Chương 6: Sắp xếp

Ths. Phạm Thanh An
Bộ môn Khoa học máy tính - Khoa CNTT
Trường Đại học Ngân hàng TP.HCM

LOGO

Mục tiêu

- Trình bày các thuật toán thông dụng cho việc sắp xếp trong (sắp xếp trên bộ nhớ trong - RAM)
- Minh họa các thuật toán
- Đánh giá thuật toán

Tại sao cần phải sắp xếp dữ liệu

- Chúng ta cần có trật tự yêu cầu nào đó trên tập dữ liệu
- Chúng ta cần thực hiện các phép tìm kiếm nhị phân, chỉ mục một CSDL
- Là bước khởi đầu cho nhiều giải thuật trên tập dữ liệu

SẮP XẾP (SORTING)

- Ví dụ 1:
  - Sắp xếp một danh sách sinh viên theo vần A, B, C
  - Sắp xếp theo thứ tự điểm tổng kết từ cao đến thấp để xét học bổng sinh viên

SẮP XẾP (SORTING)

- Ví dụ 2:
  - Sắp xếp một danh sách cán bộ theo mức thu nhập
  - Sắp xếp danh sách các các em học sinh theo trật tự xếp hàng: thấp đứng trước, cao đứng sau

SẮP XẾP (SORTING)

- Định nghĩa
  - Sắp xếp là quá trình tổ chức lại tập dữ liệu theo một trật tự tăng dần hay giảm dần
- Hai mô hình sắp xếp
  - Sắp xếp trong (internal), các phần tử cần sắp xếp được lưu sẵn trong RAM
  - Sắp xếp ngoài (external), một số các phần tử cần sắp xếp lưu trong RAM, còn lại được lưu ở bộ nhớ ngoài

Các phương pháp sắp xếp

- Các thuật toán cơ bản
  - Thuật toán “Selection sort”
  - Thuật toán “Insertion sort”
  - Thuật toán “Buble sort”
  - Thuật toán “Heap sort”
  - Thuật toán “Quick sort”
- Để tiện trình bày, giả sử sắp xếp các phần tử trên mảng A, N phần tử : A [0], A [1], A [2], …, A [N-1].

Sắp xếp lựa chọn (selection sort)

- Ý tưởng:
  - Giải thuật "*selection sort" sắp xếp một danh sách các giá trị bằng cách lặp lại việc đặt một giá trị cụ thể vào đúng vị trí thích hợp cho nó trong dãy sắp xếp*
  - Nói cách khác, với mỗi vị trí trong danh sách, giải thuật đi tìm giá trị phù hợp cho vị trí đó.

Sắp xếp lựa chọn (Selection sort)

- Ví dụ: sắp xếp một dãy các số nguyên theo trật tự tăng dần, ta làm như sau:
  - Ở bước thứ i, chọn phần tử nhỏ nhất trong dãy a[i], a[i+1], …, a[n]
  - Đổi chỗ phần tử này với phần tử a[i]

Sắp xếp lựa chọn (Selection sort)

44	55	12	42	94	18	06	67
44	55	12	42	94	18	06	67
06	55	12	42	94	18	44	67
06	12	55	42	94	18	44	67
06	12	18	42	94	55	44	67
06	12	18	42	94	55	44	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	67	94

Sắp xếp lựa chọn (Selection sort)

- Giải thuật

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

Sắp xếp lựa chọn (Selection sort)

- Độ phức tạp tính toán
  - Ở bước thứ i, có (n-i) lần so sánh, với i=1…n-1

(n-1) + (n-2) + … + 1 = n(n-1)/2 = **O(n2)**

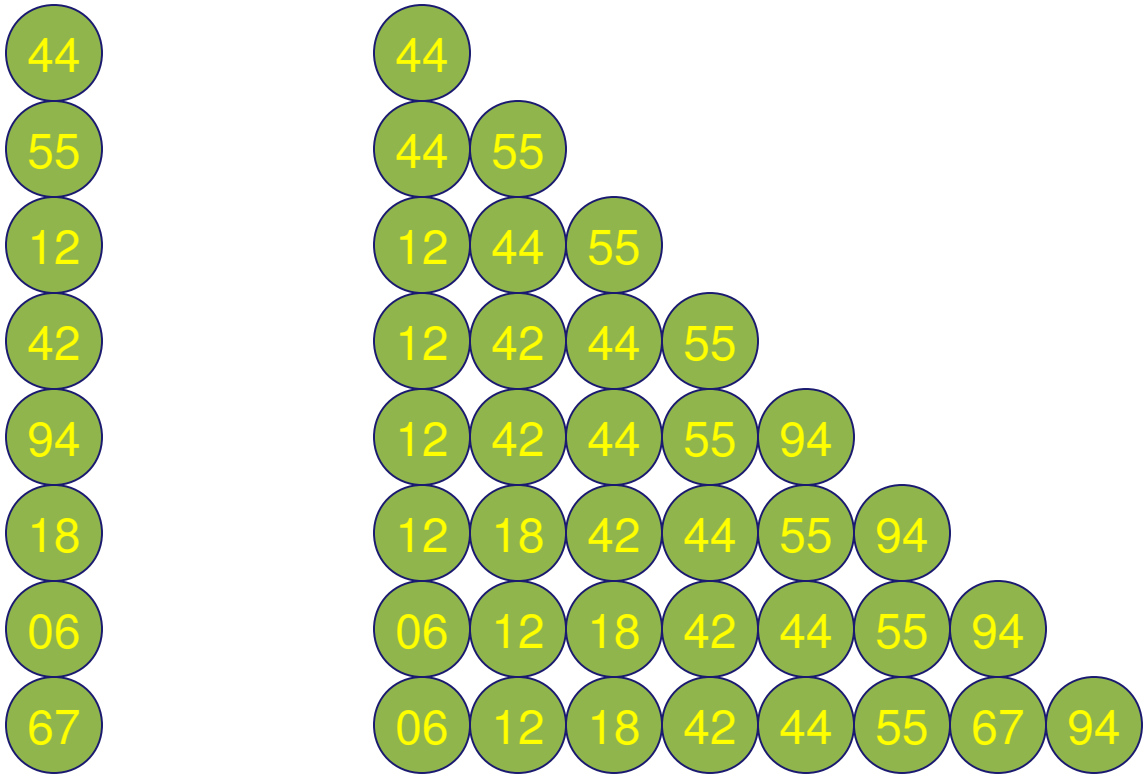
  - Thời gian thực hiện giải thuật T(n) ~ $O(n^2)$

Sắp xếp chèn (Insert sort)

- Ý tưởng: Dựa theo ý tưởng của người chơi bài
  - Giả sử ở bước thứ i các phần tử đã được sắp xếp theo thứ tự khóa ki_1, ki_2, …, ki_{i-1}
  - Xét phần tử thứ i có khóa ki_i, ta lần lượt so sánh với các phần tử đã được sắp sẵn, để tìm vị trí chèn thích hợp

Sắp xếp chèn (Insert sort)

- Ban đầu xem như phần sắp xếp chỉ có 1 phần tử



Sắp xếp chèn (Insert sort)

- Ví dụ

Dãy ban đầu	34	8	64	51	32	21	Moved
Sau i=1	*8*	*34*	64	51	32	21	1
Sau i=2	*8*	*34*	*64*	51	32	21	0
Sau i=3	*8*	*34*	*51*	*64*	32	21	1
Sau i=4	*8*	*32*	*34*	*51*	*64*	21	3
Sau i=5	*8*	*21*	*32*	*34*	*51*	*64*	4

Sắp xếp chèn (Insert sort)

- Giải thuật

```
void InsertionSorting(int a[], int n){
        int x,i,j;
        for (i=1;i<n;i++){
                x=a[i];
                j=i-1;
                while (x<a[j]) && (j>=0){
                        a[j+1]=a[j];
                        j=j-1;
                }
                a[j+1]=x;
        }
}
```

Sắp xếp chèn (Insert sort)

- Độ phức tạp tính toán
  - Ở bước thứ i, có tối đa i-1, tối thiểu 1 phép so sánh
  - Thời gian thực hiện giải thuật T(n) ~ $O(n^2)$
  - Trường hợp xấu nhất có:
  - 1 + 2 + 3 + … + (n-1) = n(n-1)/2 = **$O(n^2)$**

  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

Sắp xếp nổi bọt (Buble Sort)

- Ý tưởng: ở bước i, kể từ phần tử thứ i
  - So sánh hai phần tử kề nhau, nếu khóa của phần tử trước lớn hơn khóa của phần tử sau, thì đổi chỗ cho nhau.
  - Cuối cùng, ta được phần tử có khóa lớn nhất đặt tại vị trí n-i+1

Sắp xếp nổi bọt (Buble Sort)

1 1 23 2 56 9 8 10 100
2 1 2 23 56 9 8 10 100
3 1 2 23 9 56 8 10 100
4 1 2 23 9 8 56 10 100
5 1 2 23 9 8 10 56 100
---- Kết thúc vòng đầu tiên ----
1 2 23 9 8 10 56 100
1 1 2 9 23 8 10 56 100
2 1 2 9 8 23 10 56 100
3 1 2 9 8 10 23 56 100
---- Kết thúc vòng 2 ----

Sắp xếp nổi bọt (Buble Sort)

- Giải thuật

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

Sắp xếp nổi bọt (Buble Sort)

- Độ phức tạp tính toán
  - Ở bước thứ i, có n-i phép so sánh
  - Thời gian thực hiện giải thuật $T(n) \sim O(n^2)$

Sắp xếp nhanh (Quick sort)

- Ý tưởng
  - Xét một dãy n phần tử $a_1, a_2, \ldots, a_n$
  - (1) chọn phần tử x=a[(n+1)div 2] làm khóa
  - (2) đi từ hai đầu của dãy, nếu gặp một cặp a[i]≥x≥a[j] thì hoán vị hai phần tử này
  - (3) tăng i=i+1, giảm j=j-1
  - (4) lặp lại (2) cho đến khi i>j (kết quả thu được phân đoạn AxB)
  - (5) lặp lại (1)-(4) với hai phân đoạn A và B
  - Kết thúc khi tất cả các phân đoạn thu được có chiều dài là 1

Sắp xếp nhanh (Quick sort)

06 18 12 42 94 55 44 67

06 12 18 42 44 55 94 67

06 12 18 42 44 55 67 94

Sắp xếp nhanh (Quick sort)

- Giải thuật

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

Sắp xếp nhanh (Quick sort)

- Nhận xét
  - Nếu chọn phần tử khóa là nhỏ nhất (lớn nhất) có thể dẫn đến tình huống xấu nhất của phương pháp
  - Khi số lượng phần tử thấp, nên dùng phương pháp sắp xếp đơn giản

Sắp xếp nhanh (Quick sort)

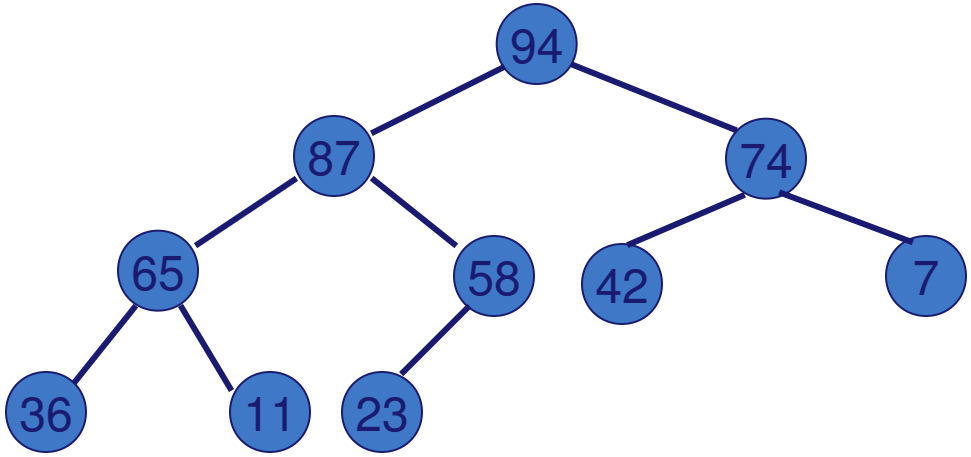
- Đánh giá
  - T(n) thời gian thực hiện QS (n phần tử)
  - P(n) thời gian phân mảng n phần tử thành hai đoạn (1,j) và (i,r)
  - T(n)=P(n)+T(1..j)+T(i..r), P(n)=Cn
  - Trường hợp tốt nhất, mỗi bước phân thành hai đoạn có chiều dài bằng nhau
    - T(n) = 2T(n/2)+Cn ~ O(nlogn)
  - Trường hợp xấu nhất, mỗi bước phân mảng r thành hai đoạn có chiều dài r-1 và 1
    - T(n) = Cn+T(n-1)+T(1) ~ $O(n^2)$

Heap sort

- Định nghĩa
  - Dãy $h_1,\ldots,h_n$ gọi là một ***HEAP***, nếu thỏa mãn
    - $h_i > h_{2i+1}$
    - $h_i > h_{2i+2}$
  - Chú ý
    - Các phần tử có chỉ số [n/2],…,n-1 là nút lá
    - ***HEAP*** có n phần tử thì có [n/2] nút trong

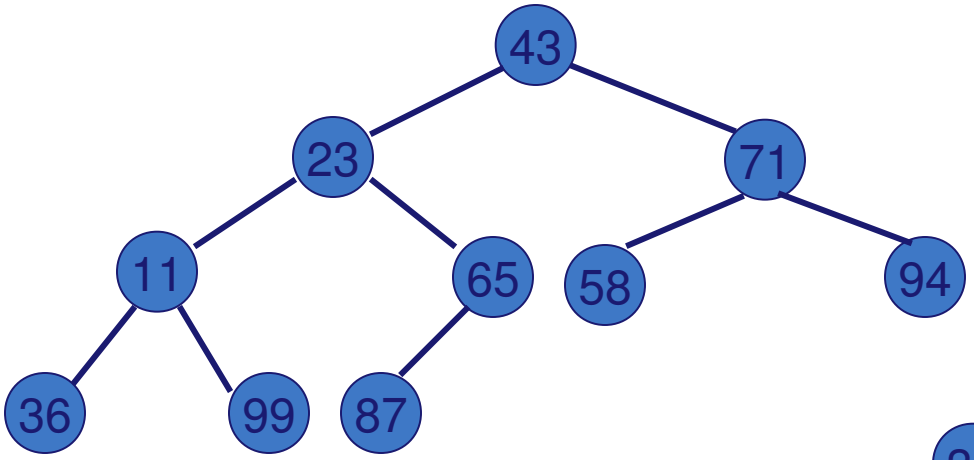
Heap sort

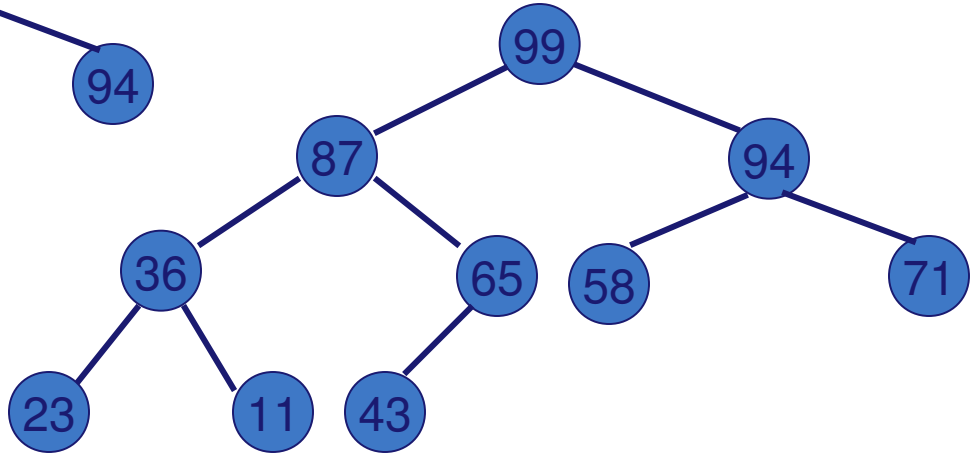
- Heap sort



Heap sort

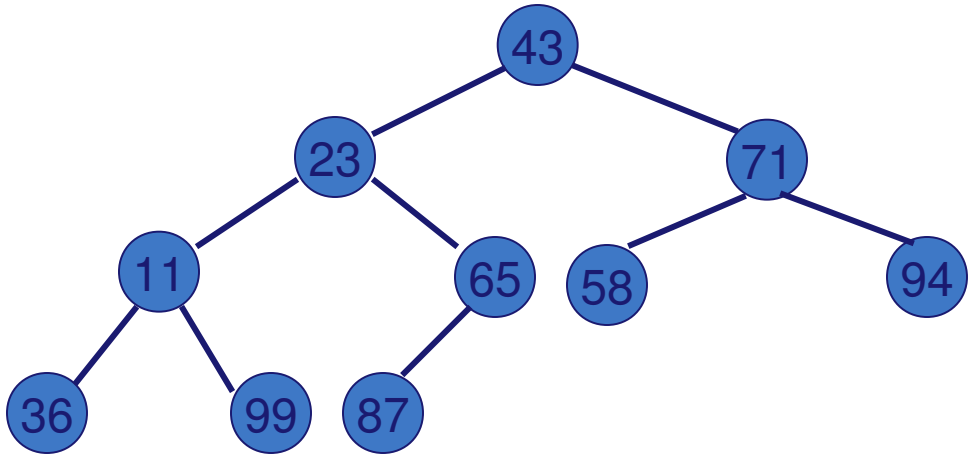
- Heap sort
  - Xét dãy 43,23,71,11,65,58,94,36,99,87





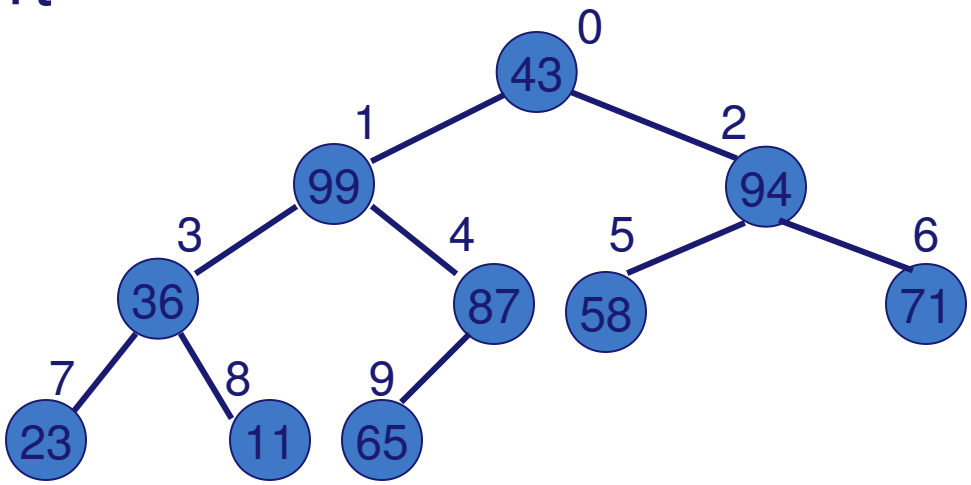
Heap sort

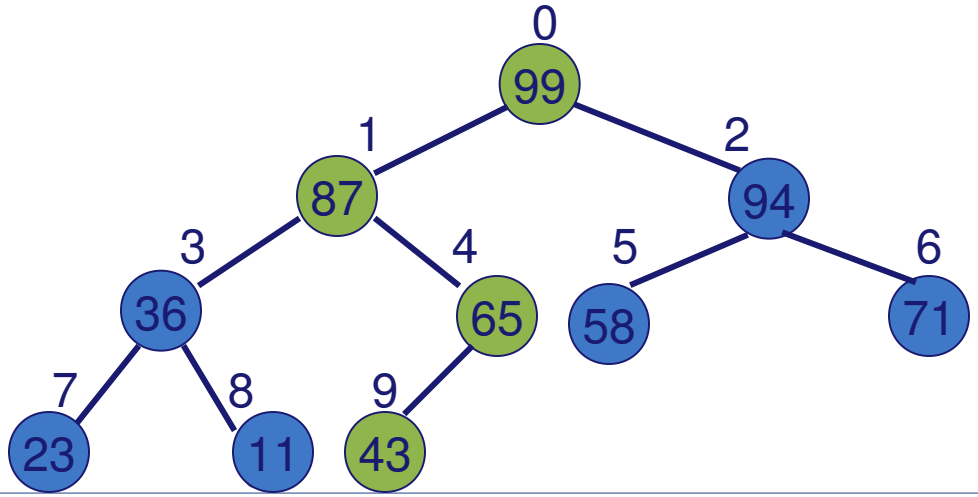
- Phương pháp tạo HEAP từ đáy lên
- Việc tạo HEAP bắt đầu từ các nút trong (nút số 0 đến nút số [n/2]-1)



Heap sort

- Heap sort





Heap sort

- Giải thuật

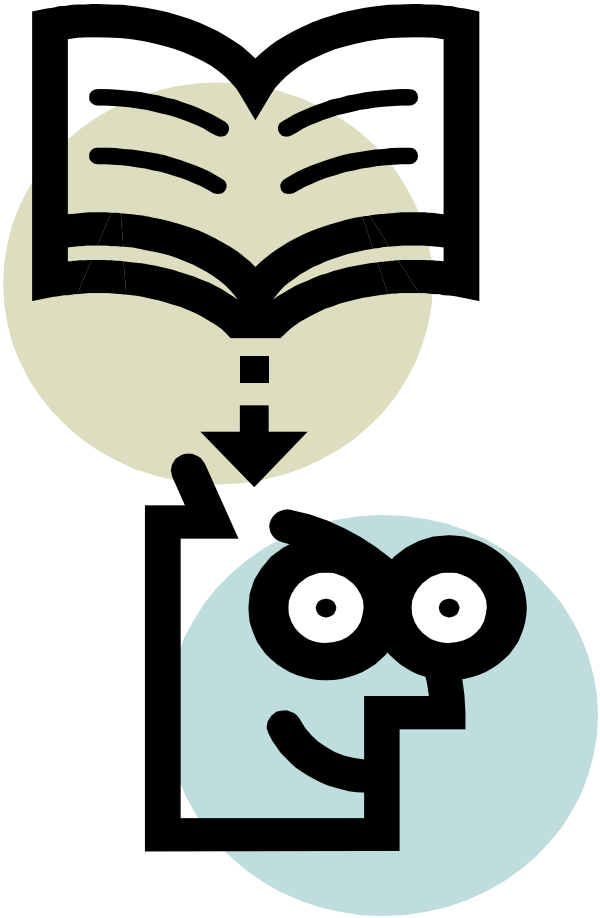
```
void SetupHeap(int a[], int k, int n) // điều chỉnh phần tử thứ k
{
        int x=a[k];
        while (k<n/2){
                int j=2*k+1;
                if (j+1<n && a[j]<a[j+1]) j++;
                if (x>a[j]) break;
                a[k]=a[j];k=j;
        }
        a[k]=x;
}
void MakeHeap(int a[], int n){ // tạo đống
        for (int i=n/2-1;i>=0;i--)
        SetupHeap(a, i, n);
}
```

Heap sort

- Giải thuật

```
void Heapsort()
    {
        int tmp;
        makeheap(a,n)
        for (int i=n-1;i>0;i--){
            tmp=a[0];a[0]=a[i];a[i]=tmp;
        setupHeap(a,0,i);
        }
    }
```

Heap sort

- Heap sort
  - Nhận xét
    - Thời gian thực hiện SetupHeap là O(logn)
    - Thời gian thực hiện MakeHeap là O(nlogn)
    - Thời gian thực hiện HeapSort là O(nlogn)

Q&A