CHƯƠNG 2: SẮP XẾP

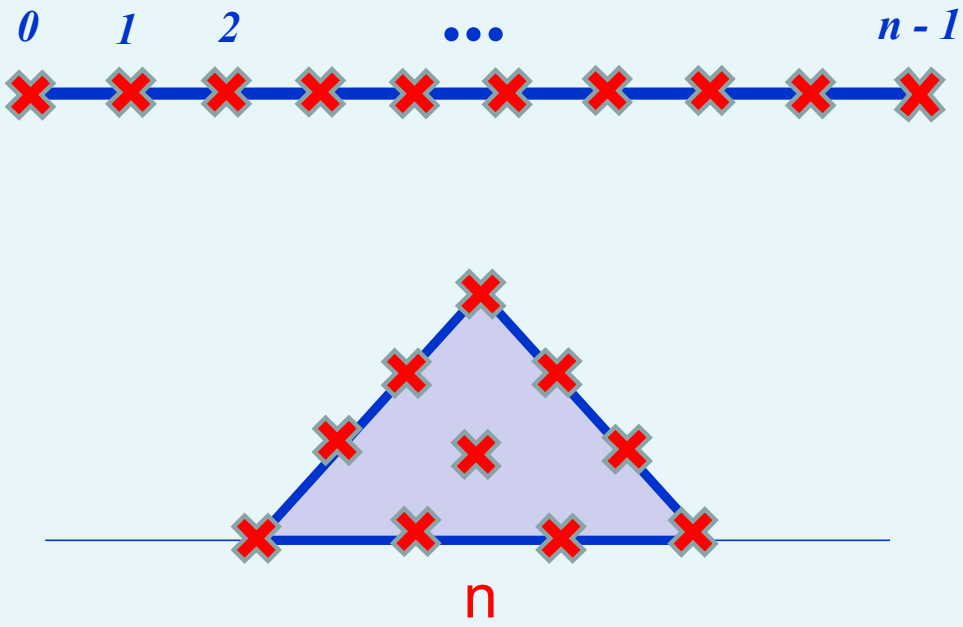
Tuần 5 – Sắp xếp vun đống (HEAP SORT)

Bộ môn CÔNG NGHỆ PHẦN MỀM
Khoa Công nghệ Thông tin & Truyền thông
ĐẠI HỌC CẦN THƠ

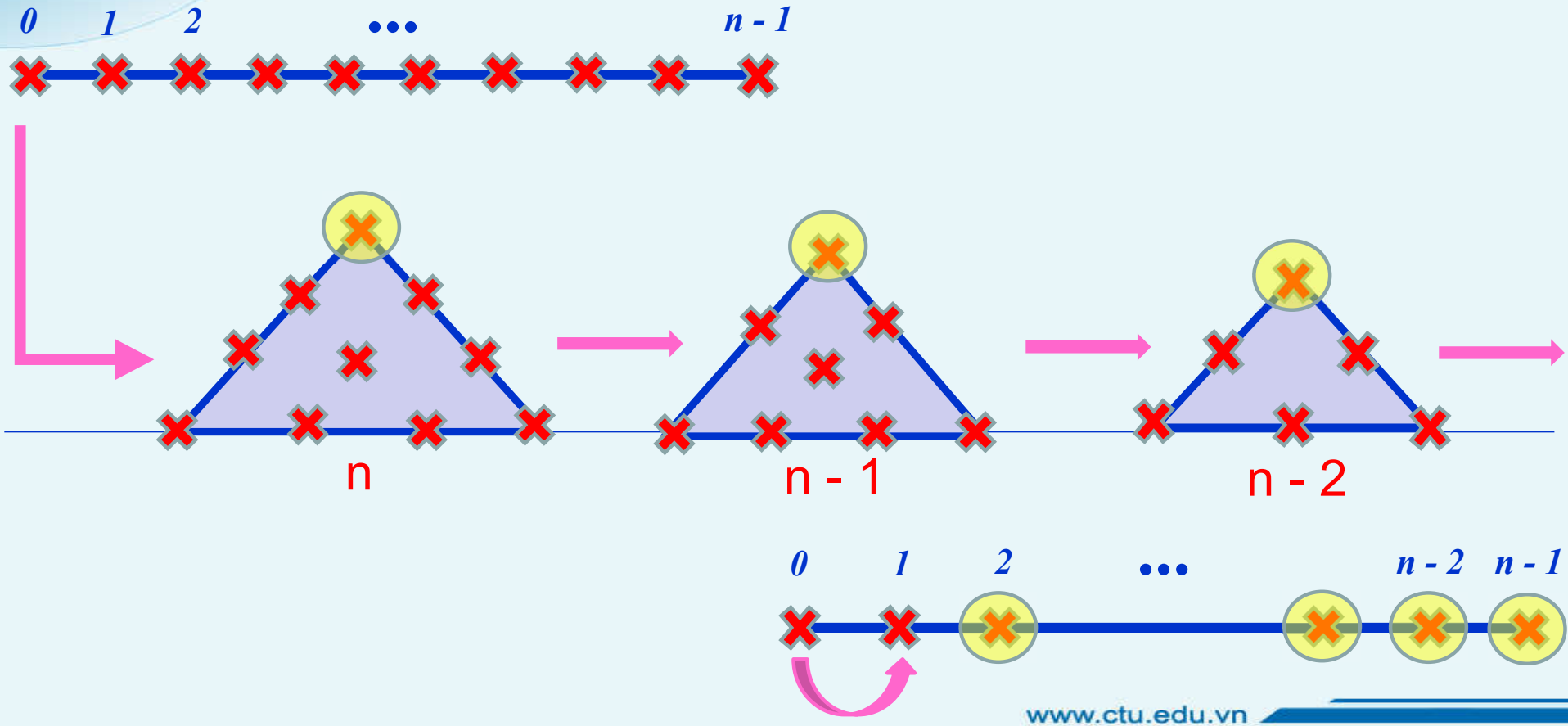
Các thuật toán Sắp xếp

- Các thuật toán sắp xếp đơn giản (độ phức tạp $O(n^2)$):
  - Sắp xếp **chọn** (Selection Sort)
  - Sắp xếp **xen** (Insertion Sort)
  - Sắp xếp **nổi bọt** (Bubble Sort)
- Các thuật toán sắp xếp phức tạp (độ phức tạp O(n logn))
  - Sắp xếp **phân đoạn/nhanh** (Quick Sort)
  - Sắp xếp **vun đống** (Heap Sort)
  - *Trường hợp dữ liệu đặc biệt (Bin Sort) (độ phức tạp O(n))*

Sắp xếp vun đống : Ý tưởng thuật toán



HeapSort: Ý tưởng



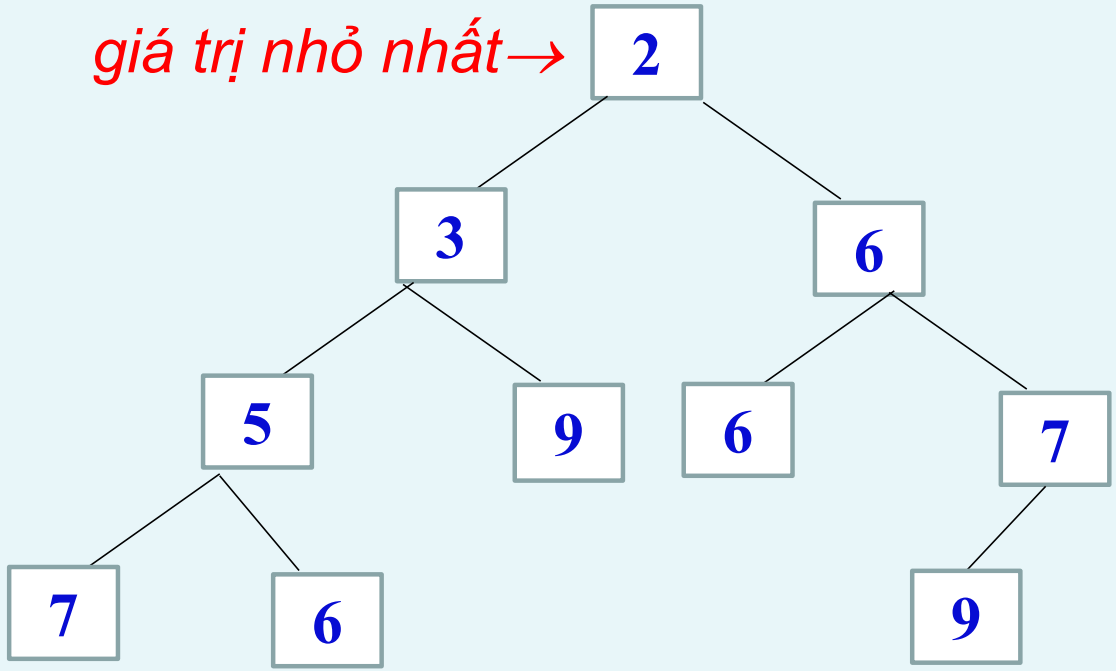
HeapSort: Định nghĩa Heap

HEAP = Cây sắp thứ tự bộ phận *(Đống)*

- Cây **nhị phân**
- Giá trị nút (khác nút lá) ***<u>không lớn hơn</u>*** giá trị các nút con của nó.

- ***Nhận xét***:
  - **Nút gốc** của cây sắp thứ tự bộ phận có *giá trị nhỏ nhất*.
  - Đây không phải là *cây tìm kiếm nhị phân*.

Ví dụ về Heap



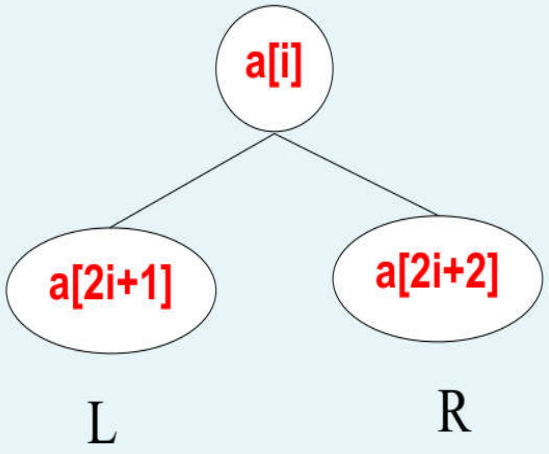
- Cây **nhị phân**
- Trị nút trong ***không lớn hơn*** trị nút con

HeapSort : Thuật toán

(1) Xem mảng là cây nhị phân:

- a[0] là **nút gốc**
- **Nút trong a[i]** có ***con trái* a[2i+1]** và ***con phải* a[2i+2]**
- Nút trong từ **a[0],…, a[(n-2)/2]** đều có 2 con

  (trừ nút a[(n-2)/2] có thể chỉ có 1 con trái khi n chẵn)



(2) Sắp xếp cây thành Heap: dùng thủ tục **PushDown**

(3) Hoán đổi nút gốc a[0] cho nút lá cuối

(4) Sắp xếp lại cây sau khi đã bỏ đi nút lá cuối thành Heap mới dùng **PushDown**

Lặp lại (3) và (4) cho đến khi cây chỉ còn 2 nút.

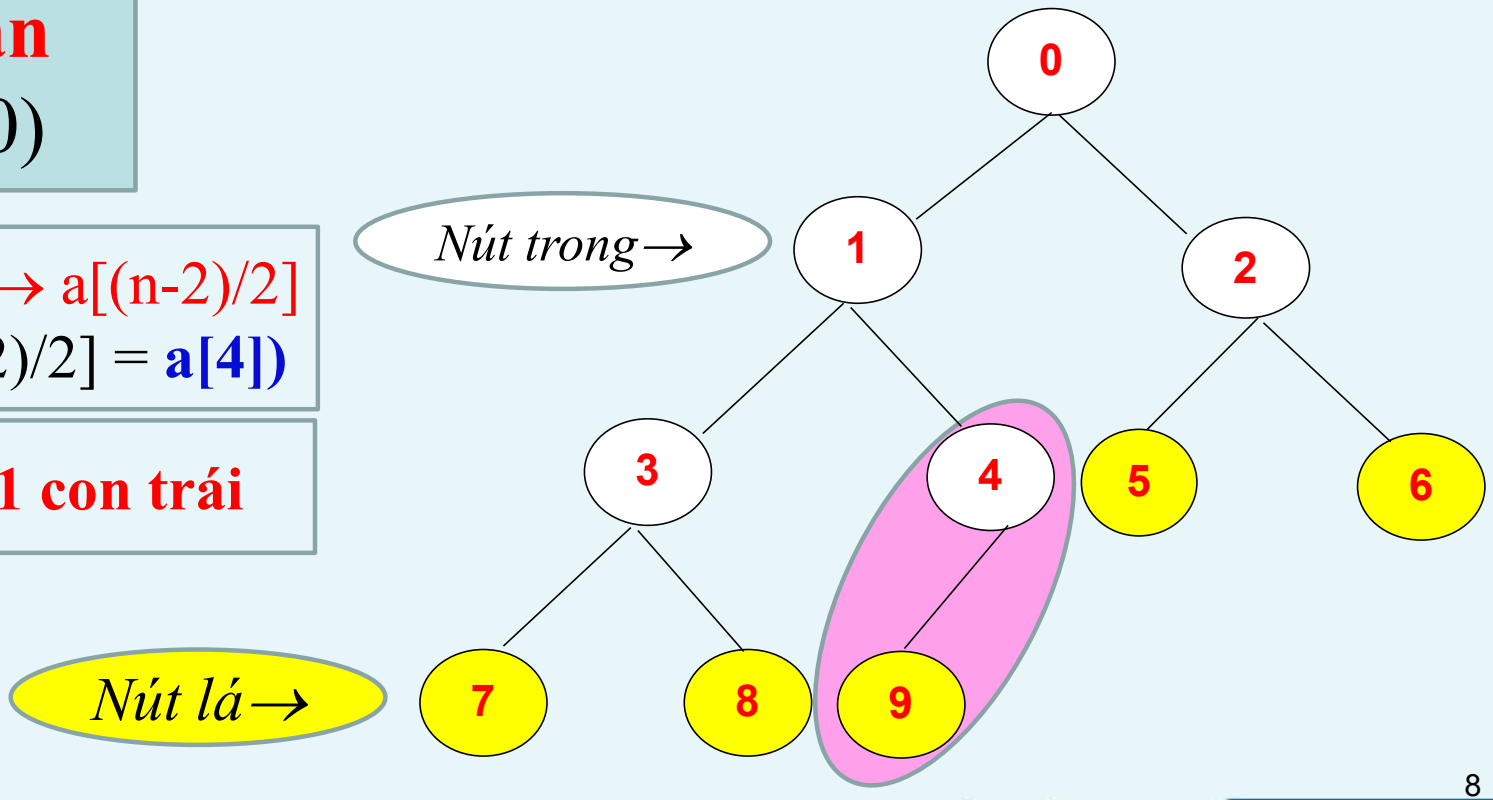
2 nút này cùng các nút lá đã bỏ đi tạo thành mảng theo *thứ tự giảm dần*.

Dựng cây nhị phân từ mảng

n chẵn
(n=10)

Nút trong : a[0] → a[(n-2)/2]
(**a[0]** → a[(10-2)/2] = **a[4]**)

⇒ **a[4]** chỉ có **1 con trái**

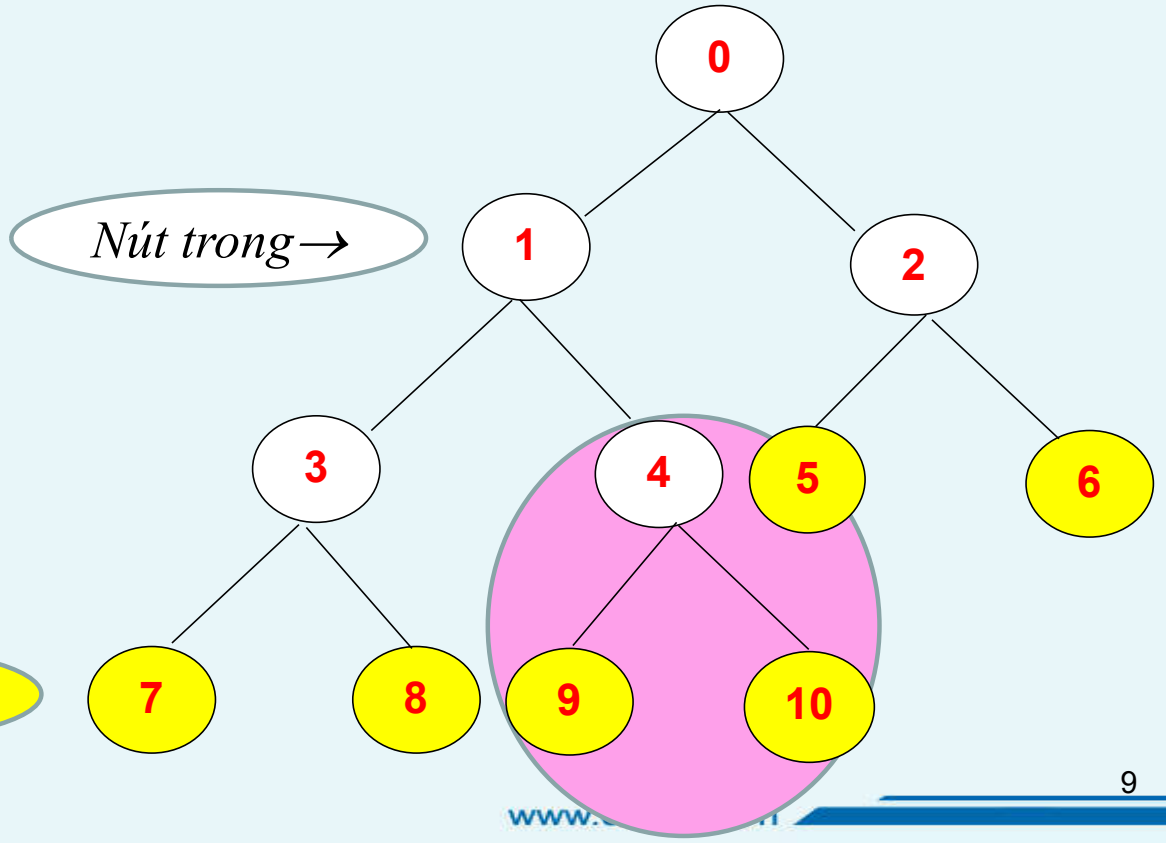


Dựng cây nhị phân từ mảng

n lẻ
(n =11)

Nút trong : a[0] → a[(n-2)/2]
(**a[0]** → a[(11-2)/2] = **a[4]**)

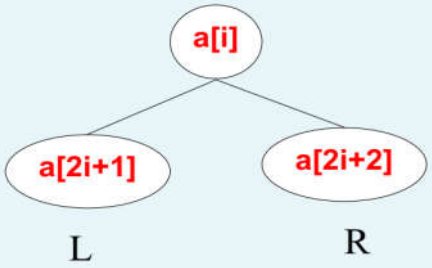
⇒ **a[4]** có đủ **2 con**



Ví dụ: Dựng cây từ mảng

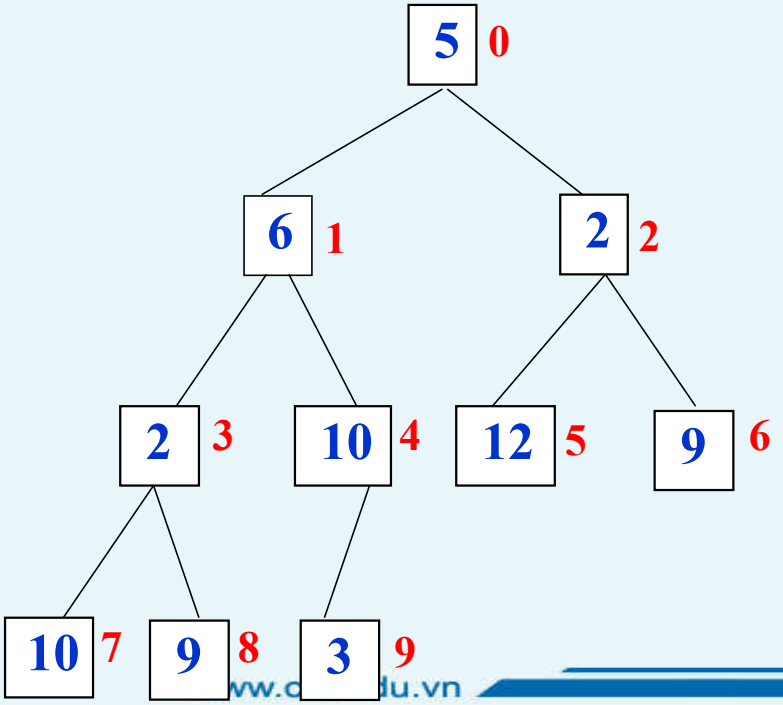
Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Khóa	**5**	**6**	**2**	**2**	**10**	**12**	**9**	**10**	**9**	**3**

- a[0] là **nút gốc**
- a[i] có ***con trái*** a[2i+1] và ***con phải*** a[2i+2].



- Nút trong: [0] → a[(n-2)/2]

(n chẵn: nút a[(n-2)/2] có 1 con trái)



PushDown: Tạo Heap từ cây

- **PushDown** nhận 2 tham số ***first*** và ***last*** để đẩy nút first xuống.
- Giả sử cây a[first],..,a[last] đã đúng vị trí của một Heap trừ a[first]: PushDown sẽ đẩy a[first] xuống đúng vị trí của nó trong cây.

- ***Các khả năng có thể của a[first]:***

  (1) Nếu a[first] chỉ có **một con trái và khoá > khoá con trái**: *hoán đổi a[first] cho con trái của nó và kết thúc.*

  (2) Nếu a[first] có **khoá > khóa con trái và khoá con trái ≤ khoá con phải**: *hoán đổi a[first] cho con trái của nó* → con trái có thể không đúng vị trí nên phải xem *xét lại con trái* để có thể đẩy xuống.

  (3) Ngược lại, nếu a[first] có **khoá > khoá con phải và khoá con phải < khoá con trái** : *hoán đổi a[first] cho con phải của nó* → con phải có thể không đúng vị trí nên phải xem *xét lại con phải* để có thể đẩy xuống.

  Nếu các trường hợp trên không xảy ra : a[first] đã đúng vị trí.

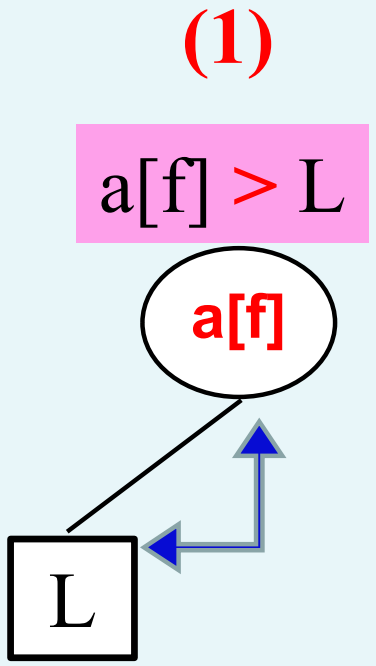
PushDown: Trường hợp 1

(1) Nếu a[first] chỉ có

1 con trái *(là nút lá)* và

khoá > khoá con trái:

Hoán đổi *a[first] cho con trái của nó và kết thúc.*



PushDown: Trường hợp 2

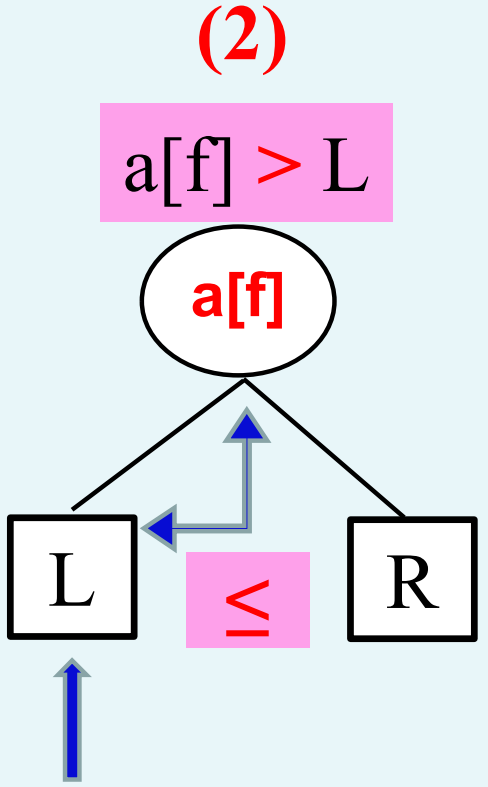
(2) Nếu a[first] có

khoá > khóa con trái và

khoá con trái ≤ khoá con phải:

- ***Hoán đổi*** *a[first] cho con trái của nó*

→ con trái có thể không đúng vị trí nên phải ***xem xét lại con trái*** để đẩy xuống (PushDown L)



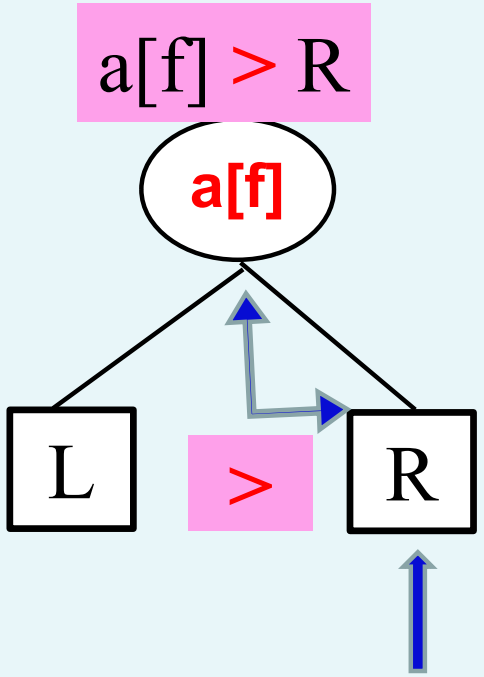
PushDown: Trường hợp 3

(3) Nếu a[first] có
khoá > khoá con phải và
khoá con phải < khoá con trái:

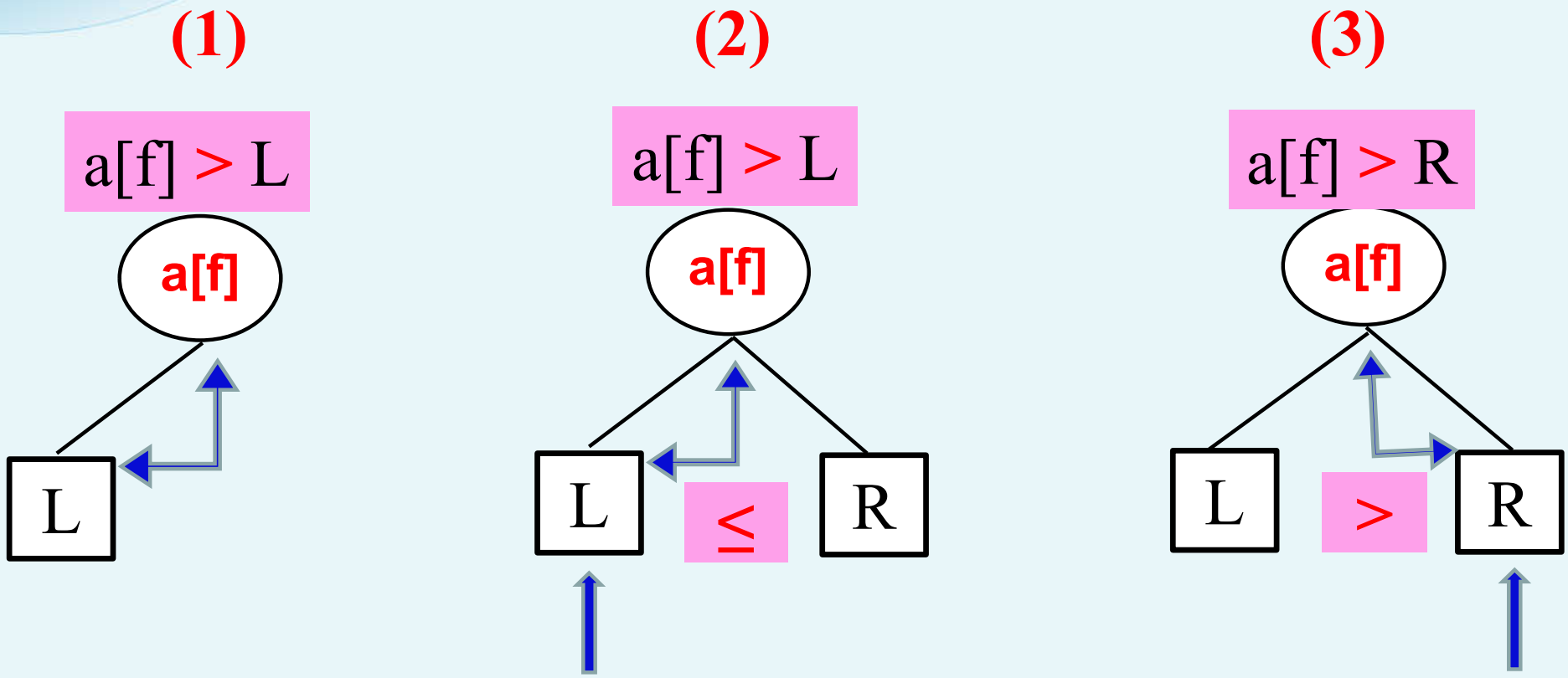
- ***Hoán đổi*** *a[first] cho con phải của nó*

→ con phải có thể không đúng vị trí nên phải ***xem xét lại con phải*** để đẩy xuống (PushDown R)

(3)

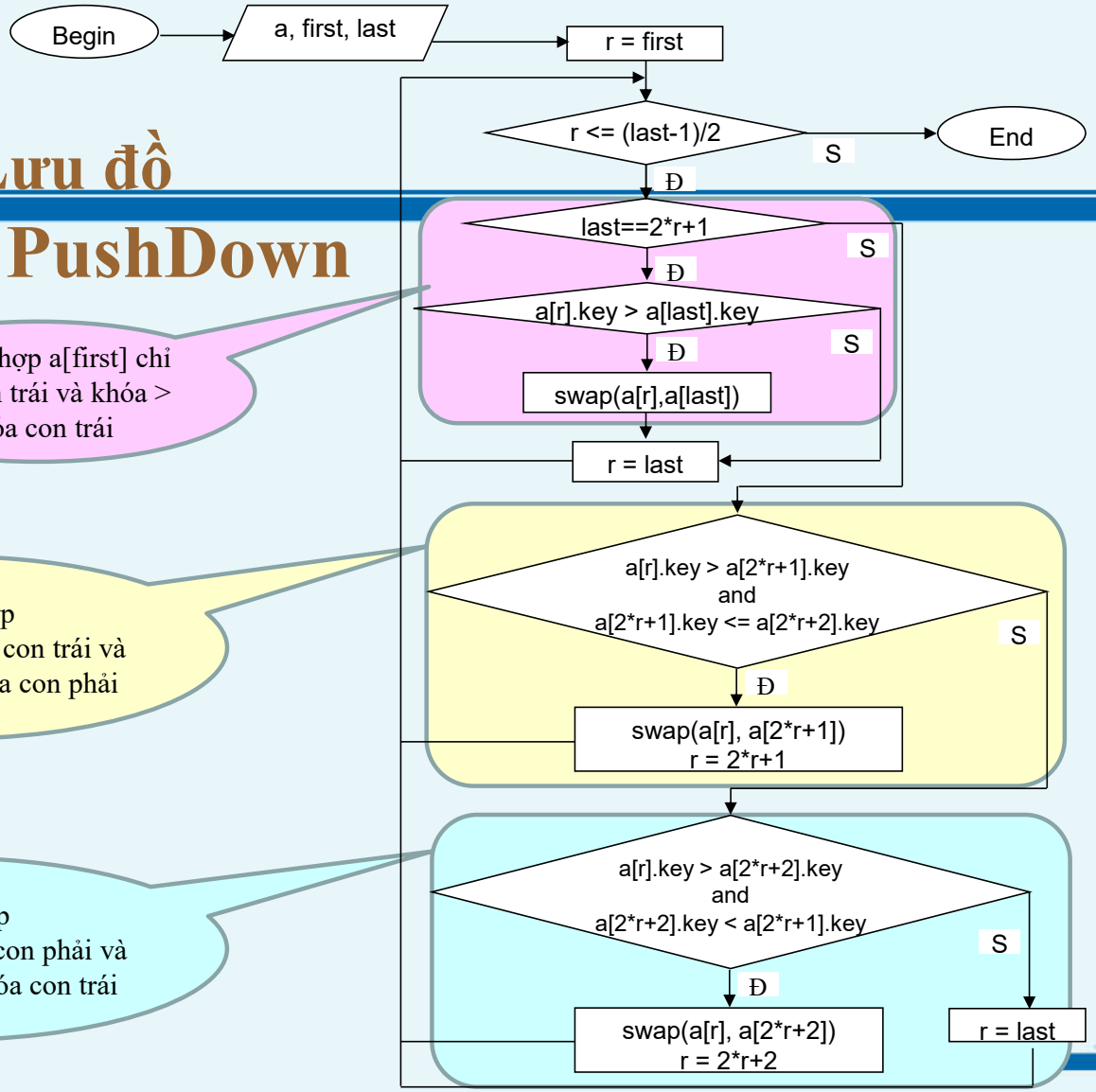


3 trường hợp PushDown: Tạo Heap từ cây



CANTHO UNIVERSITY

Lưu đồ thủ tục PushDown



Trường hợp a[first] chỉ có 1 con trái và khóa > khóa con trái

Trường hợp khóa a[first] > khóa con trái và khóa con trái ≤ khóa con phải

Trường hợp khóa a[first] > khóa con phải và khóa con phải < khóa con trái

Cài đặt thủ tục Pushdown

```
void PushDown(recordtype a[ ], int first,int last)
{ int  r= first;
  while (r <= (last-1)/2)
   if (last == 2*r+1) {
     if (a[r].key > a[last].key) Swap(a[r],a[last]);
     r = last;
   } else
   if ((a[r].key>a[2*r+1].key) && (a[2*r+1].key<=a[2*r+2].key))
   {
      Swap(a[r],a[2*r+1]);
      r = 2*r+1 ;
   } else
      if ((a[r].key>a[2*r+2].key) && (a[2*r+2].key<a[2*r+1].key))
      {
       Swap(a[r],a[2*r+2]);
       r = 2*r+2 ;
      }
      else
       r = last;
}
```

→ **T(n) = O(logn)**

Phân tích thủ tục PushDown

- Xét **PushDown(0, n-1)** : PushDown trên cây n nút.
- PushDown chỉ duyệt trên một nhánh nào đó của cây nhị phân, tức là sau mỗi lần lặp thì số nút còn lại một nửa.

  - Ban đầu PushDown trên cây có **n** nút;
  - Sau lần lặp thứ nhất: PushDown trên cây có **n/2** nút;
  - Sau lần lặp thứ hai: PushDown trên cây có **n/4** nút;…

  Tổng quát, sau lần lặp thứ i: PushDown trên cây có $\mathbf{n/2^i}$ nút.
- Trường hợp xấu nhất (luôn phải thực hiện việc đẩy xuống): while thực hiện i lần sao cho $n/2^i = 1$, tức i=logn. Mỗi lần lặp chỉ thực hiện lệnh IF với thân là lời gọi Swap và lệnh gán, do đó tốn O(1) = 1.
- PushDown lấy **O(logn)** thời gian để đẩy xuống 1 nút trong cây n nút.

Thủ tục HeapSort

(1) Sắp xếp cây ban đầu thành Heap dùng thủ tục **PushDown**

Khởi đầu từ nút **a[(n-2)/2],** lần ngược tới nút gốc **a[0]**

(2) Hoán đổi nút gốc a[0] cho a[i].

(3) Sắp xếp cây **a[0] .. a[i – 1]** thành Heap dùng thủ tục **PushDown**

Lặp lại (2) và (3) cho *i chạy* ***từ n – 1 đến 2***

(4) Cuối cùng chỉ cần hoán đổi a[0] với a[1]

Chương trình hàm HeapSort

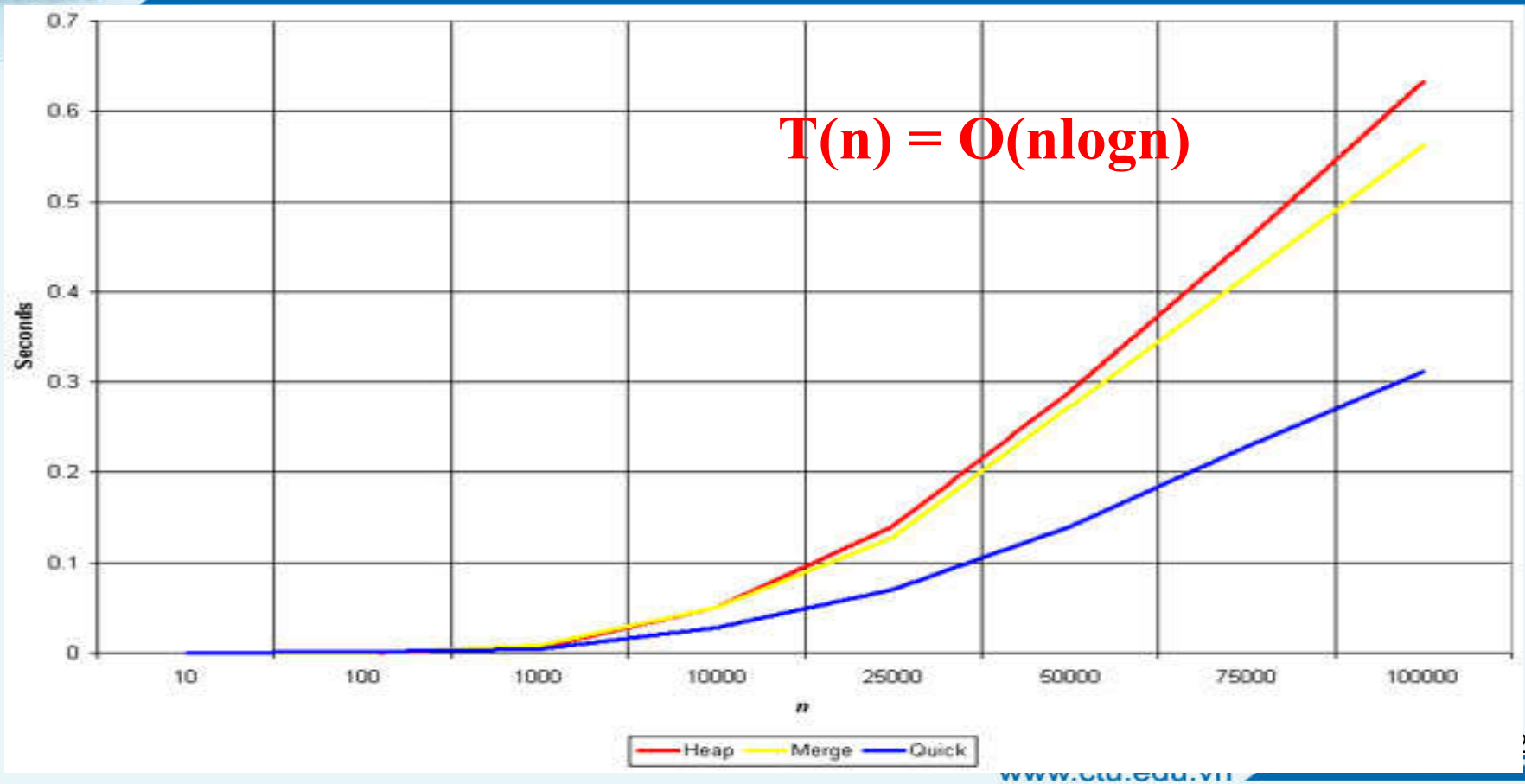
```
void HeapSort(recordtype a[], int n)
    {  int i;
{1}   for (i = (n-2)/2; i>=0; i--)
{2}         PushDown(a, i, n-1);
{3}   for (i = n-1; i>=2; i--) {
{4}         Swap(a[0],a[i]);
{5}         PushDown(a, 0, i-1);
       }
{6}   Swap(a[0],a[1]);
    }
```

→ **T(n) = O(nlogn)**

Phân tích HeapSort

- Hàm **PushDown** lấy O(logn).
- Trong **HeapSort**,
  - Vòng lặp {1}-{2} lặp (n-2)/2+1 lần, mỗi lần tốn O(logn) nên thời gian thực hiện {1}-{2} là O(n logn).
  - Vòng lặp {3}-{5} lặp n-2 lần, mỗi lần tốn O(logn) nên thời gian thực hiện của {3}-{5} là O(n logn).
- Thời gian thực hiện **HeapSort** là **O(nlogn).**

Biểu đồ so sánh hiệu quả



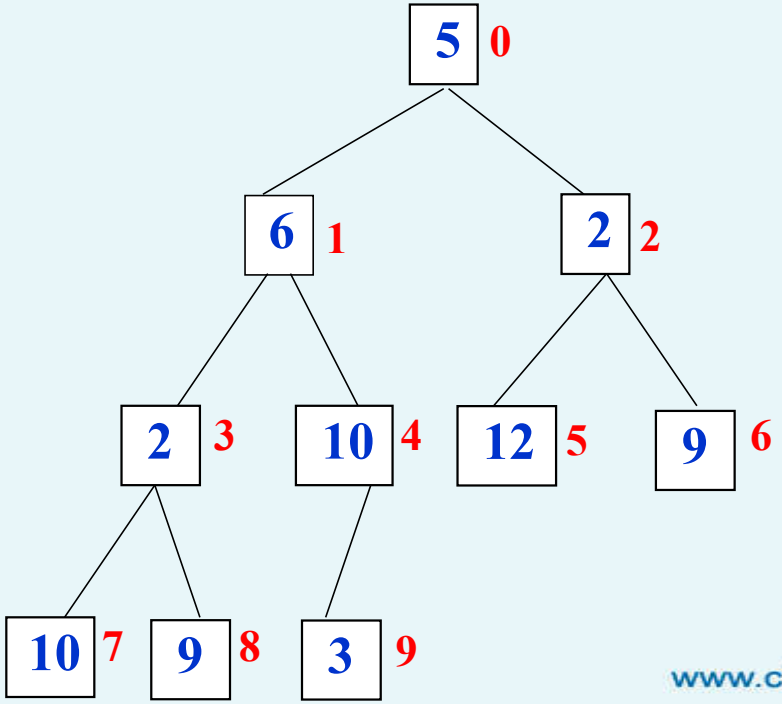
Ví dụ HeapSort: Mô hình cây

Trình bày việc sắp xếp mảng gồm 10 phần tử có khóa là các số nguyên sau bằng cách sử dụng **thuật toán HeapSort** (Sắp thứ tự giảm, sử dụng mô hình cây):

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Khóa	**5**	**6**	**2**	**2**	**10**	**12**	**9**	**10**	**9**	**3**

Ví dụ: Dựng cây từ mảng

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Khóa	5	6	2	2	10	12	9	10	9	3



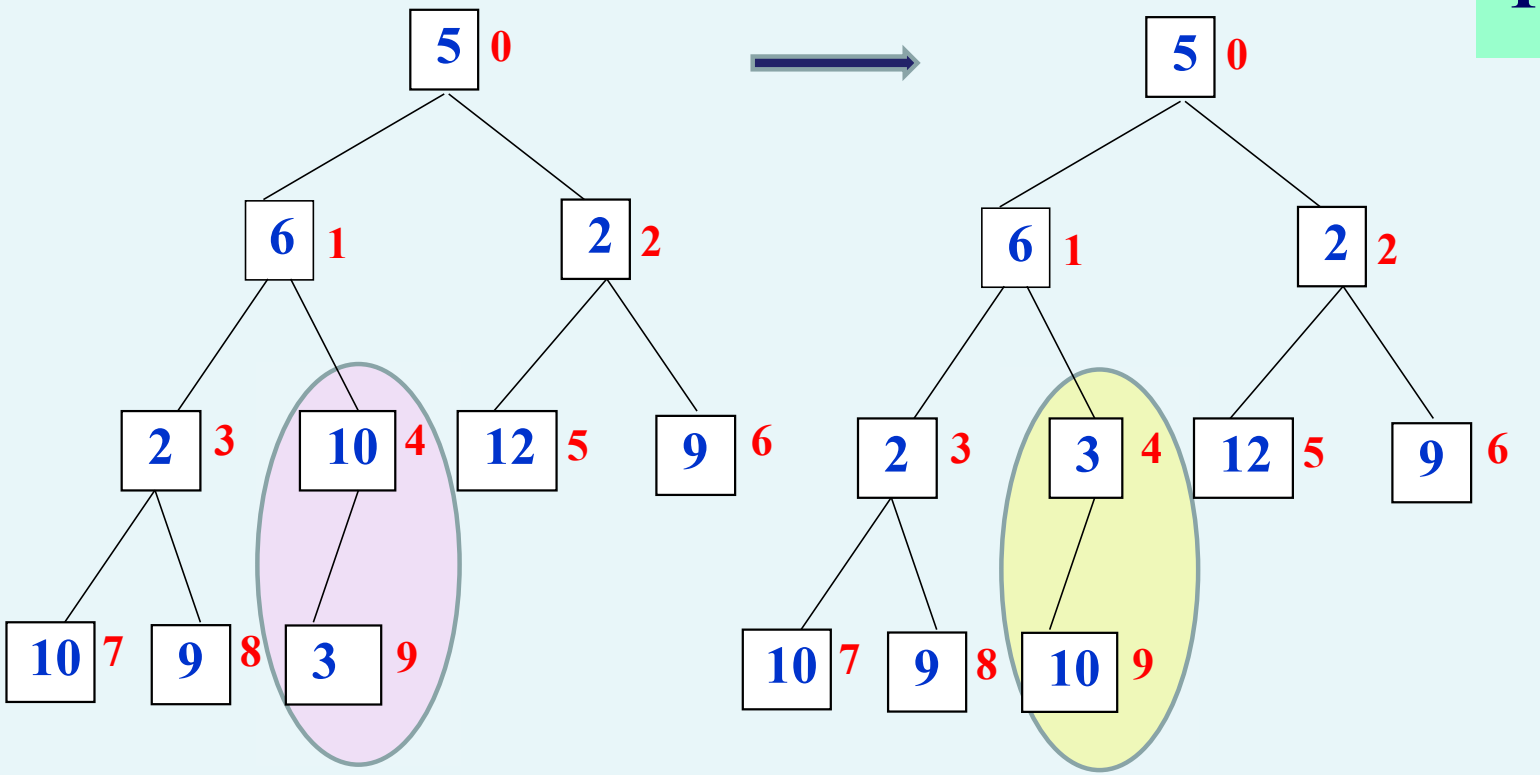
Chương trình hàm HeapSort

```
void HeapSort(recordtype a[], int n)
    {  int i;
{1}   for (i = (n-2)/2; i>=0; i--)
{2}         PushDown(a, i, n-1);
{3}   for (i = n-1; i>=2; i--) {
{4}         Swap(a[0],a[i]);
{5}         PushDown(a, 0, i-1);
       }
{6}  Swap(a[0],a[1]);
    }
```

→ **T(n) = O(nlogn)**

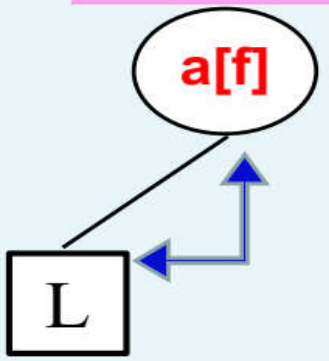
Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]

Pushdown a[4]



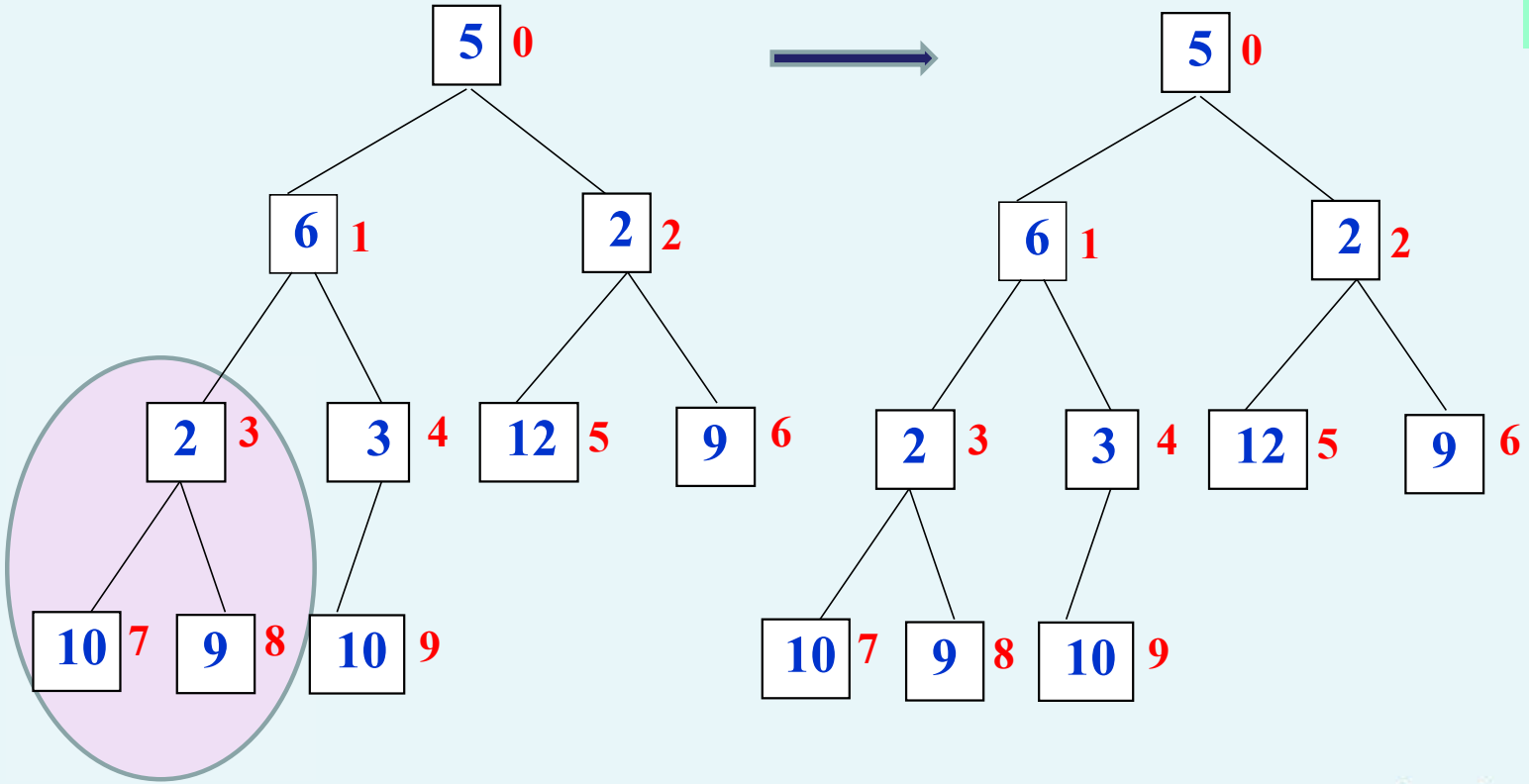
(1)

a[f] > L



Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]

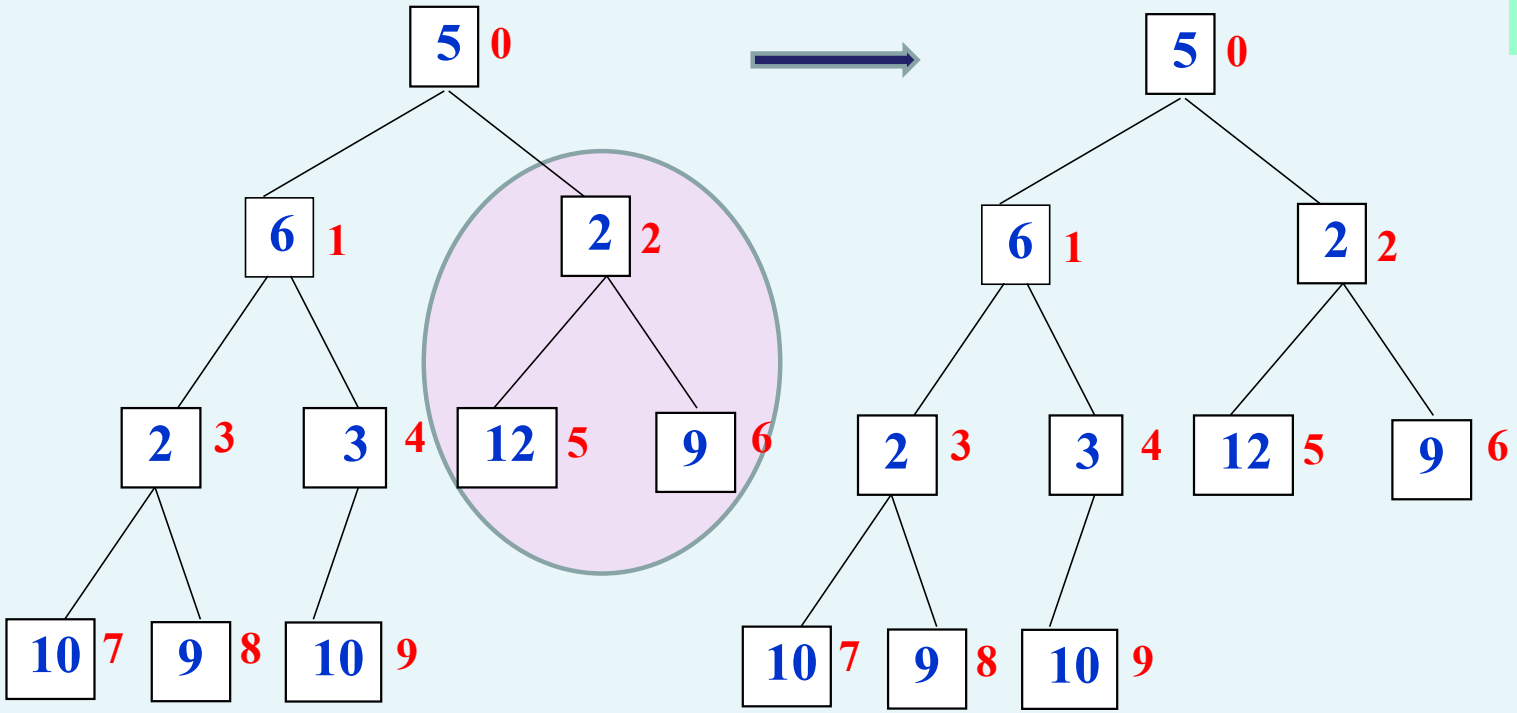
Pushdown a[3]



CANTHO UNIVERSITY

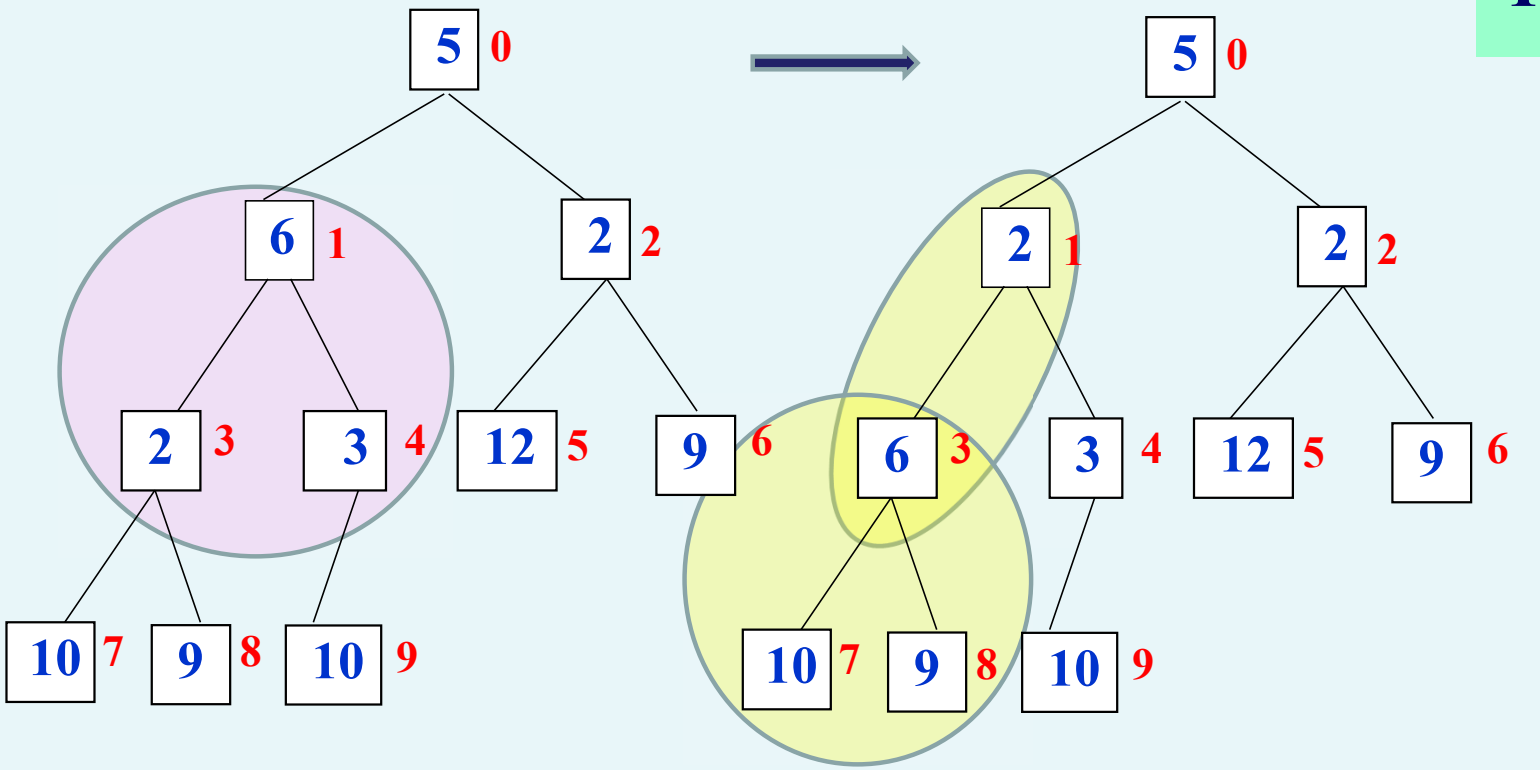
Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]

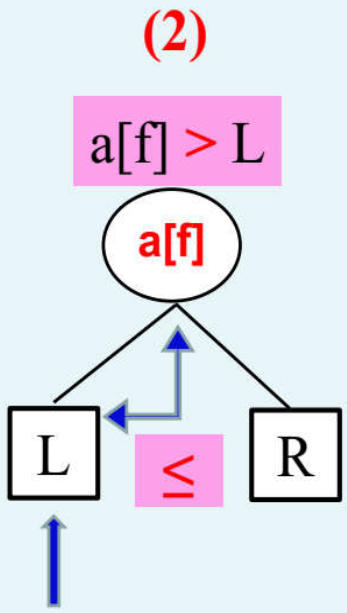
Pushdown a[2]



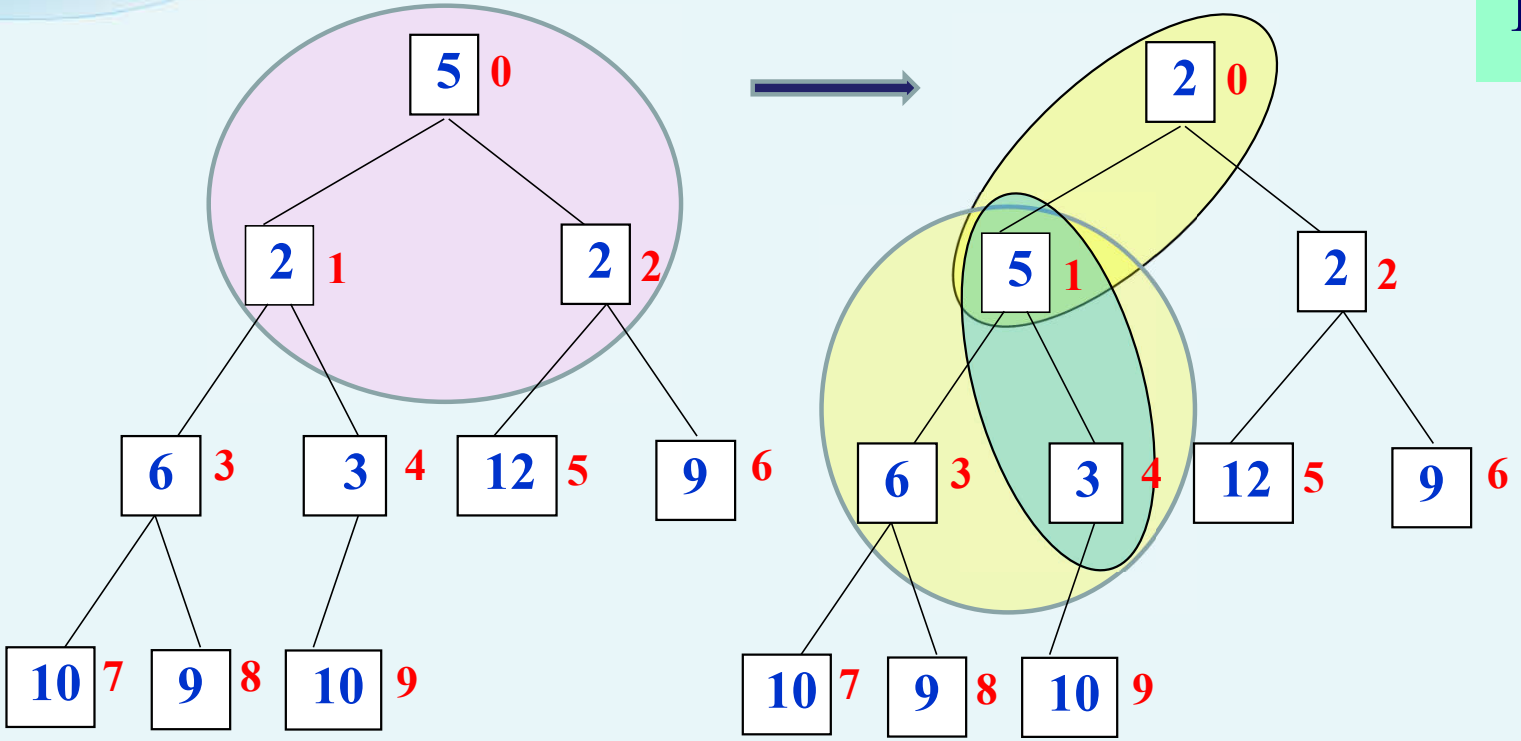
Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]

Pushdown a[1]





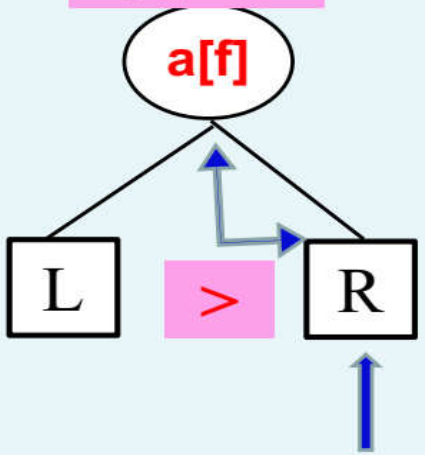
Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]



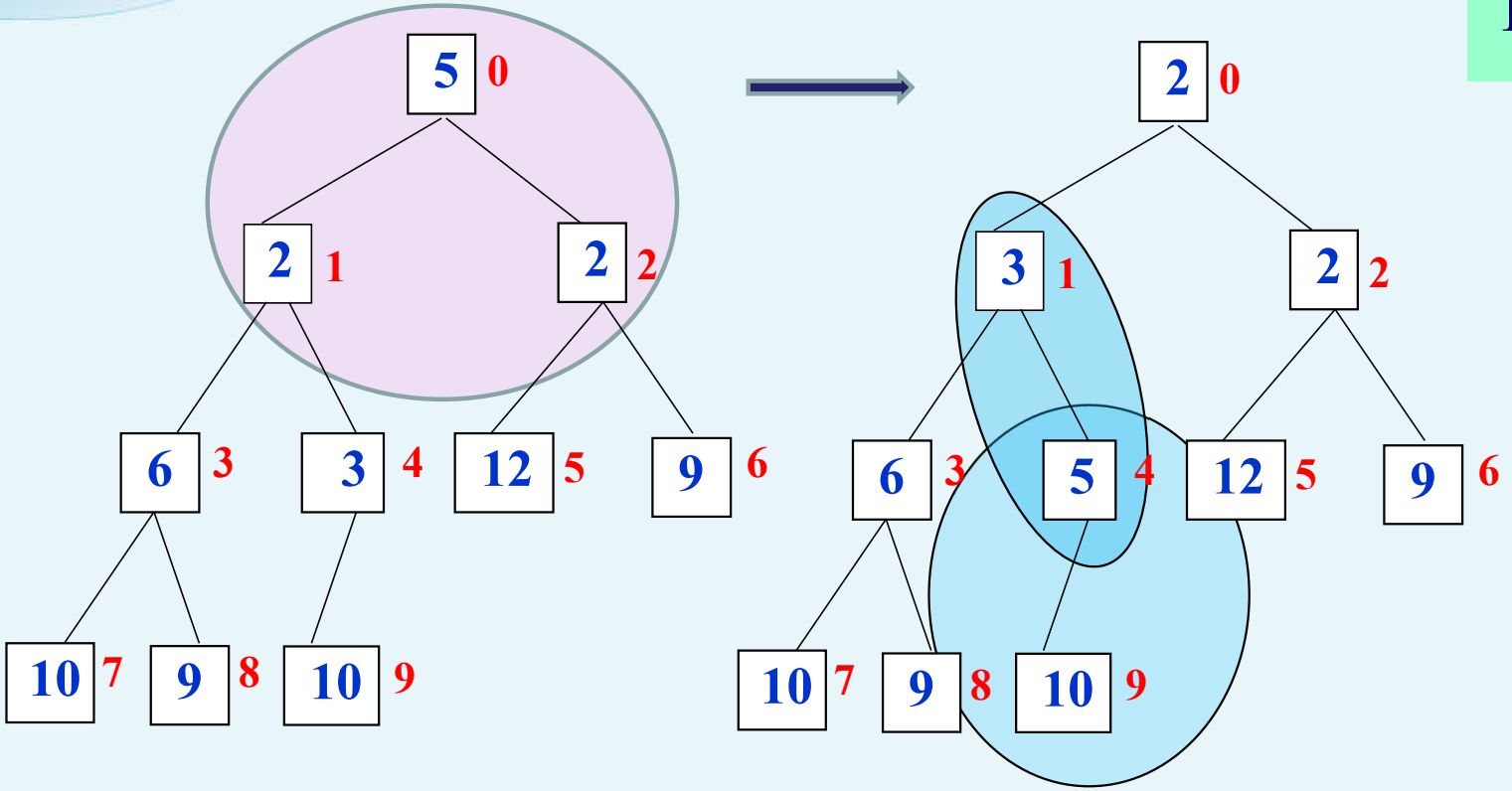
Pushdown a[0]

(3)

a[f] > R



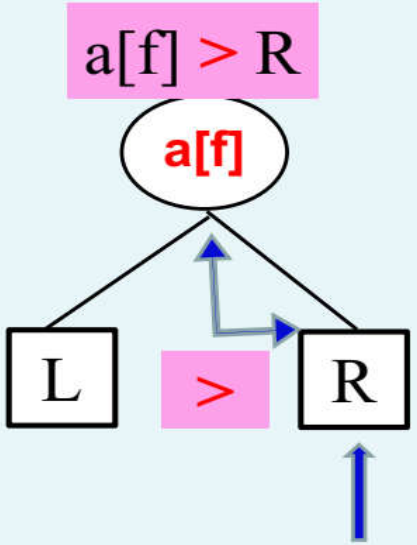
Ví dụ: (1) Tạo Heap từ cây ban đầu a[4]→ a[0]



Pushdown a[0]

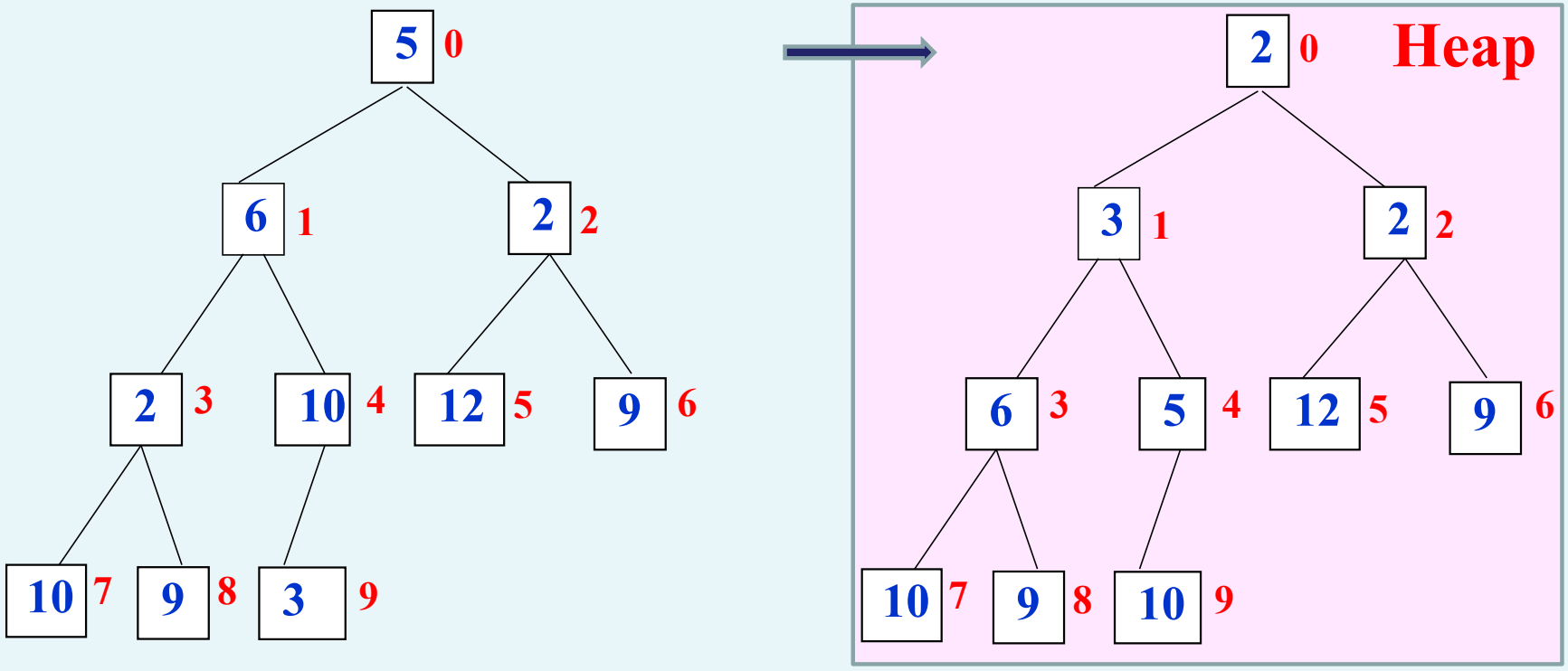
(3)

a[f] > R



Ví dụ: (1) Tạo Heap từ cây ban đầu

→ *Cây ban đầu* đã được tạo thành **Heap**



Chương trình hàm HeapSort

```
void HeapSort(recordtype a[], int n)
     {  int i;
{1}    for (i = (n-2)/2; i>=0; i--)
{2}          PushDown(a, i, n-1);
{3}    for (i = n-1; i>=2; i--) {
{4}          Swap(a[0],a[i]);
{5}          PushDown(a, 0, i-1);
       }
{6}  Swap(a[0],a[1]);
     }
```

→ **T(n) = O(nlogn)**

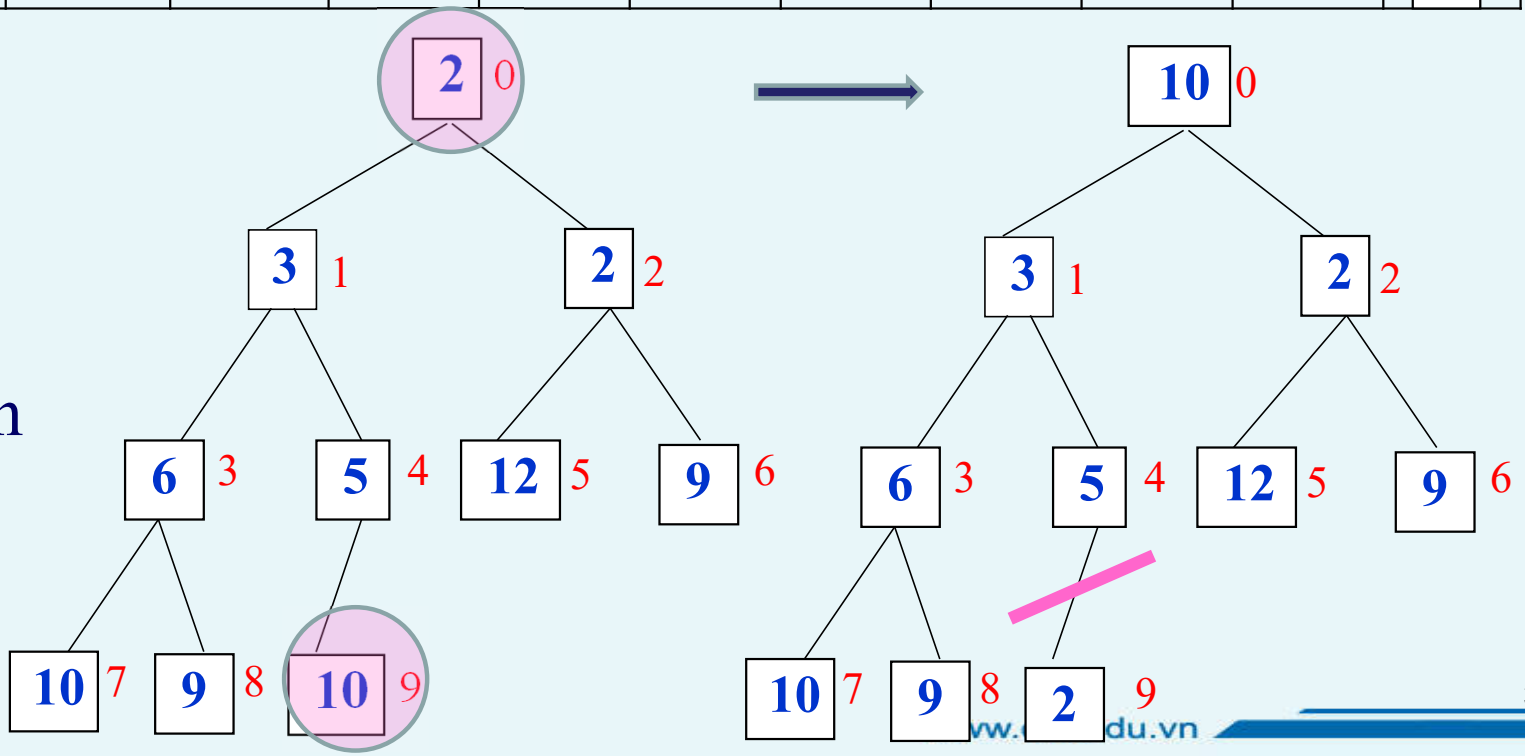
Ví dụ: (2)-(3) Đổi nút và PushDown cây con

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Kết quả										2

→ Đổi a[0] cho a[9]

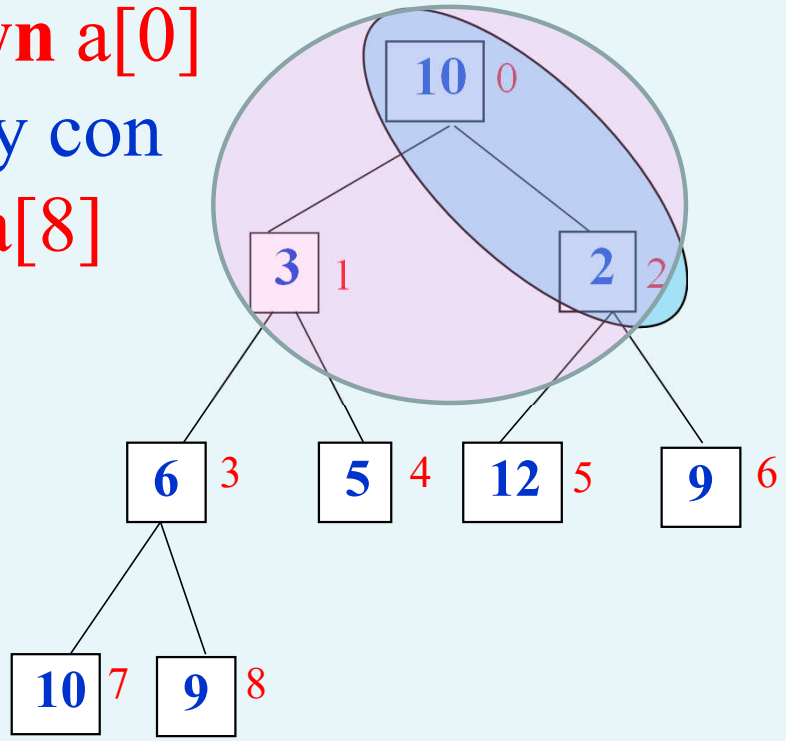
→ Cắt a[9]

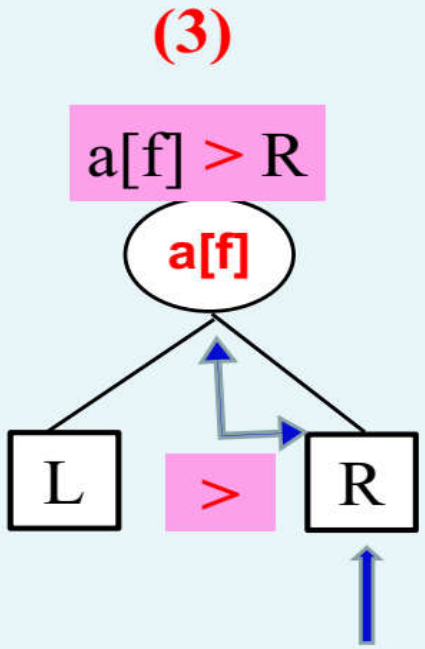
→ PushDown a[0] trong a[0].. a[8]



Ví dụ: (2)-(3) Đổi nút và PushDown cây con

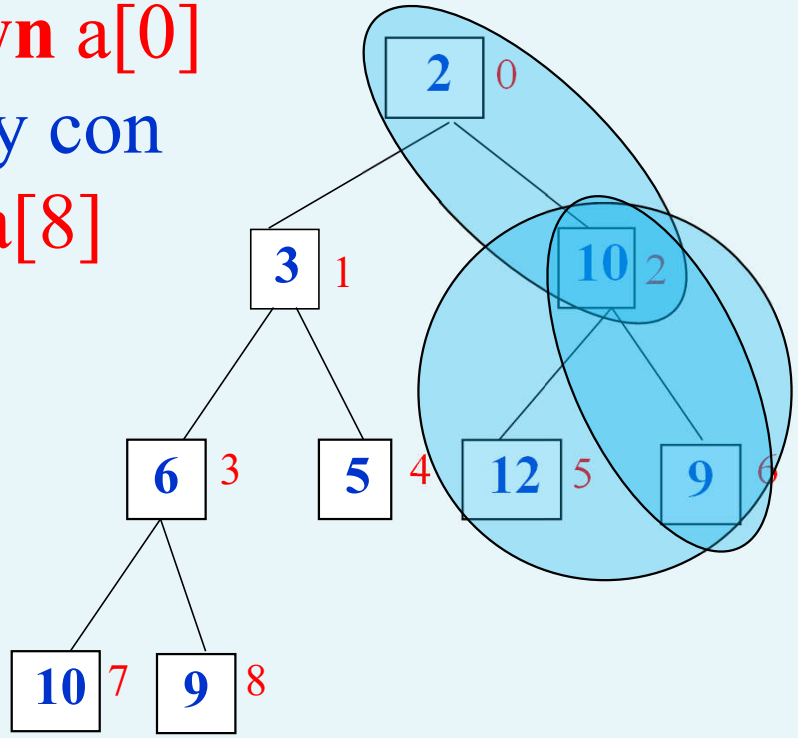
PushDown a[0] trong cây con a[0].. a[8]

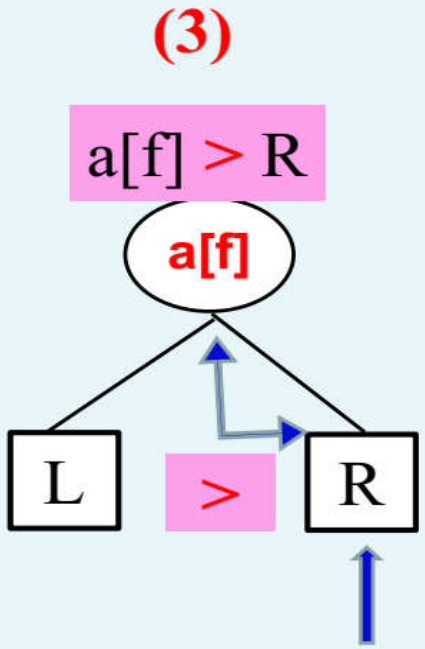




Ví dụ: (2)-(3) Đổi nút và PushDown cây con

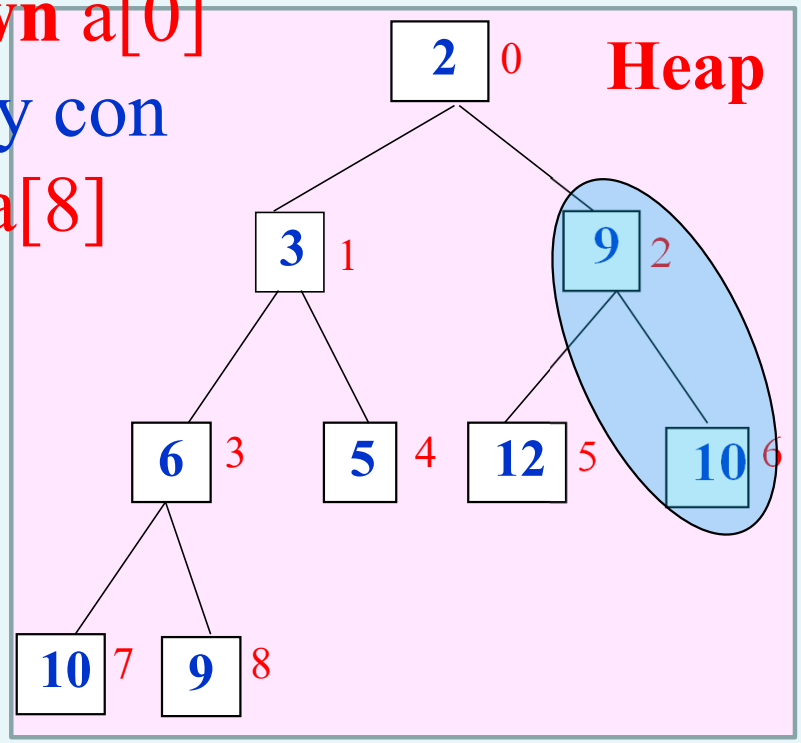
PushDown a[0] trong cây con a[0].. a[8]

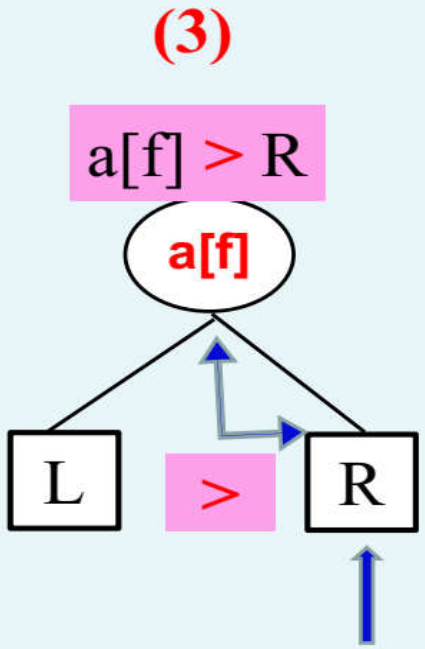




Ví dụ: (2)-(3) Đổi nút và PushDown cây con

PushDown a[0] trong cây con a[0].. a[8]





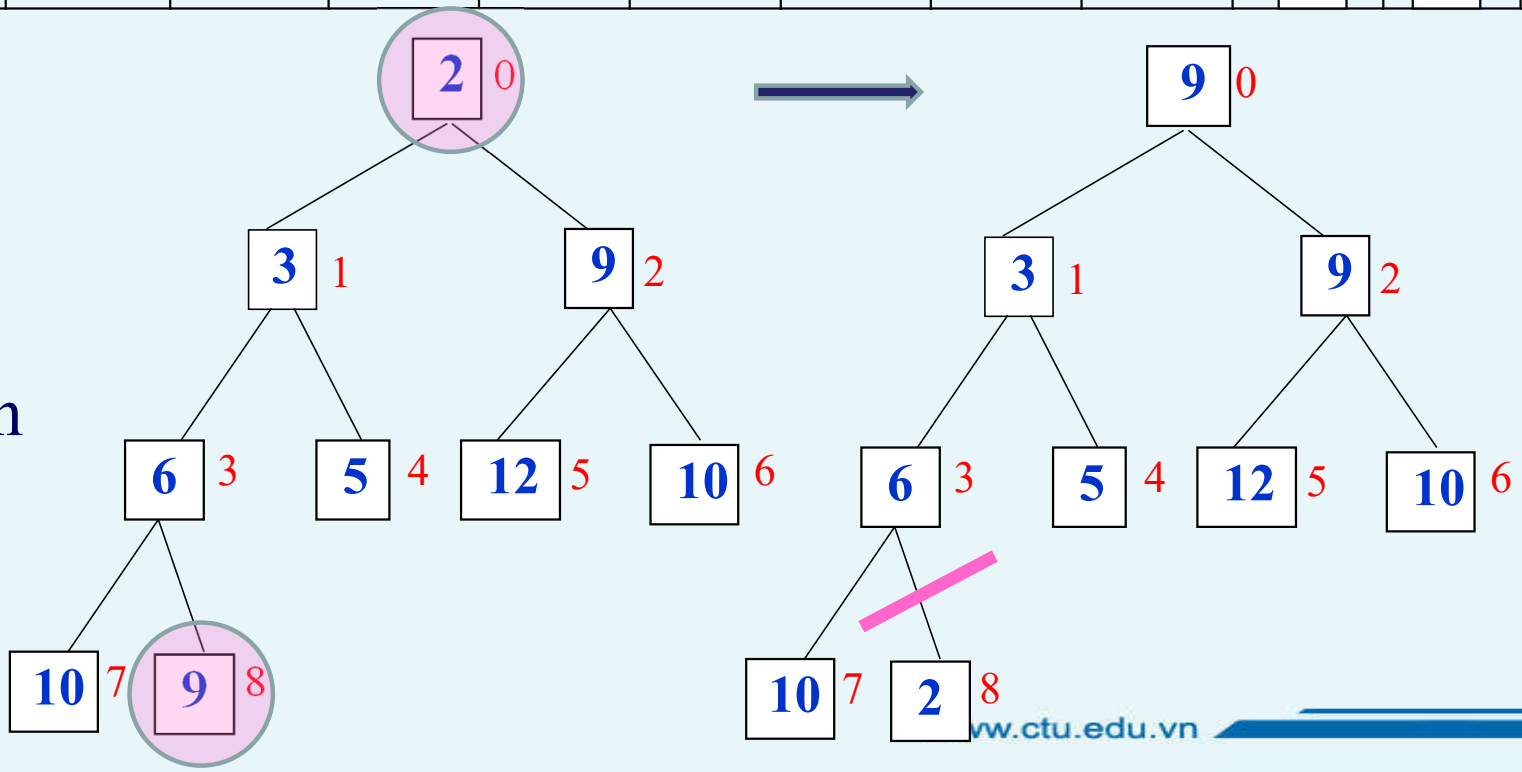
Ví dụ: (2)-(3) Đổi nút và PushDown cây con

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Kết quả									2	2

→ Đổi a[0] cho a[8]

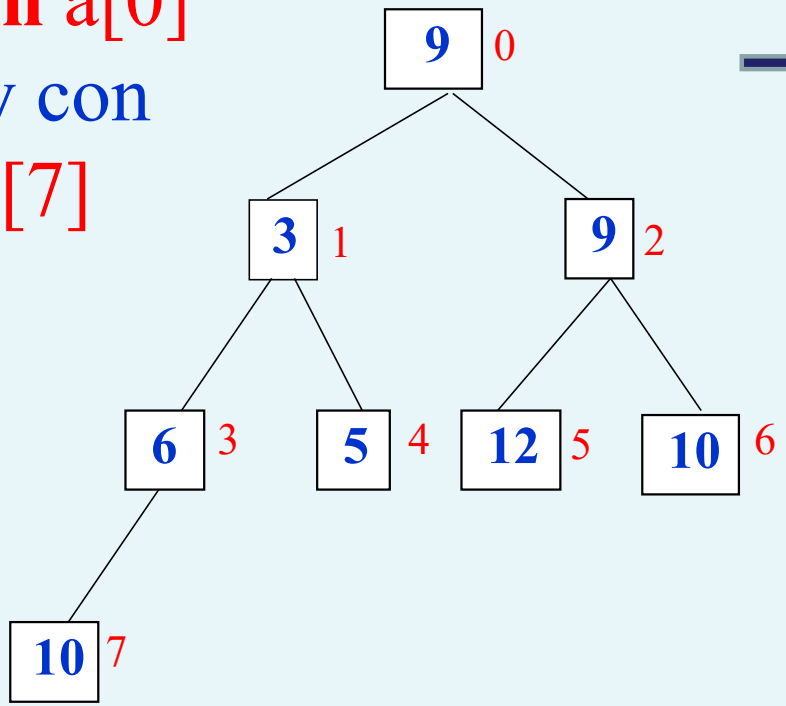
→ Cắt a[8]

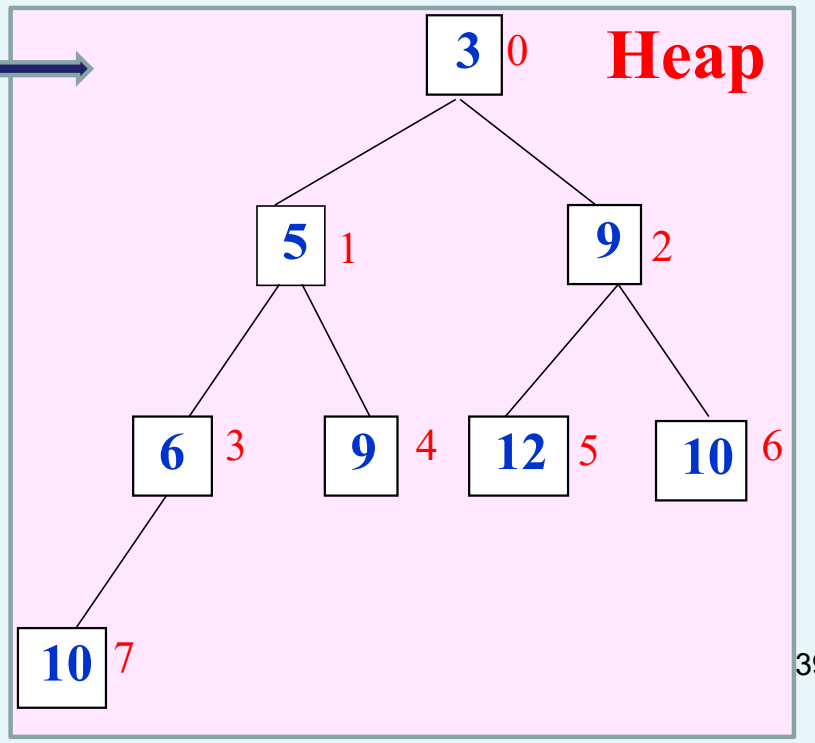
→ PushDown a[0] trong a[0].. a[7]



Ví dụ: (2)-(3) Đổi nút và PushDown cây con

PushDown a[0] trong cây con a[0].. a[7]





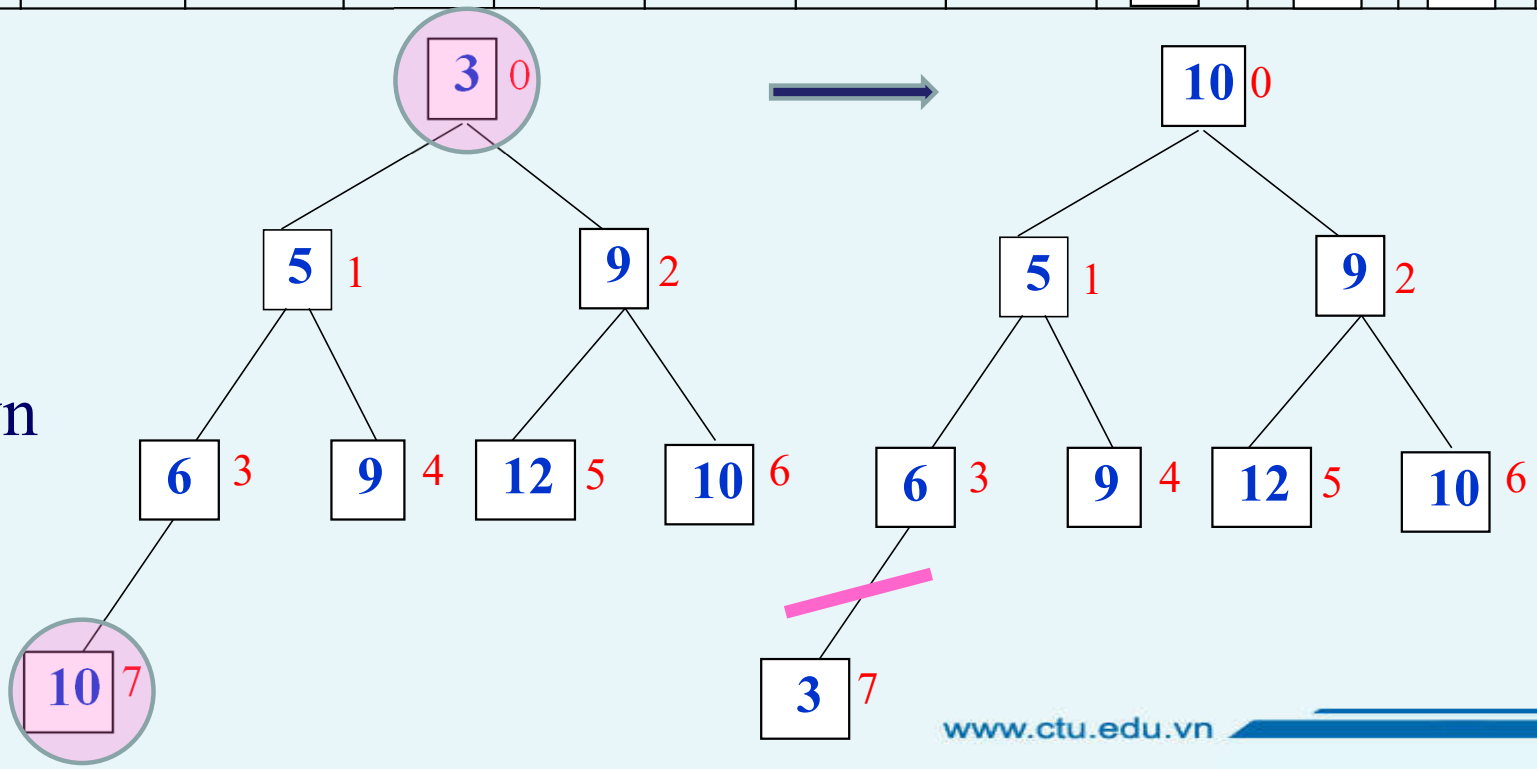
Ví dụ: (2)-(3) Đổi nút và PushDown cây con

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Kết quả								3	2	2

→ Đổi a[0] cho a[7]

→ Cắt a[7]

→ PushDown a[0] trong a[0].. a[6]



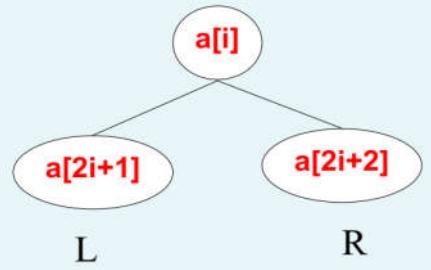
Ví dụ HeapSort: Mô hình cây

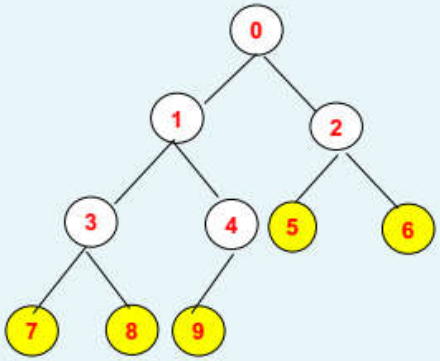
Tiếp tục quá trình trên sẽ được một mảng có thứ tự giảm:

Chỉ số	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5**	**6**	**2**	**2**	**10**	**12**	**9**	**10**	**9**	**3**
Kết quả	**12**	**10**	**10**	**9**	**9**	**6**	**5**	**3**	**2**	**2**

Ví dụ HeapSort: Mô hình mảng

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5**	**6**	**2**	**2**	**10**	**12**	**9**	**10**	**9**	**3**
Tạo Heap										
i = 9										
i = 8										
i = 7										
i = 6										
…										





Chương trình hàm HeapSort

```
void HeapSort(recordtype a[], int n)
    {  int i;
{1}   for (i = (n-2)/2; i>=0; i--)
{2}         PushDown(a, i, n-1);
{3}   for (i = n-1; i>=2; i--) {
{4}         Swap(a[0],a[i]);
{5}         PushDown(a, 0, i-1);
       }
{6}  Swap(a[0],a[1]);
    }
```

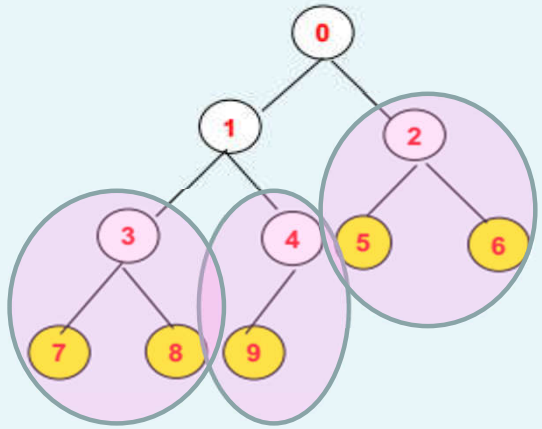
→ **T(n) = O(nlogn)**

CANTHO UNIVERSITY

HeapSort: Tạo Heap ban đầu

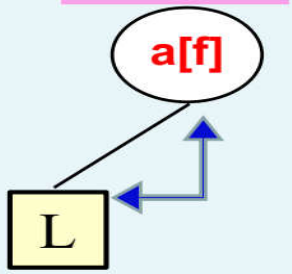
a[(n-2)/2] → a[0]

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
					3					10
Tạo Heap										



(1)

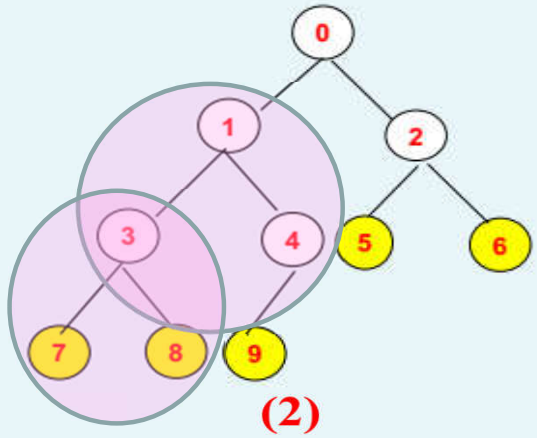
a[f] > L



HeapSort: Tạo Heap ban đầu

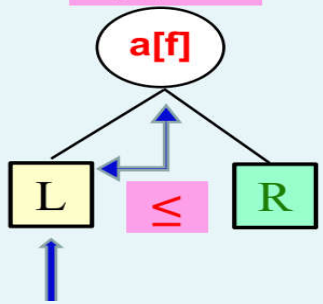
a[(n-2)/2] → a[0]

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5**	6	**2**	2	**10**	**12**	**9**	10	9	**3**
		2		6	3					10
Tạo Heap										



(2)

a[f] > L

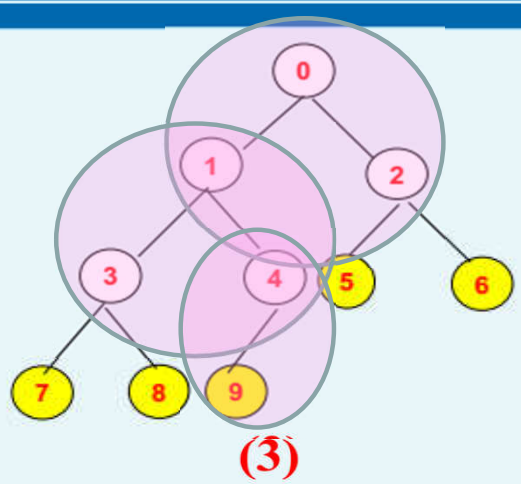


CANTHO UNIVERSITY

HeapSort: Tạo Heap ban đầu

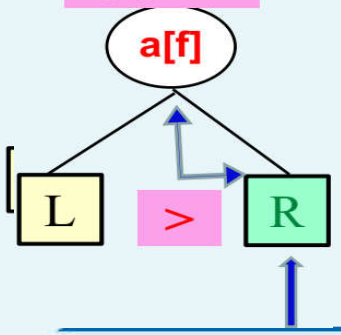
a[(n-2)/2] → a[0]

Bước \ Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Tạo Heap	2	2 5 3		6	3 5					10



(3)

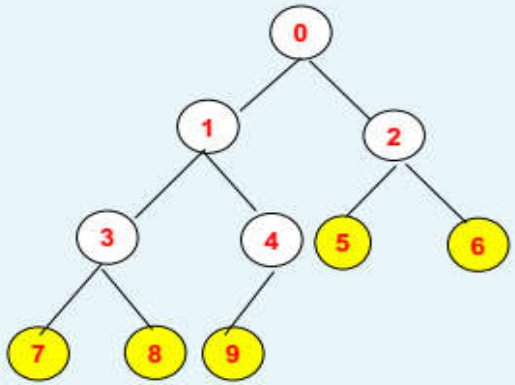
a[f] > R



HeapSort: Tạo Heap ban đầu

a[(n-2)/2] → a[0]

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Heap	**2**	**3**	**2**	**6**	**5**	**12**	**9**	**10**	**9**	**10**



Chương trình hàm HeapSort

```
void HeapSort(recordtype a[], int n)
     {  int i;
{1}    for (i = (n-2)/2; i>=0; i--)
{2}          PushDown(a, i, n-1);
{3}    for (i = n-1; i>=2; i--) {
{4}          Swap(a[0],a[i]);
{5}          PushDown(a, 0, i-1);
       }
{6}  Swap(a[0],a[1]);
     }
```

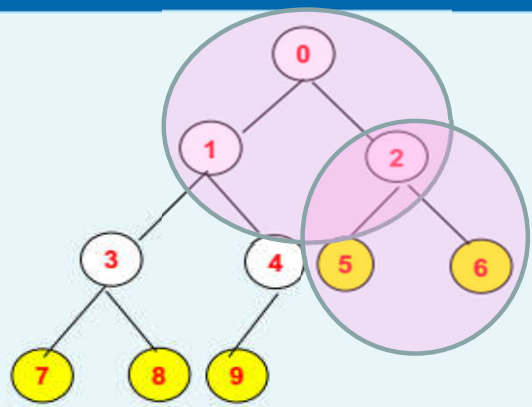
→ **T(n) = O(nlogn)**

CANTHO UNIVERSITY

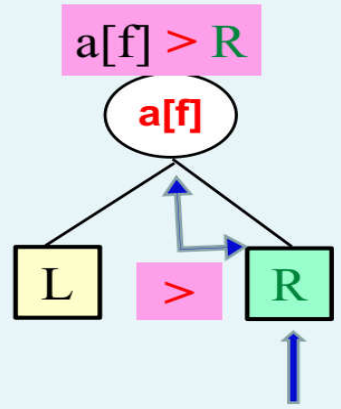
Ví dụ HeapSort: i = 9

i : (n-1) → 2

Bước \ Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Tạo Heap	**2** 10 2	**3**	**2** 10 9	**6**	**5**	**12**	**9** 10	**10**	**9**	**10** **2**
i = 9	**2**	**3**	**9**	**6**	**5**	**12**	**10**	**10**	**9**	
i = 8										
i = 7										
i = 6										
…										



(3)

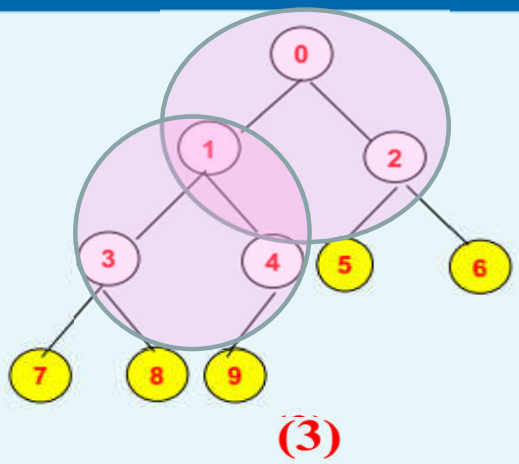


CANTHO UNIVERSITY

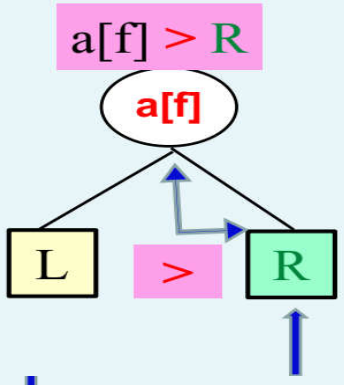
Ví dụ HeapSort: i = 8

i : (n-1) → 2

Bước \ Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Tạo Heap	**2** 10 2	**3**	**2** 10 9	**6**	**5**	**12**	**9** 10	**10**	**9**	**10** **2**
i = 9	**2** 9 3	**3** 9 5	**9**	**6**	**5** 9	**12**	**10**	**10**	**9** **2**	
i = 8	**3**	**5**	**9**	**6**	**9**	**12**	**10**	**10**		
i = 7										
i = 6										
…										



(3)

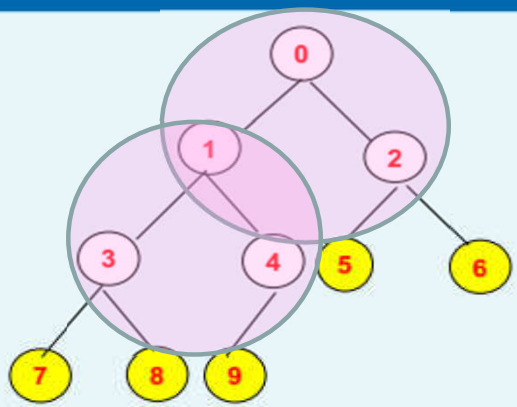


CANTHO UNIVERSITY

Ví dụ HeapSort: i = 7

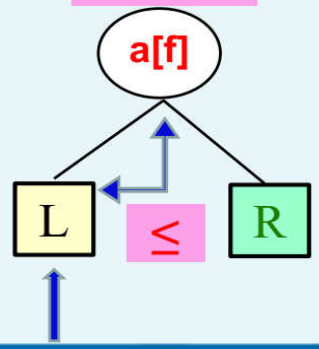
i : (n-1) → 2

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Tạo Heap	**2** 10 2	**3**	**2** 10 9	**6**	**5**	**12**	**9** 10	**10**	**9**	**10** **2**
i = 9	**2** 9 3	**3** 9 5	**9**	**6**	**5** 9	**12**	**10**	**10**	**9** **2**	
i = 8	**3** 10 5	**5** 10 6	**9**	**6** 10	**9**	**12**	**10**	**10** **3**		
i = 7	**5**	**6**	**9**	**10**	**9**	**12**	**10**			
i = 6										
…										



(2)

a[f] > L



CANTHO UNIVERSITY

HeapSort: Trình bày bằng mảng

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Tạo Heap	**2** 10 2	**3**	**2** 10 9	**6**	**5**	**12**	**9** 10	**10**	**9**	**10** 2
i = 9	**2** 9 3	**3** 9 5	**9**	**6**	**5** 9	**12**	**10**	**10**	**9** 2	**2**
i = 8	**3** 10 5	**5** 10 6	**9**	**6** 10	**9**	**12**	**10**	**10** 3	**2**	
i = 7	**5** 10 6	**6** 10 9	**9**	**10**	**9** 10	**12**	**10** 5	3		
i = 6	**6** 12 9	**9** 12 10	**9**	**10** 12	**10**	**12** 6	**5**			
…										

CANTHO UNIVERSITY

HeapSort: Trình bày bằng mảng

Bước \ Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
i = 6	**6** 12 9	**9** 12 10	**9**	**10** 12	**10**	**12** 6	**5**			
i = 5	**9** 10 9	**10**	**9** 10	**12**	**10** 9	**6**				
i = 4	**9** 12 10	**10** 12	**10**	**12** 9	**9**					
i = 3	**10** 10	**12**	**10** 10	**9**						
i = 2	**10** 12	**12** 10	**10**							
Kết quả	**12**	**10**	**10**	**9**	**9**	**6**	**5**	**3**	**2**	**2**

Bài tập 5

Minh họa việc sắp xếp mảng gồm 12 phần tử có khóa là các số nguyên sau bằng cách sử dụng **thuật toán HeapSort** (Sắp thứ tự giảm, trình bày bằng mảng):

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
Ban đầu	**50**	**10**	**143**	**0**	**10**	**120**	**90**	**10**	**90**	**140**	**20**	**30**

Chỉ số	0	1	2	3	4	5	6	7	8	9	10	11
Ban đầu	**50** 0	**10** 0 50 10	**143** 30	**0** 10 50 10	**10**	**120** 30 143 120	**90**	**10** 50	**90**	**140**	**20**	**30** 120 143
Heap	**0** 143 10	**10** 143 10	**30**	**10** 143 50	**10**	**120**	**90**	**50** 143	**90**	**140**	**20**	**143** 0
i = 11	**10** 20 10	**10** 20 10	**30**	**50**	**10** 20	**120**	**90**	**143**	**90**	**140**	**20** 10	**0**
i = 10	**10** 140 10	**10** 140 20	**30**	**50**	**20** 140	**120**	**90**	**143**	**90**	**140** 10	**10**	
i = 9	**10** 90 20	**20** 90 50	**30**	**50** 90	**140**	**120**	**90**	**143**	**90** 10	**10**		
i = 8	**20** 143 30	**50**	**30** 143 90	**90**	**140**	**120**	**90** 143	**143** 20	**10**			
i = 7	**30** 143 50	**50** 143 90	**90**	**90** 143	**140**	**120**	**143** 30	**20**				
i = 6	**50** 120 90	**90** 120	**90**	**143**	**140**	**120** 50	**30**					
i = 5	**90** 140 90	**120**	**90** 140	**143**	**140** 90	**50**						
i = 4	**90** 143 120	**120** 143	**140**	**143** 90	**90**							
i = 3	**120** 140	**143**	**140** 120	**90**								
i = 2	**140** 143	**143** 140	**120**									
Kết quả	**143**	**140**	**120**	**90**	**90**	**50**	**30**	**20**	**10**	**10**	**10**	**0**