10



2	3	6	7	10	12	16	18

Merge sort Algorithm

- Ví dụ:

16	12	7	6	3	2	18	10

Chia

16	12	7	6

3	2	18	10

Merge sort Algorithm

- <u>Ví dụ:</u>

16	12	7	6	3	2	18	10

Chia

16	12	7	6

3	2	18	10

Chia

16	12

7	6

3	2

18	10

Merge sort Algorithm

- Ví dụ:

16	12	7	6	3	2	18	10

Chia

16	12	7	6	3	2	18	10

Chia

16	12	7	6	3	2	18	10

Chia

16	12	7	6	3	2	18	10

Merge sort Algorithm

- Ví dụ:

16	12	7	6	3	2	18	10

Chia [16 | 12 | 7 | 6] [3 | 2 | 18 | 10]

Chia [16 | 12] [7 | 6] [3 | 2] [18 | 10]

Chia [16] [12] [7] [6] [3] [2] [18] [10]

Trộn [12 | 16] [6 | 7] [2 | 3] [10 | 18]

Merge sort Algorithm

- Ví dụ:

16	12	7	6	3	2	18	10

Chia [16 | 12 | 7 | 6] [3 | 2 | 18 | 10]

Chia [16 | 12] [7 | 6] [3 | 2] [18 | 10]

Chia [16] [12] [7] [6] [3] [2] [18] [10]

Trộn [12 | 16] [6 | 7] [2 | 3] [10 | 18]

Trộn [6 | 7 | 12 | 16] [2 | 3 | 10 | 18]

Merge sort Algorithm

- <u>Ví dụ:</u>

	[16 12 7 6 3 2 18 10]
Chia	[16 12 7 6] [3 2 18 10]
Chia	[16 12] [7 6] [3 2] [18 10]
Chia	[16] [12] [7] [6] [3] [2] [18] [10]
Trộn	[12 16] [6 7] [2 3] [10 18]
Trộn	[6 7 12 16] [2 3 10 18]
Trộn	[**2 3 6 7 10 12 16 18**]

Merge sort Algorithm (Minh họa chương trình)

```
void MergeSort(int a[], int Left, int Right)
{
    int Mid;       // Vị trí của phần tử giữa
    if (Left < Right)  {  // Dãy có > 1 phần tử
        Mid = (Left + Right)/2;           // Chia thành 2 dãy …
        MergeSort(a, Left, Mid);          // Sort ½ dãy bên trái
        MergeSort(a, Mid+1, Right);       // Sort ½ dãy bên phải
        // Trộn 2 dãy lại với nhau
        Merge(a, Left, Mid, Right);
    }
}   // end of MergeSort
```

Merge sort Algorithm
Thao tác “trộn”

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

c1 c2

Bảng tạm

?	?	?	?	?	?	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

d

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

c1 c2

Bảng tạm

2	?	?	?	?	?	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

d

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

c1 c2

Bảng tạm

2	3	?	?	?	?	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

d

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	c1					c2	

Bảng tạm

2	3	6	?	?	?	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
			d				

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

c1 c2

Bảng tạm

2	3	6	7	?	?	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

d

tiếp tục…

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

6	7	12	16	2	3	10	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

c1 c2

Bảng tạm

2	3	6	7	10	12	16	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Hoàn tất !

d

Merge sort Algorithm
Thao tác "trộn"

Hai dãy con đã sắp

2	3	6	7	10	12	16	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Copy trở lại vào mảng

Bảng tạm

2	3	6	7	10	12	16	18
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Merge sort Algorithm
Thao tác “trộn” – Minh họa chương trình

```
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…

Merge sort Algorithm
Thao tác "trộn" – Minh họa chương trình

```
    // 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
```

Đánh giá thuật toán (Merge sort Algorithm)

Trộn 2 dãy có kích thước *n/2*:

O(*n*)

Trộn 4 dãy có kích thước *n/4*:

O(*n*)

.
.
.

Trộn n dãy có kích thước *1*:

O(*n*)

$O(\log_2 n)$ lần

Đánh giá thuật toán (Merge sort Algorithm)

- Chi phí **O(n*log$_2$n)** để sắp xếp bất kỳ 1 dãy nào
- Sử dụng 1 vùng nhớ trung gian **O(n)** phần tử
- Có độ ổn định cao (không bị ảnh hưởng bởi thứ tự ban đầu của dữ liệu)

Thuật toán “Sắp xếp nhanh” (Quick sort Algorithm)

- Quick sort cũng là một thuật toán “chia để trị”
- Ý tưởng:
  - Chia dãy cần sắp thành 2 phần
  - Cách “chia” của Quick sort khác với cách chia của Merge sort: ½ dãy bên trái chứa các giá trị nhỏ hơn ½ dãy bên phải.
  - Thực hiện việc sắp xếp trên từng dãy con **(đệ qui)**

Quick sort Algorithm
Minh họa chương trình

```
void QuickSort(int a[], int Left, int Right)
{
    // Chỉ xử lý khi dãy có > 1 phần tử
    if (Left < Right) {
            int m1, m2;
            // chia dãy (Left, Right) thành 2 dãy con
            Partition(a, Left, Right, m1, m2);
            QuickSort(a, Left, m1);          // dãy con bên trái
            QuickSort(a, m2, Right);         // dãy con bên phải
    }
}
```

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

Chọn 1 phần tử làm "chuẩn", thường ta chọn phần tử giữa của dãy

4	**12**	**10**	**8**	**5**	**2**	**11**	**7**	**3**

5

Phần tử "chuẩn"

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

Bắt đầu so sánh các phần tử từ 2 đầu của dãy với phần tử chuẩn…

4	**12**	**10**	**8**	**5**	**2**	**11**	**7**	**3**

low **high**

5

Phần tử “chuẩn”

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…tìm ra được 1 phần tử “sai vị trí” ở phía bên phải…

4	**12**	**10**	**8**	**5**	**2**	**11**	**7**	**3**

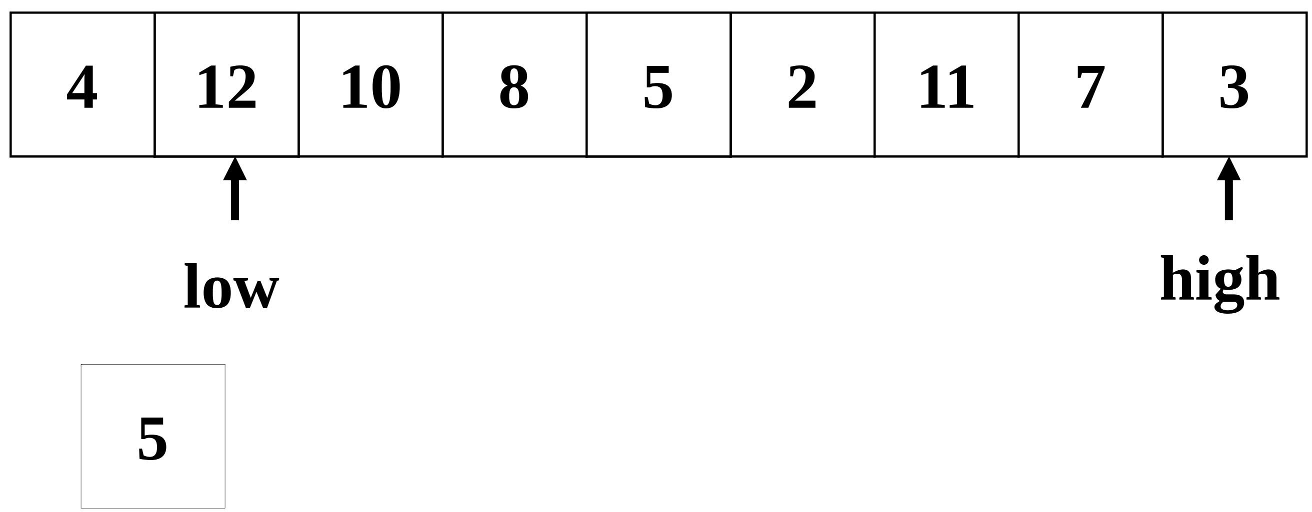
low **high**

5

Phần tử “chuẩn”

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…tìm ra được thêm 1 phần tử “sai vị trí” ở phía bên trái…



Phần tử “chuẩn”

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…hoán vị 2 phần tử cho nhau, và lại tiếp tục…

4	**3**	**10**	**8**	**5**	**2**	**11**	**7**	**12**

low **high**

5

Phần tử "chuẩn"

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…tìm thấy 2 phần tử “sai vị trí” mới…

4	**3**	**10**	**8**	**5**	**2**	**11**	**7**	**12**

low **high**

5

Phần tử “chuẩn”

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…hoán vị cho nhau, và tiếp tục…

4	**3**	**2**	**8**	**5**	**10**	**11**	**7**	**12**

low **high**

5

Phần tử “chuẩn”

Quick sort Algorithm
Cách chia thành 2 dãy con (Partition)

…low > high → kết thúc quá trình phân chia

½ dãy bên trái ½ dãy bên phải

4	**3**	**2**	**5**	**8**	**10**	**11**	**7**	**12**

high (m1) **low (m2)**

Quick sort Algorithm Partition – Minh họa chương trình

```
void  Partition(int a[], int Left, int Right, int &m1, int &m2)
{
    int Pivot = a[(Left+Right)/2];       // phần tử "chuẩn"
    int low = Left, high = Right;
    while (low < high) {
            while (a[low] < Pivot) low++;
            while (a[high] > Pivot) high--;
            if (low <= high)  {
                "Hoán vị a[low] và a[high]"
                low++; high--;
            }
    }
    m1 = high; m2 = low;    // 2 dãy con (Left – m1), và (m2 – Right)
}
```

Quick sort Algorithm
Cách chọn phần tử Pivot

Đánh giá thuật toán (Quick sort Algorithm)

- Chi phí trung bình **$O(n*\log_2 n)$**
- Chi phí cho trường hợp xấu nhất **$O(n^2)$**
- Chi phí tùy thuộc vào cách chọn phần tử "chuẩn":
  - Nếu chọn được phần tử có giá trị trung bình, ta sẽ chia thành 2 dãy bằng nhau;
  - nếu chọn nhằm phần tử nhỏ nhất (hay lớn nhất) → $O(n^2)$