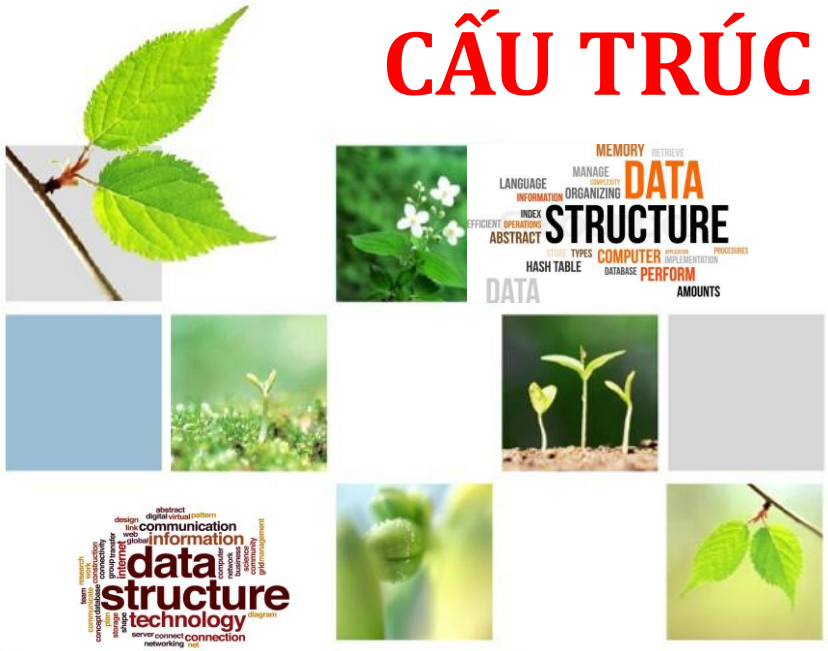
BUỔI 04: TÌM KIẾM & SẮP XẾP (tt)

LÝ THUYẾT

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

IT003.N210



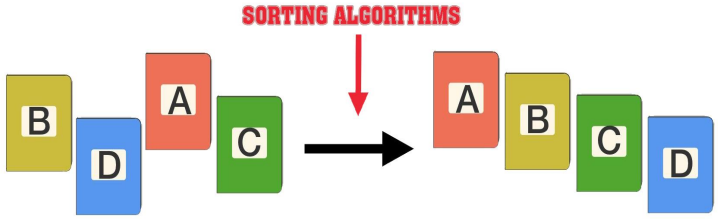
Giảng viên: Ths Đặng Văn Em

Email: vanem@uit.edu.vn

Điện thoại: 0966661006

CÁC GIẢI THUẬT SẮP XẾP:

1) Thuật toán sắp xếp vun đống (Heap sort).
2) Thuật toán sắp xếp nhanh (Quick sort)
3) Thuật toán sắp xếp trộn (Merge sort)
4) Cấu trúc priority queue.
5) Bài tập chương.



GIẢI THUẬT SẮP XẾP VUN ĐỐNG (HEAP SORT):

- **Từ khóa: Heap sort**
- **Phân tích:** Trong phương pháp chọn trực tiếp (Selection Sort), mỗi lần chọn phần tử cực tiểu theo quan hệ $\Re$ đều không tận dụng được các kết quả so sánh trước đó → độ phức tạp theo phép so sánh là $\mathbf{O(n^2)}$.

→ Tận dụng kết quả so sánh bằng cấu trúc **Heap**

GIẢI THUẬT SẮP XẾP VUN ĐỐNG (HEAP SORT):

Ý tưởng:

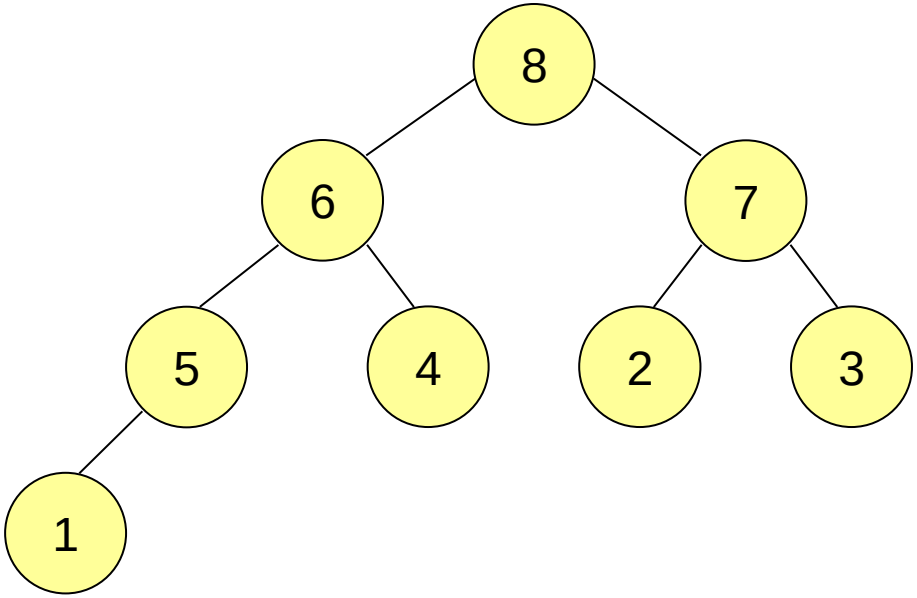
- Xây dựng cấu trúc Heap:
  - Là một cây nhị phân hoàn chỉnh
  - Nếu giá trị khóa của nút cha và hai nút con lần lượt là K, K_1, K_2, thì:

$$\begin{cases} K_1 \ \mathfrak{R} \ K \\ K_2 \ \mathfrak{R} \ K \end{cases}$$

❖PHƯƠNG PHÁP VUN ĐỐNG

Ý tưởng:

Ví dụ quan hệ $\mathfrak{R}$ là **<**



❖PHƯƠNG PHÁP VUN ĐỐNG

Ý tưởng:

- Xây dựng cấu trúc Heap:
  - Nếu dùng mảng A để biểu diễn cấu trúc Heap, A có đặc điểm:
    - ✓ A_0 là cực đại theo $\Re$
    - ✓ A_{i*2+1} $\Re$ A_i và $A_{(i+1)*2}$ $\Re$ A_i
    - ✓ A_{i*2+1} và $A_{(i+1)*2}$ là phần tử liên đới với A_i

❖PHƯƠNG PHÁP VUN ĐỐNG

Ý tưởng:

- Sắp xếp dựa trên cấu trúc Heap:
  - Hoán đổi vị trí của A_0 và A_{n-1}, đưa giá trị cực đại về cuối dãy ⇒ dãy **A** trong bước tiếp theo đã giảm được một phần tử, còn lại là $A_0..A_{n-2}$
  - Bắt đầu từ A_0, điều chỉnh các phần tử trong $A_0..A_{n-2}$ để đảm bảo tính chất của Heap.
  - Thực hiện hoán đổi A_0 và điều chỉnh dãy mới đến khi dãy A chỉ còn lại một phần tử

❖PHƯƠNG PHÁP VUN ĐỐNG

Thuật toán:

Đầu vào: $A=\{a_0, a_1, .., a_{n-1}\}$ chưa có thứ tự $\mathfrak{R}$

Đầu ra: $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\mathfrak{R}$

❖PHƯƠNG PHÁP VUN ĐỐNG

Thuật toán: (Sắp xếp)

```
buildHeap(A, n)
while n > 1
  n ← n-1
  swap(A[0], A[n])
  heapify(A, 0, n)
```

❖PHƯƠNG PHÁP VUN ĐỐNG

Thuật toán: (tạo Heap)

```
//chương trình con buildHeap(A, n)
i ← n/2 - 1
while i ≥ 0
  heapify(A, i, n)
  i ← i-1
```

❖PHƯƠNG PHÁP VUN ĐỐNG

Thuật toán: (điều chỉnh Heap)

```
// chương trình con heapify(A, k, n)
j ← 2*k+1
while j < n
  if j+1 < n then
     if a[j] ℜ a[j+1] then j ← j+1 end if
  if (a[j] ℜ a[k]) then return end if
  swap(a[k], a[j])
  k = j
  j = 2*k+1
end while
```

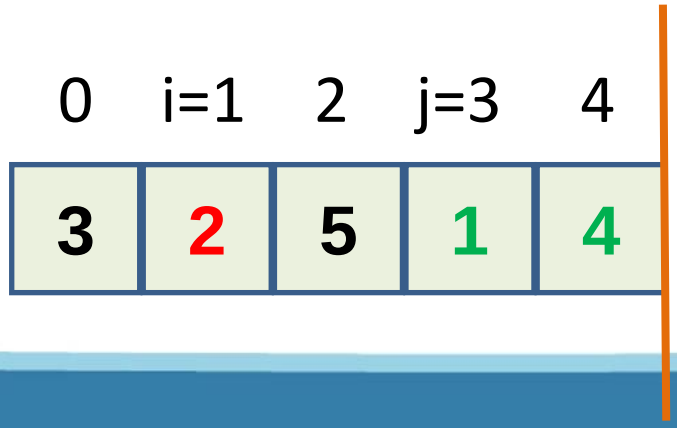
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Tạo Heap: i=1

heapify: j = 3, j+1 = 4



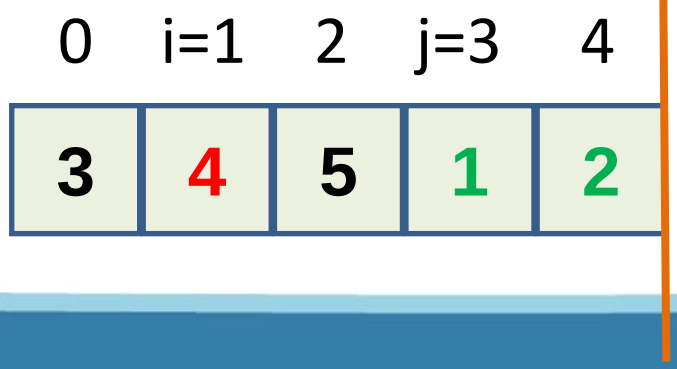
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Tạo Heap: i=1

heapify: j = 3, j+1 = 4, hoán đổi A[1] và A[4]



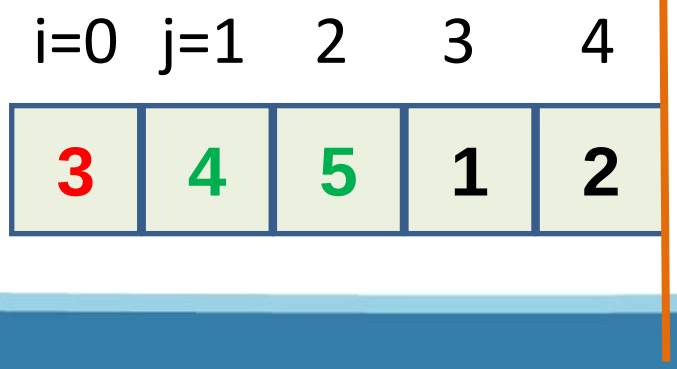
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Tạo Heap: i=0

heapify: j = 1, j+1 = 2



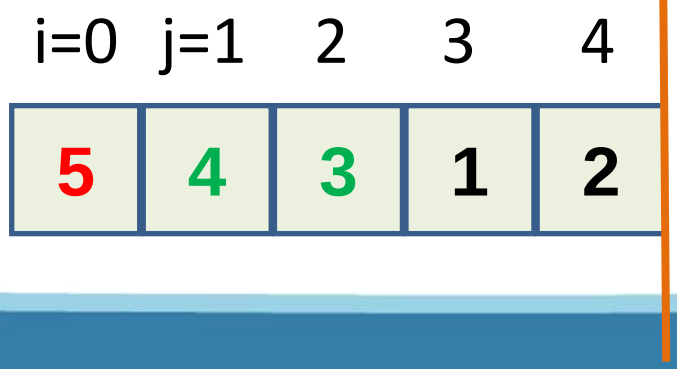
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Tạo Heap: i=0

heapify: j = 1, j+1 = 2, hoán đổi A[0] và A[2]



❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=5

Hoán đổi: A[0] và A[4]

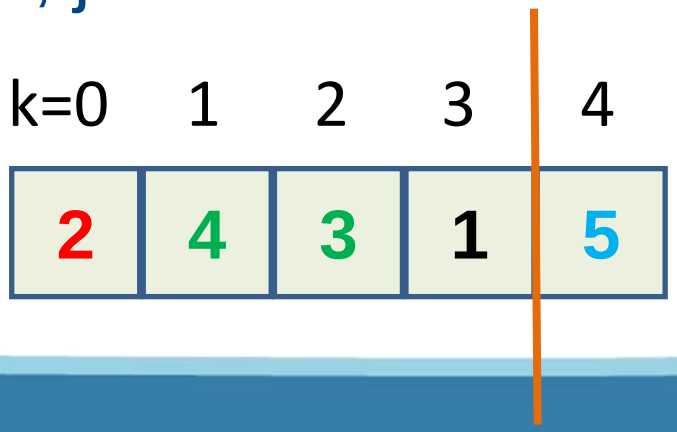
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp **<** (tăng dần)

Sắp xếp: n=4

heapify: k=0, j=1, j+1=2



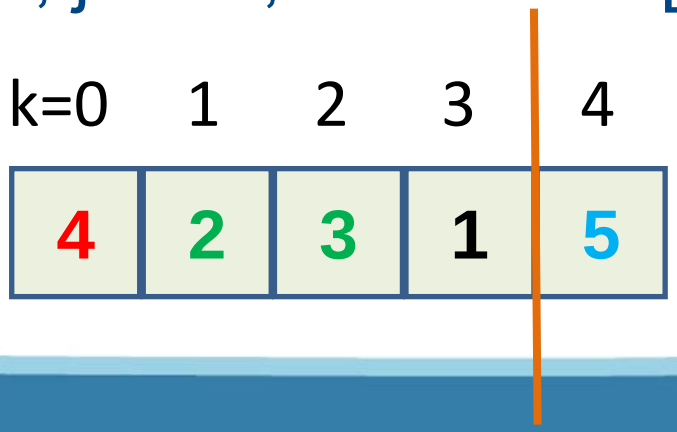
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=4

heapify: k=0, j=1, j+1=2, hoán đổi A[0] và A[1]



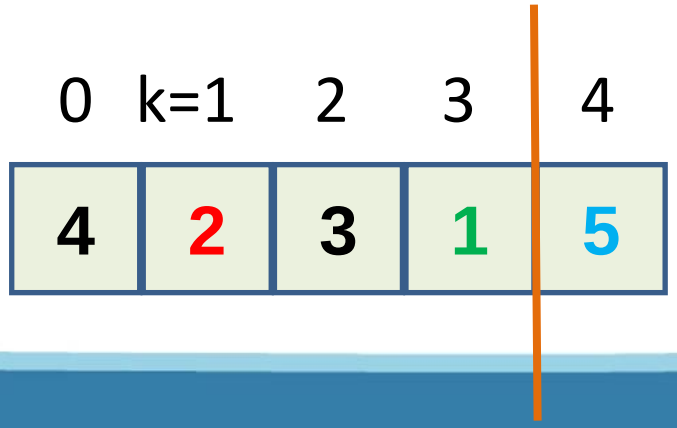
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp **<** (tăng dần)

Sắp xếp: n=4

heapify: k=1, j=3



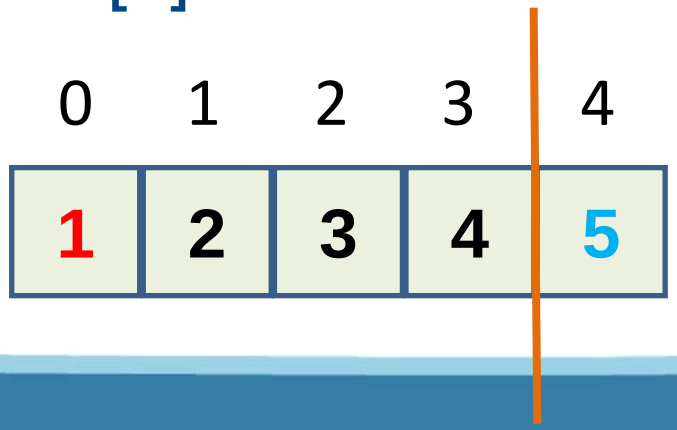
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=4

Hoán đổi A[0] và A[3]



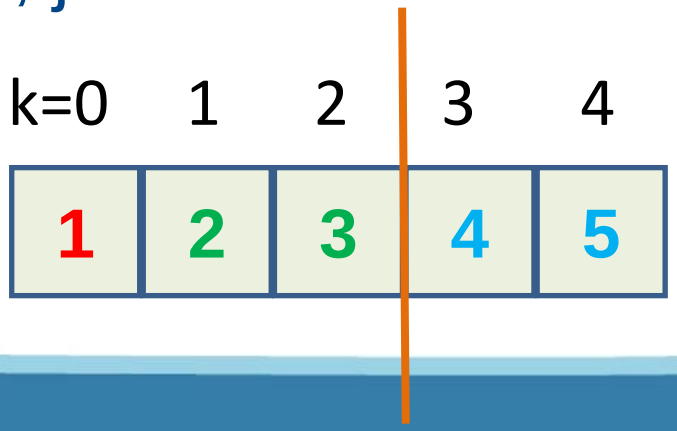
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=3

heapify k=0, j=1, j+1=2



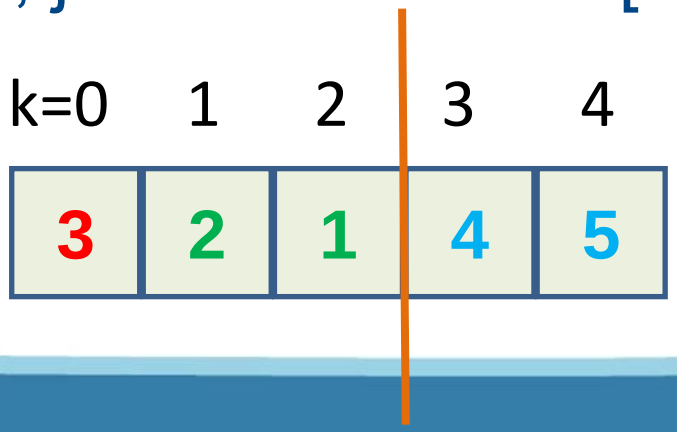
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=3

heapify k=0, j=1, j+1=2 hoán đổi A[0] và A[2]



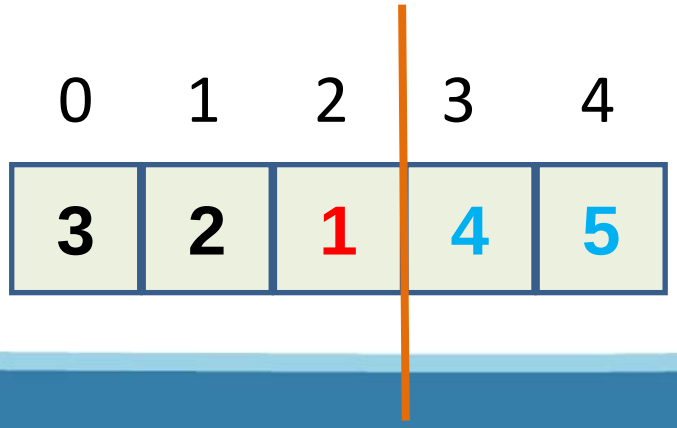
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp **<** (tăng dần)

Sắp xếp: n=3

heapify k=2,



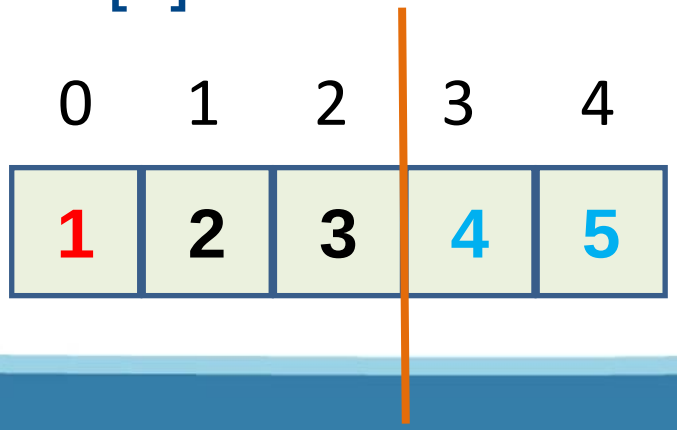
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=3

Hoán đổi A[0] và A[2]



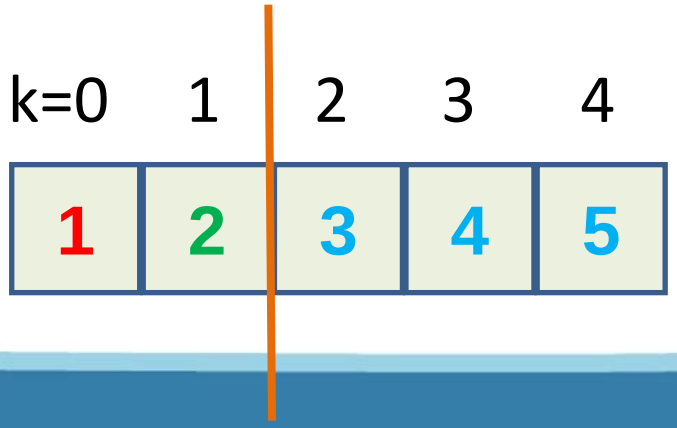
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=2

heapify: k=0, j=1



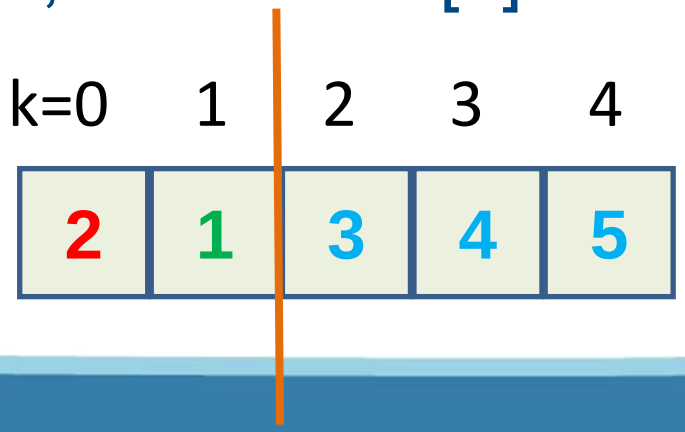
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp **<** (tăng dần)

Sắp xếp: n=2

heapify: k=0, j=1, hoán đổi A[0] và A[1]



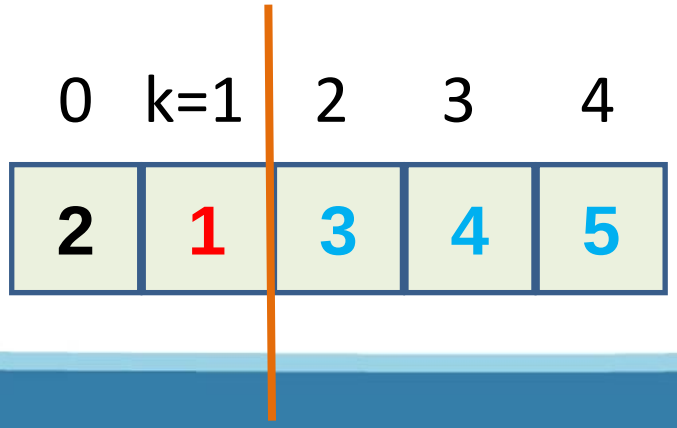
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=2

heapify: k=1,



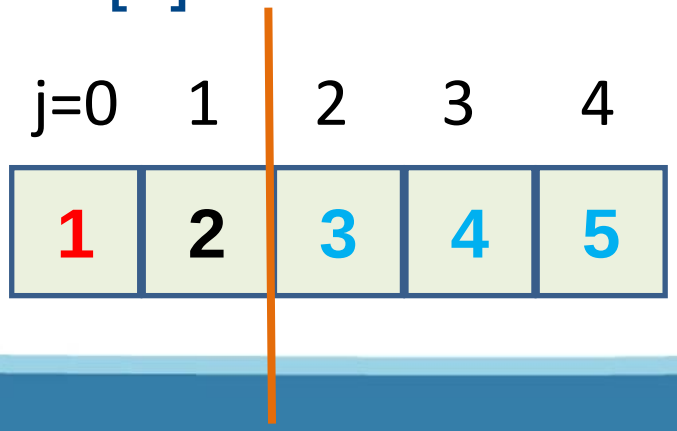
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=2

Hoán đổi A[0] và A[1]



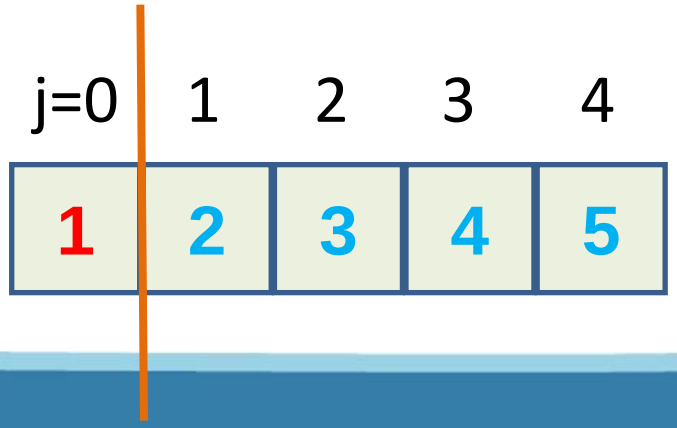
❖PHƯƠNG PHÁP VUN ĐỐNG

Quá trình tính toán:

Giả sử A={3,2,5,1,4} và thứ tự cấp sắp xếp <
(tăng dần)

Sắp xếp: n=1

Dừng



❖PHƯƠNG PHÁP VUN ĐỐNG

Cài đặt: Trên mảng, Giả sử thứ thự là **<** (tăng dần)

❖PHƯƠNG PHÁP VUN ĐỐNG

```
void heapify(int *a, int k, int n) {
  int j = 2*k+1;
  while (j < n) {
          if (j + 1 < n)
              if (a[j] < a[j + 1]) j = j
  + 1;
          if (a[k] >= a[j]) return;
          swap(a[k], a[j]);
          k = j; j = 2*k + 1;
  }
}
```

❖PHƯƠNG PHÁP VUN ĐỐNG

```
void buildHeap(int *a, int n) {
  int i;
  i = n/2 - 1;
  while (i >= 0) {
          heapify(a, i, n);
          i--;
  }
}
```

❖PHƯƠNG PHÁP VUN ĐỐNG

```
void heapSort(int *a, int n) {
  buildHeap(a, n);
  while (n > 0) {
          n = n - 1;
          swap(a[0], a[n]);
          heapify(a, 0, n);
  }
}
```

III. CÁC GIẢI THUẬT SẮP XẾP

❖PHƯƠNG PHÁP VUN ĐỐNG

Đánh giá: phương pháp vun đống có thời gian sắp xếp ổn định.

	TỐT NHẤT (thứ tự ngược)	**TRUNG BÌNH** (chưa có thứ tự)	**XẤU NHẤT** (đúng thứ tự)
Theo phép so sánh	O(nlogn)	O(nlogn)	O(nlogn)
Theo phép gán giá trị khóa	O(nlogn)	O(nlogn)	O(nlogn)

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

- **Từ khóa: Quick sort**
- **Phân tích:** Giả sửa dãy **A** đã có thứ tự $\Re$, với A_i bất kỳ:
  - $A_j \Re A_i \ \forall \ j < I$
  - $A_i \Re A_j \ \forall \ j > I$
  - $A_0, .., A_{i-1}$ và $A_{i+1}, .., A_{n-1}$ đều có thứ tự $\Re$

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

❖ **Ý tưởng:** Áp dụng chiến thuật chia để trị

- Nếu A có không quá 1 phần tử → đã có thứ tự.
- Chọn phần tử chốt (pivot) **x**
- Chia dãy **A** thành hai phần:
  - Phần trước chứa A_i sao cho $A_i \Re x$
  - Phần sau chứa A_j sao cho $x \Re A_j$
- Sắp xếp hai dãy $A_0,..,A_{k-1}$ và $A_{k+1},..,A_{n-1}$ tương tự.

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

❖ **Giải thuật:**

Bước 1: Nếu left ≥ right //dãy có ít hơn 2 phần tử
Kết thúc; //dãy đã được sắp xếp

Bước 2: Phân hoạch dãy a_{left} … a_{right} thành các đoạn:

a_{left}.. a_j, a_{j+1}.. a_{i-1}, a_i.. a_{right}

- *Đoạn 1* $\leq x$
- *Đoạn 2:* a_{j+1}.. a_{i-1} $= x$
- *Đoạn 3:* a_i.. a_{right} $\geq$ x

Bước 3: Sắp xếp đoạn 1: a_{left}.. a_j

Bước 4: **Sắp xếp đoạn 3**: a_i.. a_{right}

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

❖ **Giải thuật:**

Bước 1 : Chọn tùy ý phần tử a[k] trong dãy là giá trị mốc (l ≤ k ≤ r):

x = a[k]; i = l; j = r;

Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ:

Bước 2a : Trong khi (a[i]<x) i++;

Bước 2b : Trong khi (a[j]>x) j--;

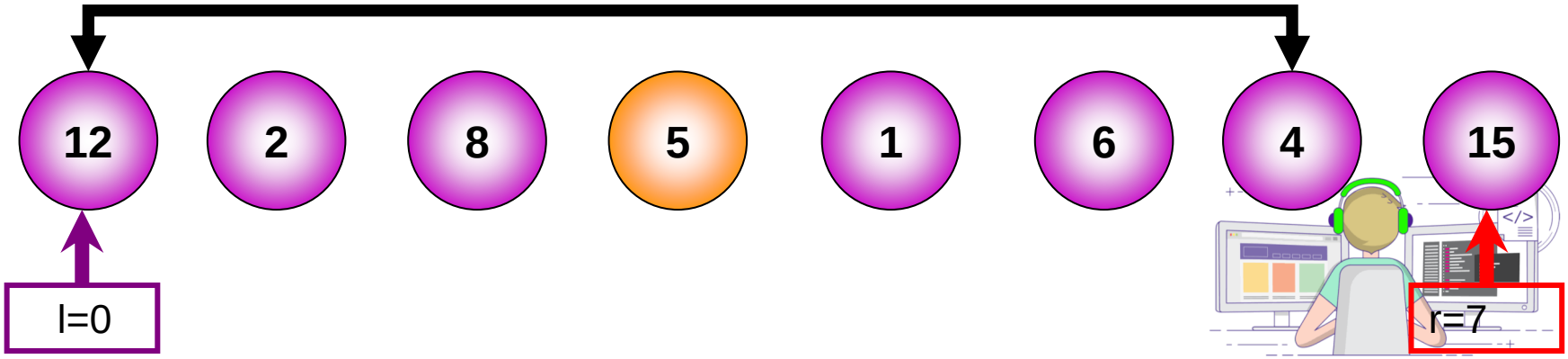
Bước 2c : Nếu i< j Swap(a[i],a[j]);

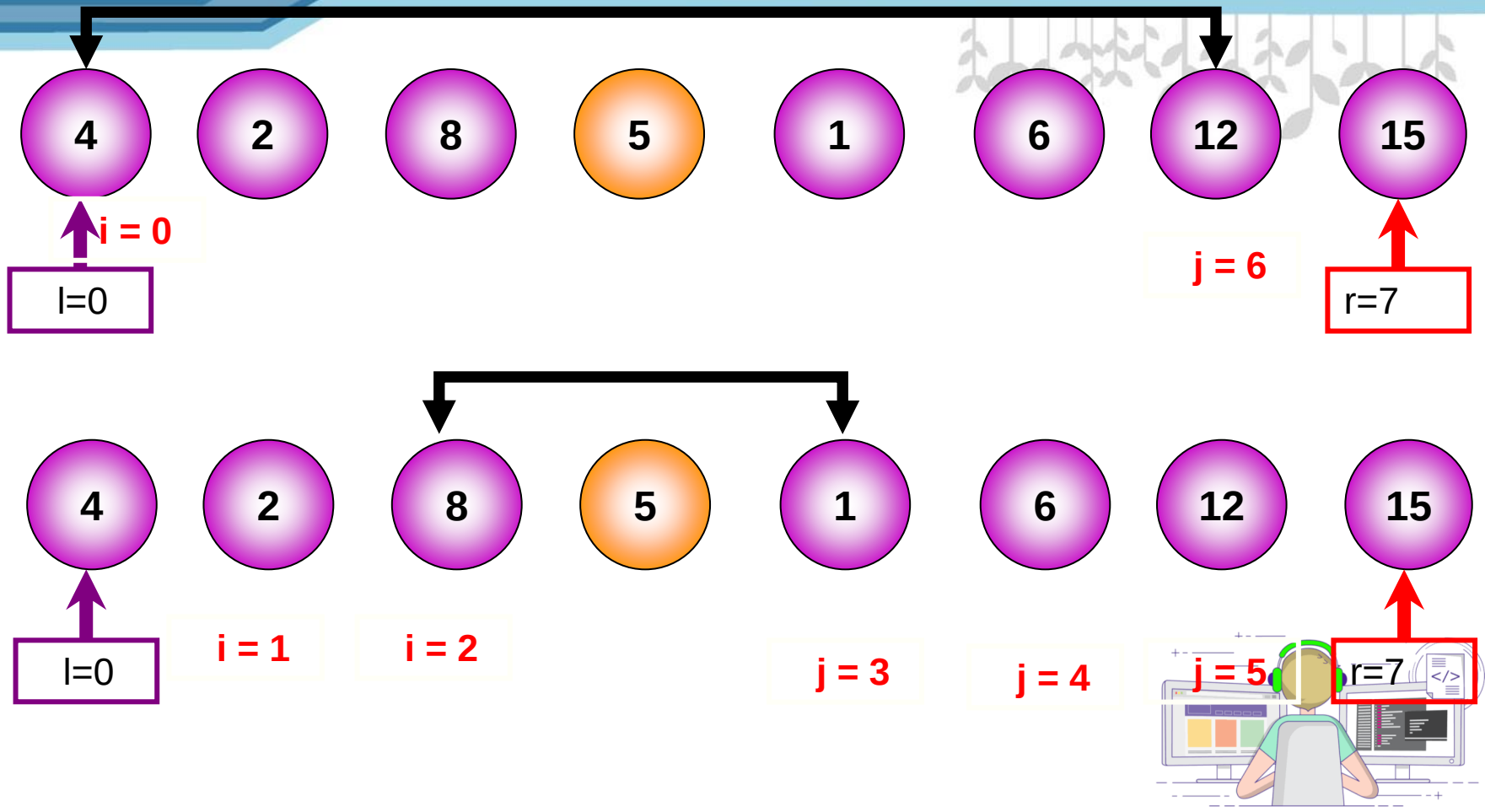
Bước 3 : Nếu i < j: Lặp lại Bước 2.
Ngược lại: **Dừng**

Cho dãy số a:

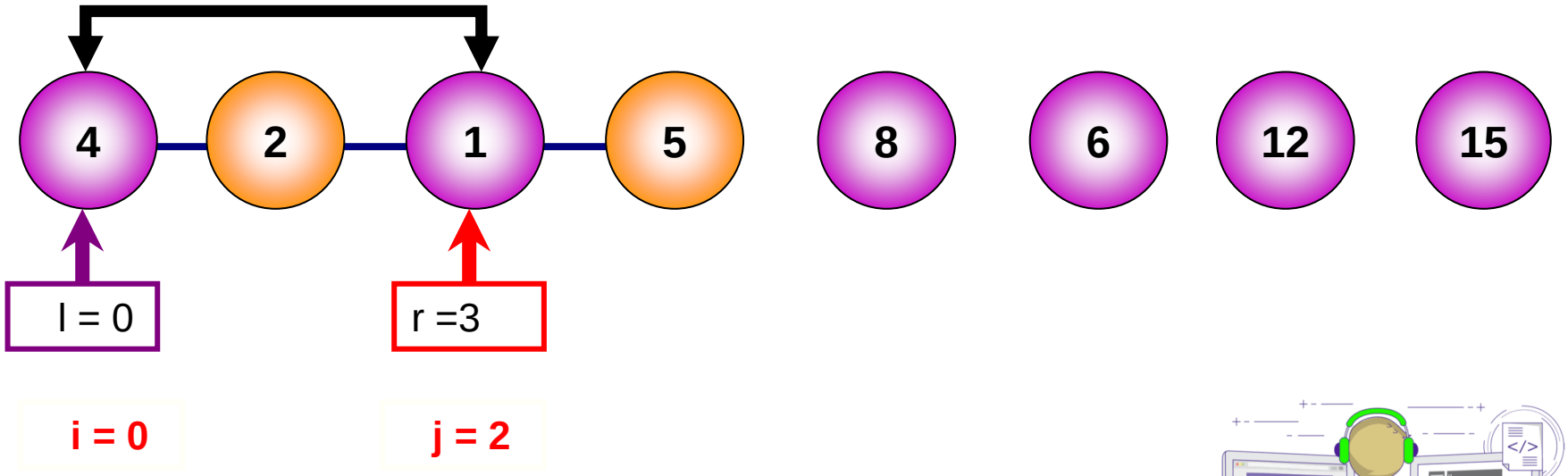
12 2 8 5 1 6 4 15

Phân hoạch đoạn l =0, r = 7: x = a[3] = 5

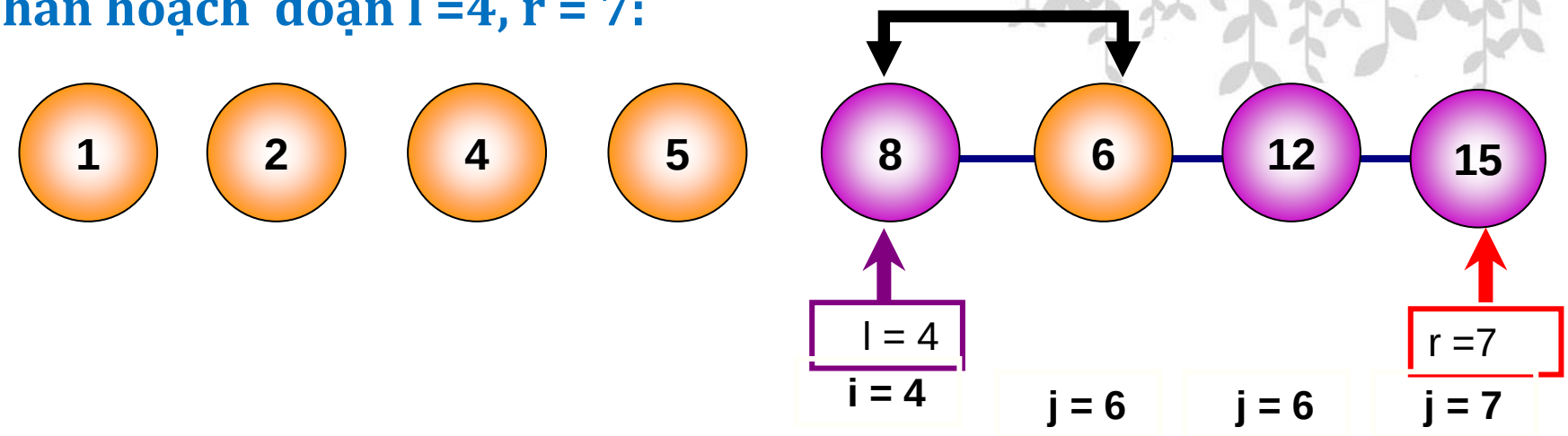


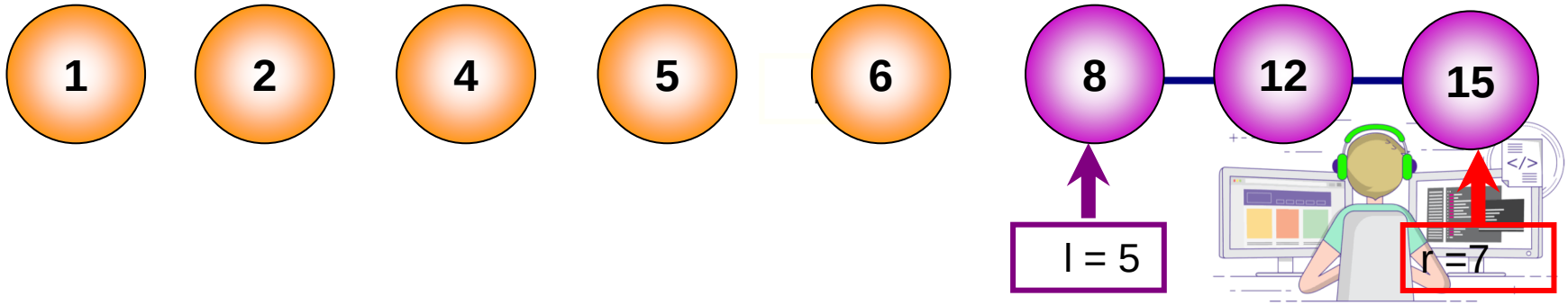

4
2
8
5
1
6
12
15
i = 0
l=0
j = 6
r=7
4
2
8
5
1
6
12
15
l=0
i = 1
i = 2
j = 3
j = 4
j = 5
r=7

➢ **Phân hoạch đoạn l = 0, r = 2:**

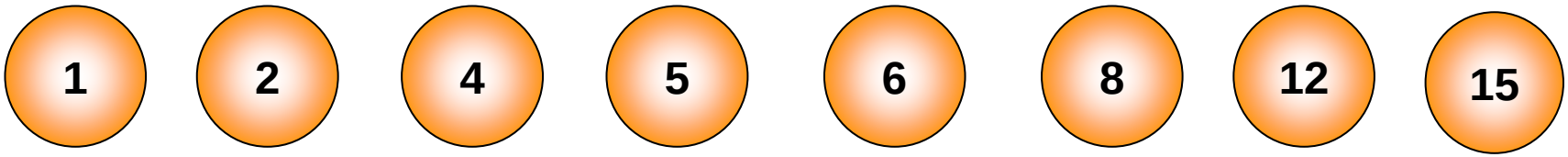


- **Phân hoạch đoạn l =4, r = 7:**





- **Phân hoạch đoạn l = 6, r = 7:**



GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

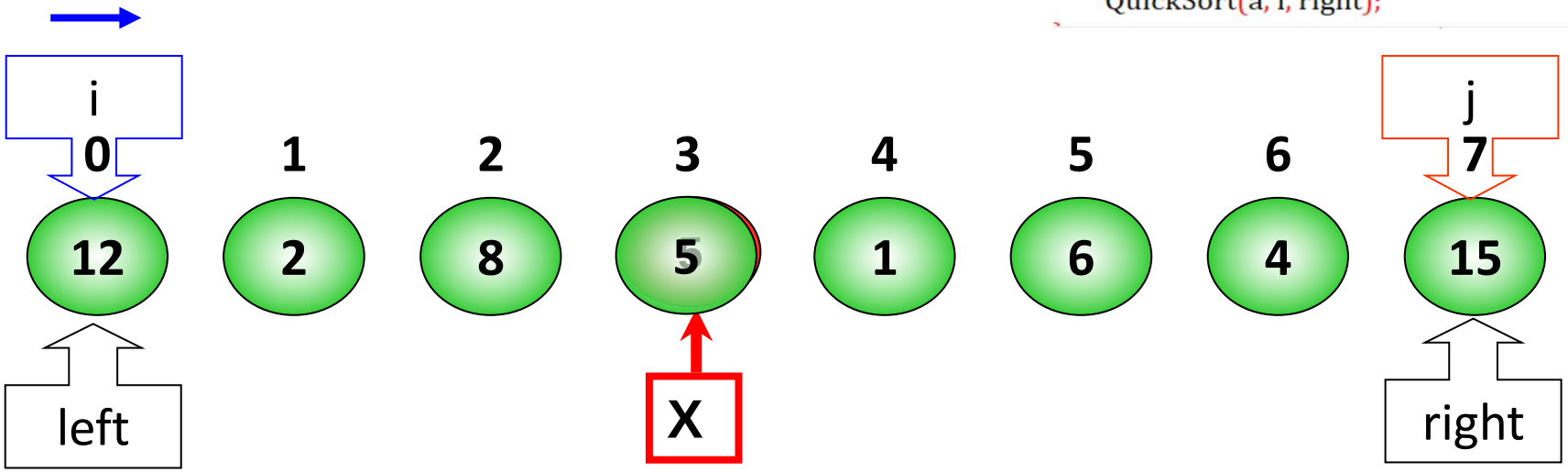
❖ **Cài đặt minh họa:**

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

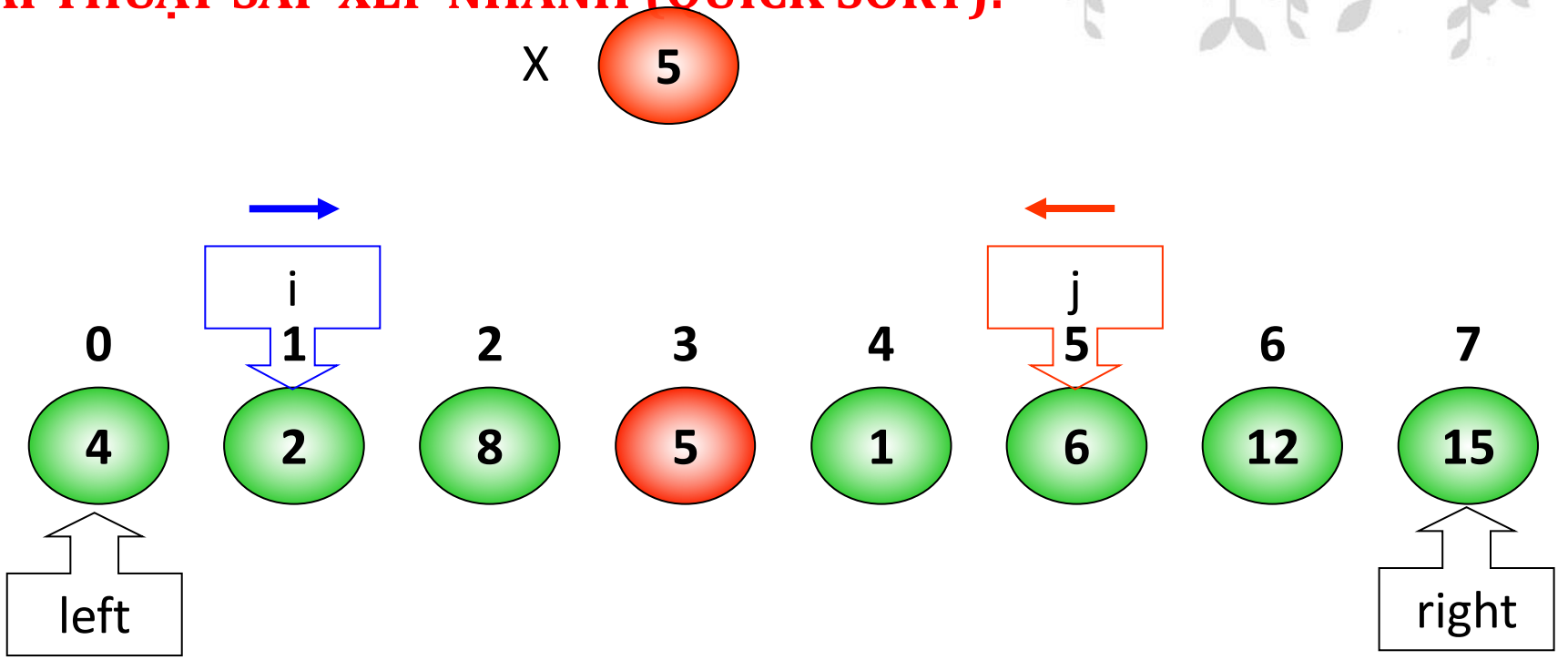
GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

Phân hoạch đoạn [0,7]

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

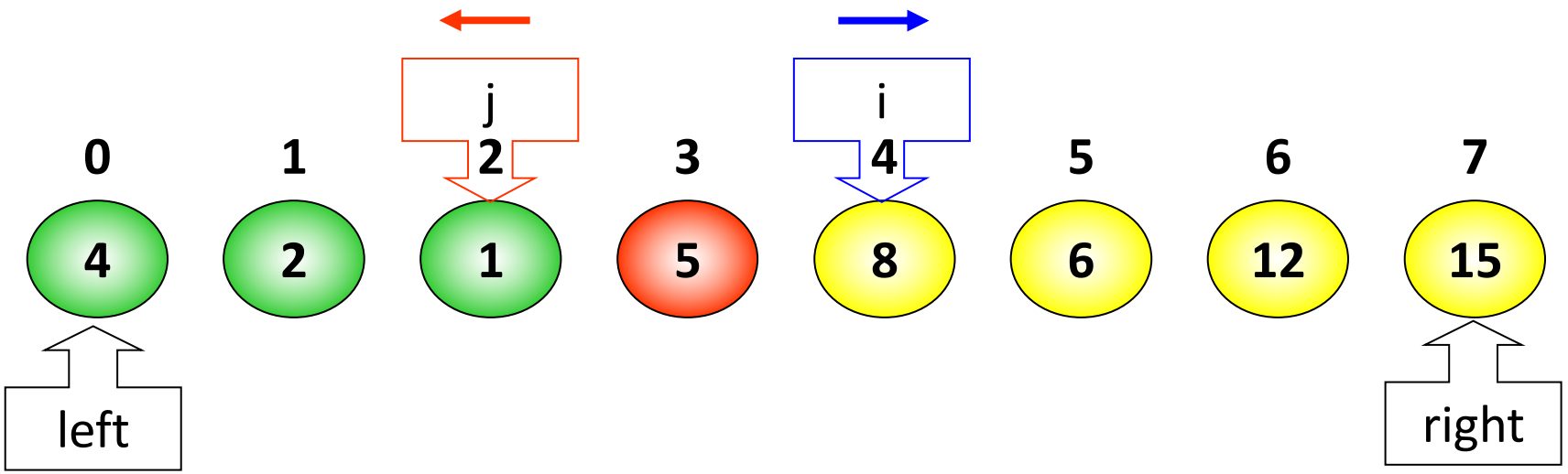


GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):



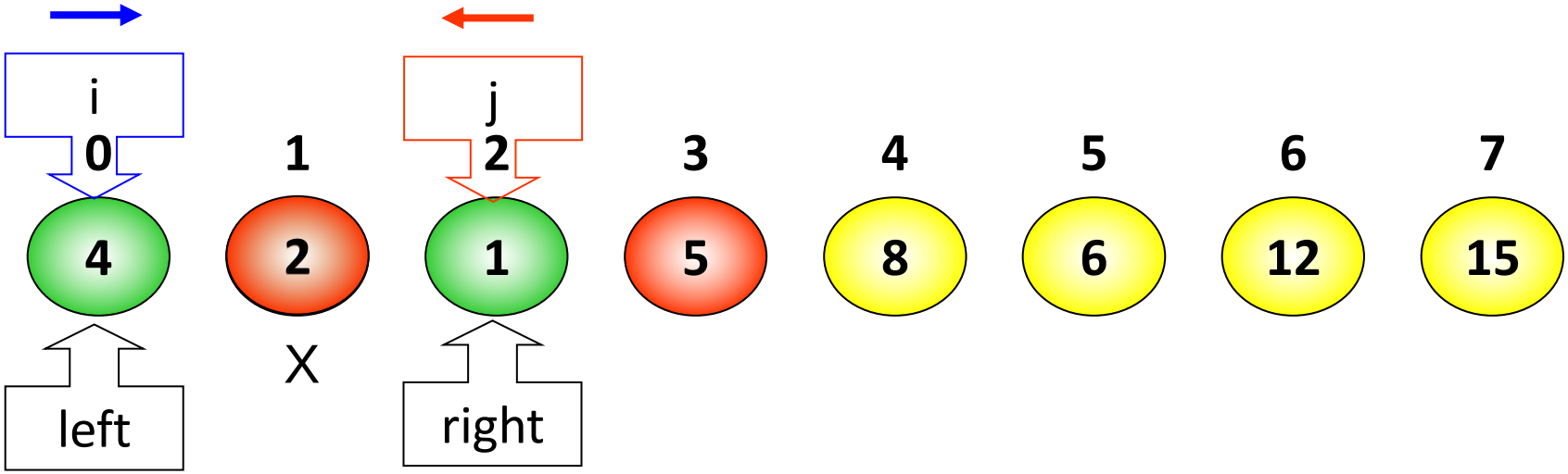
GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

Phân hoạch đoạn [0,2]



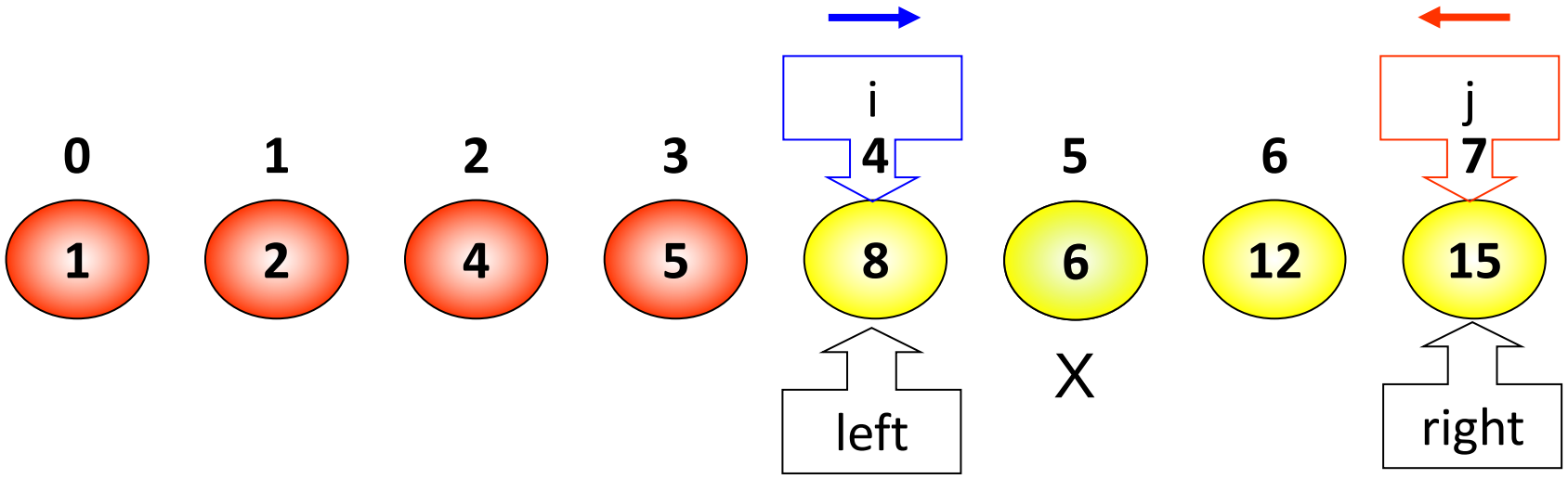
GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

Phân hoạch đoạn [0,2]

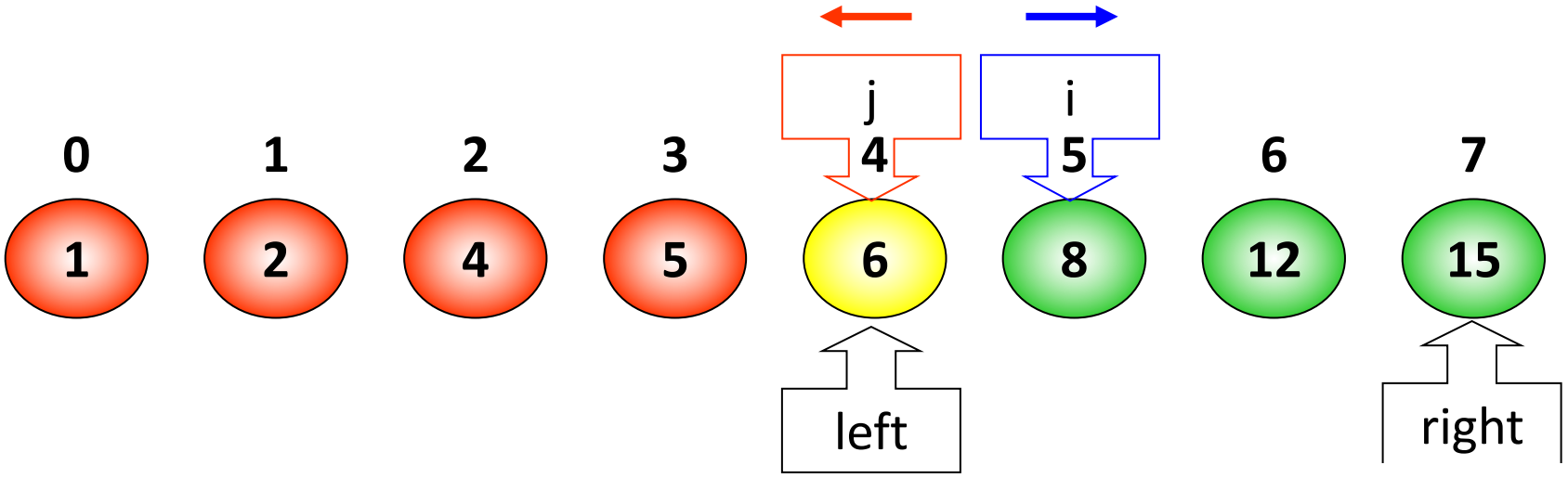


GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

Phân hoạch đoạn [4,7]

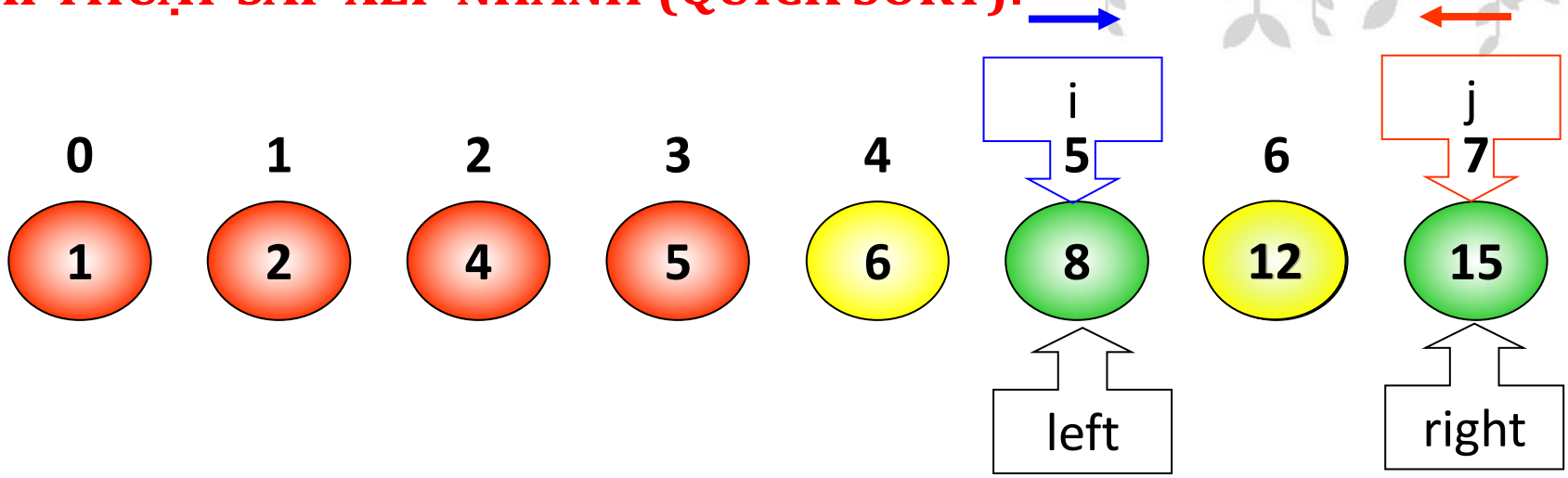


GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):



Phân hoạch đoạn [5,7]

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):



Phân hoạch đoạn [5,7]

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

0	1	2	3	4	5	6	7
1	2	4	5	6	8	12	15

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

- Thực hiện từng bước giải thuật sắp xếp QUICK SORT mảng

A[] = 23 29 31 3 5 7 11 13 17 9 1 41 43 47 53 59

A[] = 10 6 7 20 13 17 9 21 29 11 1 41 42 27 5 60

GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT):

❖ **Đánh giá:**

	TỐT NHẤT (đúng thứ tự)	TRUNG BÌNH (chưa có thứ tự)	XẤU NHẤT (thứ tự ngược)
Theo phép so sánh	**O(nlogn)**	**O(nlogn)**	**$O(n^2)$**
Theo phép gán giá trị khóa	**O(nlogn)**	**O(nlogn)**	**$O(n^2)$**

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

- **Từ khóa: Merge sort**
- **Phân tích:** Giả sử dãy A_1 và A_2 có **k** phần tử đã có thứ tự $\mathfrak{R}$, khi đó, có thể tạo dãy **A** có thứ tự $\mathfrak{R}$ gồm các phần tử của A_1 và A_2 với:
  - Chi phí thời gian là **O(k)**
  - Trộn từng theo thứ tự từ đầu danh sách.
  - Phần tử trên hai danh sách được trộn theo thứ tự $\mathfrak{R}$

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

- **Ý tưởng:** Áp dụng chiến lược chia để trị: Danh sách có 1 phần tử luôn có thứ thự. Để sắp xếp danh sách **A**:
  - Chia **A** thành hai danh sách $\mathbf{A_1}$ và $\mathbf{A_2}$
  - Sắp xếp $\mathbf{A_1}$ và $\mathbf{A_2}$ theo thứ tự $\mathfrak{R}$
  - Trộn $\mathbf{A_1}$ và $\mathbf{A_2}$ theo thứ tự $\mathfrak{R}$

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

❖ **Thuật toán**: MergeSort(A)

Đầu vào: $A=\{a_0, a_1, .., a_{n-1}\}$ chưa có thứ tự $\Re$

Đầu ra: $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\Re$

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

❖ **Thuật toán**:

Bước 1: Khởi tạo i=0,j=0,k=0 cho 3 mảng A[], B[], C[].

Bước 2: Tại mỗi bước tại chỉ số (i<n và j<m) ta chọn min B[i]C[j] lưu vào C[k]. Chuyển sang bước 4.

Bước 3: k=k+2 quay lại bước 2

Bước 4: Sao chép các giá trị còn lại từ các danh sách mà các chỉ số còn vi phạm (i<m hoặc j<m) vào mảng C[].

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

❖ **Mô phỏng thuật toán:**

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

❖ **Cài đặt thuật toán:**

```
void Distribute(int a[], int N, int &nb, int &nc, int k)
{
   int  i, pa, pb, pc;
   pa = pb = pc = 0;
   while (pa < N)
   {
      for (i=0; (pa<N) && (i<k); i++, pa++, pb++)
            b[pb] = a[pa];
      for (i=0; (pa<N) && (i<k); i++, pa++, pc++)
            c[pc] = a[pa];
   }
      nb = pb; nc = pc;
}
```

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

```
void Merge(int a[],int nb, int nc,int k)
{  int p, pb, pc, ib, ic, kb, kc;
   p=pb=pc=0;
   ib=ic=0;
   while((nb>0)&&(nc>0))
   {
      kb=min(k,nb);
      kc=min(k,nc);
       if(b[pb+ib]<=c[pc+ic])
        {
           a[p++]=b[pb+ib];
           ib++;
           if(ib==kb)
           {
```

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

```
        {
        for(;ic<kc;ic++ a[p++]=c[pc+ic];
        pb+=kb; pc+=kc; ib = ic=0;
        nb-=kb; nc-=kc;
        }
    }
else
    {
      a[p++]=c[pc+ic]; ic++;
      if(ic==kc)
      {
        for(;ib<kb;ib++) a[p++]=b[pb+ib];
        pb+=kb;  pc+=kc; ib = ic=0;
        nb-=kb; nc-=kc;
      }
    }
```

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

```
void MergeSort(int a[], int N)
{
   int k;
   for (k = 1; k < N; k *= 2)
   {
      Distribute(a, N, nb, nc, k);
      Merge(a, nb, nc, k);
   }
}
```

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

- Thực hiện từng bước giải thuật sắp xếp MERGE SORT mảng

A[] = 3 15 7 1 13 17 19 23 9 31 11 41 43 4 53 29

A[] = 1 7 4 20 3 17 19 2 29 31 11 41 13 27 52 6

GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

Đánh giá:

- Trong mọi trường hợp độ phức tạp tính toán của phương pháp trộn là **O(nlogn)**
- Thích hợp cho các danh sách truy xuất tuần tự (file, danh sách đơn).
- Có thể thực hiện sắp xếp mà không cần nạp toàn bộ danh sách lên RAM (**External Sorting**)
- Trường hợp danh sách đã có những đoạn con có thứ tự → Trộn tự nhiên (**Natural Merge Sort**)

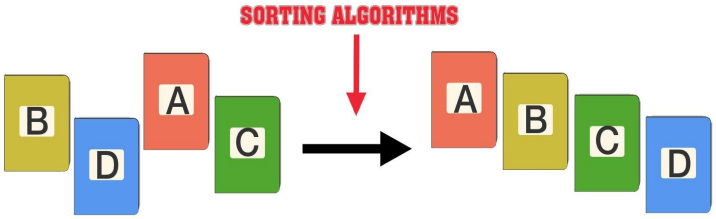
GIẢI THUẬT SẮP XẾP TRỘN (MERGE SORT):

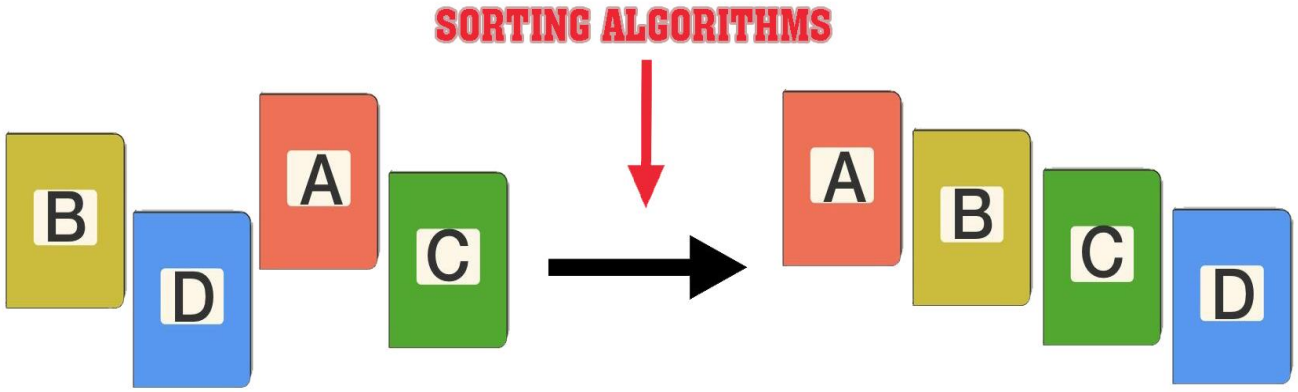
Đánh giá:

	TỐT NHẤT (đúng thứ tự)	TRUNG BÌNH (có thứ tự cục bộ)	XẤU NHẤT (không có thứ tự)
Theo phép so sánh	**O(1)**	**O(nlogn)**	**O(nlogn)**
Theo phép gán giá trị khóa	**O(1)**	**O(nlogn)**	**O(nlogn)**

CẤU TRÚC PRIORITY QUEUE

1) Cấu trúc priority queue.




SORTING ALGORITHMS
B
D
A
C
A
B
C
D


</>