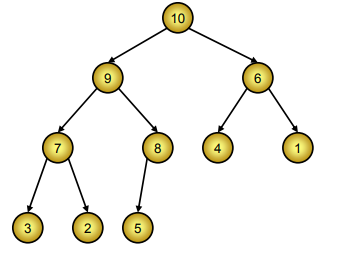
**8.7. THUẬT TOÁN SẮP XẾP KIỂU VUN ĐỐNG (HEAPSORT)**

**8.7.1. Đống (heap)**

Đống là một dạng cây nhị phân hoàn chỉnh đặc biệt mà giá trị lưu tại mọi nút nhánh đều lớn hơn hay bằng giá trị lưu trong hai nút con của nó.



**Heap**

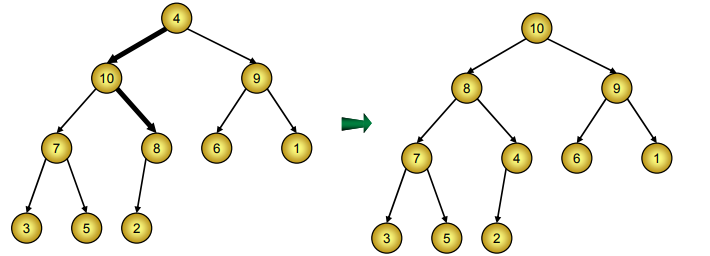
**8.7.2. Vun đống**

Trong bài học về cây, ta đã biết một dãy khoá k1, k2, …, kn là biểu diễn của một cây nhị phân hoàn chỉnh mà ki là giá trị lưu trong nút thứ i, nút con của nút thứ i là nút 2i và nút 2i + 1, nút cha của nút thứ j là nút j div 2. Vấn đề đặt ra là sắp lại dãy khoá đã cho để nó biểu diễn một đống.

Vì cây nhị phân chỉ gồm có một nút hiển nhiên là đống, nên **để vun một nhánh cây gốc r thành đống, ta có thể coi hai nhánh con của nó (nhánh gốc 2r và 2r + 1) đã là đống rồi** và thực hiện thuật toán vun đống từ dưới lên (bottom-up) đối với cây: Gọi h là chiều cao của cây, nút ở mức h (nút lá) đã là gốc một đống, ta vun lên để những nút ở mức h - 1 cũng là gốc của đống, … cứ như vậy cho tới nút ở mức 1 (nút gốc) cũng là gốc của đống.

**Thuật toán vun thành đống đối với cây gốc r, hai nhánh con của r đã là đống rồi:**

Giả sử ở nút r chứa giá trị V. Từ r, ta cứ đi tới nút con chứa giá trị lớn nhất trong 2 nút con, cho tới khi gặp phải một nút c mà mọi nút con của c đều chứa giá trị  V (nút lá cũng là trường hợp riêng của điều kiện này). Dọc trên đường đi từ r tới c, ta đẩy giá trị chứa ở nút con lên nút cha và đặt giá trị V vào nút c.

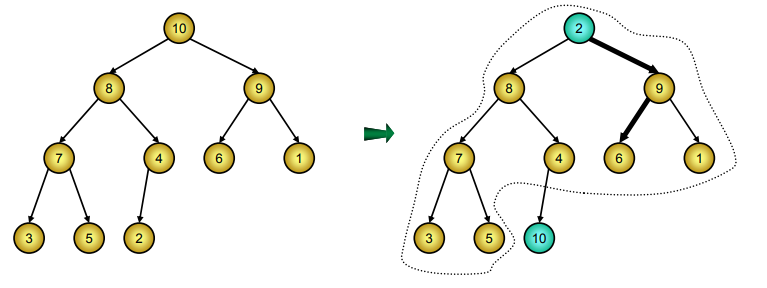


**Vun đống**

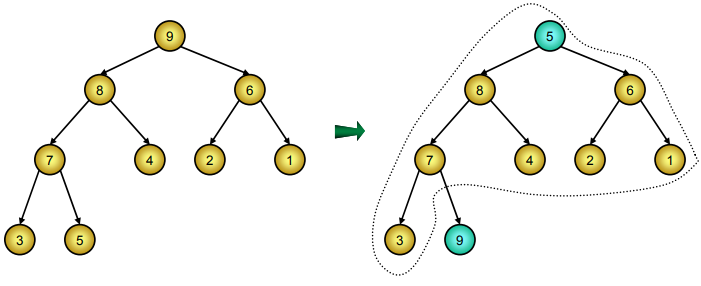
**8.7.3. Tư tưởng của HeapSort**

Đầu tiên, dãy khoá k1, k2, …, kn được vun từ dưới lên để nó biểu diễn một đống, khi đó khoá k1 tương ứng với nút gốc của đống là khoá lớn nhất, ta đảo giá trị khoá đó cho kn và không tính tới kn nữa. Còn lại dãy khoá k1, k2, …, kn-1 tuy không còn là biểu diễn của một đống nữa nhưng nó lại biểu diễn cây nhị phân hoàn chỉnh mà hai nhánh cây ở nút thứ 2 và nút thứ 3 (hai nút con của nút 1) đã là đống rồi. Vậy chỉ cần vun một lần, ta lại được một đống, đảo giá trị k1 cho kn-1 và tiếp tục cho tới khi đống chỉ còn lại 1 nút.

Ví dụ:



**Đảo giá trị k1 cho kn và xét phần còn lại**



**Vun phần còn lại thành đống rồi lại đảo trị k1 cho kn-1**

Thuật toán HeapSort có hai thủ tục chính:

Thủ tục Adjust(root, endnode) vun cây gốc root thành đống trong điều kiện hai cây gốc 2.root và 2.root +1 đã là đống rồi. Các nút từ endnode + 1 tới n đã nằm ở vị trí đúng và không được tính tới nữa.

Thủ tục HeapSort mô tả lại quá trình vun đống và chọn phần tử theo ý tưởng trên:

procedure HeapSort;

var

r, i: Integer;

procedure Adjust(root, endnode: Integer); {Vun cây gốc Root thành đống}

var

c: Integer;

Key: TKey; {Biến lưu giá trị khoá ở nút Root}

begin

Key := kroot;

while root \* 2 **** endnode do {Chừng nào root chưa phải là lá}

begin

c := Root \* 2; {Xét nút con trái của Root, so sánh với giá trị nút con phải, chọn ra nút mang giá trị lớn nhất}

if (c < endnode) and (kc < kc+1) then c := c + 1;

if kc **** Key then Break; {Cả hai nút con của Root đều mang giá trị **** Key thì dừng ngay}

kroot := kc; root := c; {Chuyển giá trị từ nút con c lên nút cha root và đi xuống xét nút con c}

end;

kroot := Key; {Đặt giá trị Key vào nút root}

end;

begin {Bắt đầu thuật toán HeapSort}

for r := n div 2 downto 1 do Adjust(r, n); {Vun cây từ dưới lên tạo thành đống}

for i := n downto 2 do

begin

<Đảo giá trị k1 và ki> {Khoá lớn nhất được chuyển ra cuối dãy}

Adjust(1, i - 1); {Vun phần còn lại thành đống}

end;

end;

Về độ phức tạp của thuật toán, ta đã biết rằng cây nhị phân hoàn chỉnh có n nút thì chiều cao của nó không quá [log2(n + 1)] + 1. Cứ cho là trong trường hợp xấu nhất thủ tục Adjust phải thực hiện tìm đường đi từ nút gốc tới nút lá ở xa nhất thì đường đi tìm được cũng chỉ dài bằng chiều cao của cây và độ phức tạp của một lần gọi Adjust là O(log2n). Từ đó có thể suy ra, trong trường hợp xấu nhất, **độ phức tạp của HeapSort cũng chỉ là O(nlog2n).** Việc đánh giá thời gian thực hiện trung bình phức tạp hơn, ta chỉ ghi nhận một kết quả đã chứng minh được là độ phức tạp trung bình của HeapSort cũng là O(nlog2n).

Có thể nhận xét thêm là QuickSort đệ quy cần thêm không gian nhớ cho Stack, còn HeapSort ngoài một nút nhớ phụ để thực hiện việc đổi chỗ, nó không cần dùng thêm gì khác. HeapSort tốt hơn QuickSort về phương diện lý thuyết bởi không có trường hợp tồi tệ nào HeapSort có thể mắc phải. Cũng nhờ có HeapSort mà giờ đây khi giải mọi bài toán có chứa mô-đun sắp xếp, ta có thể nói rằng độ phức tạp của thủ tục sắp xếp đó không quá O(nlog2n).