TRƯỜNG ĐIỆN – ĐIỆN TỬ

KHOA ĐIỆN TỬ

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CÁC GIẢI THUẬT SẮP XẾP & TÌM KIẾM

Bài toán sắp xếp trên cấu trúc mảng

- Sắp xếp cơ bản
  - Sắp xếp kiểu lựa chọn (Selection - sort)
  - Sắp xếp chèn/thêm dần (Insertion- sort)
  - Sắp xếp đổi chỗ/nổi bọt (Exchange/Bubble - sort)
- Sắp xếp nâng cao (sắp xếp nhanh)
  - Sắp xếp nhanh (Quick-sort)
  - Sắp xếp trộn (Merge-sort)
  - Sắp xếp vun đống (Heap-sort)

Điều kiện bài toán sắp xếp

- Dãy A có n phần tử : A[1], A[2],…,A[n]
- Sắp xếp dãy A theo thứ tự tăng dần hoặc giảm dần

Sắp xếp kiểu thêm dần/chèn (Insertion sort)

No.	Số so sánh	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
		32	51	27	83	66	11	45	75
1	51	32	51	27	83	66	11	45	75
2	27	27	32	51	83	66	11	45	75
3	83	27	32	51	83	66	11	45	75
4	66	27	32	51	66	83	11	45	75
5	11	11	27	32	51	66	83	45	75
6	45	11	27	32	45	51	66	83	75
7	75	11	27	32	45	51	66	75	83

Giải thuật sắp xếp chèn

- Bước 1: Xét A[1] là dãy con ban đầu đã được sắp xếp
- Bước 2: Xét A[2], nếu A[2] < A[1] chèn vào trước A[1], còn lại thì giữ nguyên A[2] tại chỗ
- Bước 3: Xét A[1], A[2] là dãy con được sắp xếp
- Bước 4: Xét A[3], so sánh với A[1], A[2] và tìm vị trí chèn
- Bước 5: Lặp lại với A[4], … đến khi dãy được sắp xếp hết

Giải thuật sắp xếp chèn

```
Procedure INSERT_SORT(A,n)
    For i = 1 to n-1 {
        temp = A[i];
        j = i-1;
        While temp<=A[j] do {
            A[j+1] = A[j];
            j = j-1;
        }
        A[j+1] = temp;
}
 Return;
```

Đánh giá độ phức tạp

$T(n) = O(n^2)$

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

Sắp xếp đổi chỗ/nổi bọt (Exchange/Bubble sort)

		1	2	3	4	5	6	7
A[1]	32	11	11	11	11	11	11	11
A[2]	51	32	27	27	27	27	27	27
A[3]	27	51	32	32	32	32	32	32
A[4]	83	27	51	45	45	45	45	45
A[5]	66	83	45	51	51	51	51	51
A[6]	11	66	83	66	66	66	66	66
A[7]	45	45	66	83	75	75	75	75
A[8]	75	75	75	75	83	83	83	83

- Chiến thuật: Dựa trên việc so sánh cặp phần tử liên kề nhau và tráo đổi vị trí nếu chúng không theo thứ tự

Sắp xếp đổi chỗ/nổi bọt (Exchange/Bubble sort)

- Nguyên tắc
  - Duyệt bảng khoá (danh sách khoá) từ đáy lên đỉnh
  - Dọc đường nếu thứ tự 2 khoá liền kế không đúng => đổi chỗ
- Ví dụ:
  - 1: 25, 36, 31, 60, 16, 88, 49, 70
  - 2: **16**, 25, 36, 31, 60, **49**, 88, 70
  - 3: 16, 25, **31**, 36, **49**, 60, **70**, 88
- Nhận xét
  - Khoá nhỏ nổi dần lên sau mỗi lần duyệt => "nổi bọt"
  - Sau một vài lần (không cần chạy n bước), danh sách khoá đã có thể được sắp xếp => Có thể cải tiến thuật toán, dùng 1 biến lưu trạng thái, nếu không còn gì thay đổi (không cần đổi chỗ) => ngừng

Giải thuật sắp xếp nổi bọt (giải thuật gốc)

```
Procedure Bubble_sort(A,n)
For i = 1 to n-1 do
    For j = n  down to i+1 do
        If A[j] < A[j-1] {
            Đổi chỗ A[j] và A[j-1]
        }
Return;
```

Đánh giá độ phức tạp

$T(n) = O(n^2)$

```
void Bubble_sort (int a[], int n)
{
for (int i = 0; i<=n - 2; i++){
        for (int j = n - 1; j>i; j--){
            if (a[j] < a[j - 1]){
                swap(a[j], a[j - 1]);
            }
        }
    }
}
```

Giải thuật sắp xếp nổi bọt (giải thuật cải tiến)

```
Procedure Bubble_sort(A,n)
i = 1;
sorted = False;
while (!sorted && i<N) {
  sorted = True;
  for (k=N-1;k>=i;k--)
      if (A[k] > A[k+1] {
         swap(A[k], A[k+1]);
         sorted = False;
      }
  i++;
}
Return
```

```
void bubbleSort(int A[], int N) {
    int i = 0;
    bool sorted = false;
    while (!sorted && i<N-1) {
        sorted = true;
        for (int k=N-2;k>=i;k--)
            if (A[k] > A[k+1]) {
                swap(A[k], A[k+1]);
                sorted = false;
            }
        i++;
    }
}
```

Sắp xếp nhanh (Quick sort)

- Hiệu năng thực thi tốt hơn
  - Chia để trị
  - Giải thuật sắp xếp đệ quy
- Phần tử được chọn là bất kỳ được gọi là "chốt" (pivot)
- Mọi phần tử nhỏ hơn chốt sẽ được đẩy lên phía trước chốt
- Hai mảng con:
  - Mảng con nhỏ hơn chốt ở phía trước chốt
  - Mảng con lớn hơn chốt ở phía sau chốt
- Chiến thuật tương tự với từng mảng con, đến khi mảng con chỉ còn một phần tử

Sắp xếp nhanh (Quick sort)

- Thuật toán cụ thể
  - 1: Thường chọn phần tử đầu tiên làm chốt.
  - 2: Tìm vị trí thực của khóa chốt để phân thành 2 đoạn:
    - 1: Dùng 2 chỉ số: i, j chạy từ hai đầu dãy số. Chạy i từ đầu dãy số trong khi khóa còn nhỏ hơn khóa chốt
    - 2: Nếu khóa >= khóa chốt: Lưu phần tử hiện tại = X và chạy j.
    - 3: Chạy j trong khi khóa lớn hơn khóa chốt
    - 4: Nếu khóa <= khóa chốt: dừng và đổi chỗ phần tử hiện tại cho X
    - 5: Tiếp tục thực hiện như vậy cho đến khi i>j thì đổi chỗ Kj cho khóa chốt. Lúc đó khóa chốt sẽ nằm đúng vị trí.
  - 3: Làm giống như vậy cho các đoạn tiếp theo

Sắp xếp nhanh (Quick sort)

- Xét mảng A có các phần tử sau:

  75, 70, 65, 84, 98, 78, 100, 93, 55, 61, 81, 68

- Bước 1: Giả sử chọn 75 làm chốt
- Bước 2: Thực hiện phép tìm kiếm các số nhỏ hơn 75 và lớn hơn 75
- Bước 3: Thu được 2 mảng con sau

  70, 65, 55, 61, 68 và 84, 98, 100, 93, 81

- Bước 4: Quá trình sắp xếp tương tự với 2 mảng con trên

Sắp xếp nhanh (Quick sort)

75	70	65	84	**X1** 98	78	100	93	55	61	81	68
			i = 4								j =12
75	70	65	68	98	78	100	93	55	61	81	84
75	70	65	68	98	**X2** 78	100	93	55	61	81	84
				i = 5					j = 10		
75	70	65	68	61	78	100	93	55	98	81	84
75	70	65	68	61	78 **X3**	100	93	55	98	81	84
					i = 6	i = 7		j = 9			
75	70	65	68	61	55	100	93	78	98	81	84
					j = 6						
55	70	65	68	61	75	100	93	78	98	81	84

Sắp xếp nhanh (Quick sort)
Giải thuật

- Hàm phân đoạn

```
Function Partition(A, first,last){
   if (first>=last) return;
   c=A[first];//phần tử chốt
   i=first+1,j=last;
   while (i<=j){
        while (A[i]<=c && i<=j) i++;
        while (A[j]>c && i<=j) j--;
        if (i<j) swap(A[i],A[j]);
   }
   swap(A[first], A[j]);
   return j;
}
```

- Hàm sắp xếp

```
Procedure QuickSort(A, first, last){
       if( first < last )
    {
       j = Partition( A, first, last);
      QuickSort(A, first, j-1);
      QuickSort(A, j+1, last);
    }
}
```

Sắp xếp nhanh (Quick sort)
Cài đặt giải thuật

- Hàm phân đoạn

```
int Partition(int A[], int first, int last){
   if (first>=last) return 0;
   int c=A[first];
   int i=first+1,j=last;
   while (i<=j){
       while (A[i]<=c && i<=j) i++;
       while (A[j]>c && i<=j) j--;
       if (i<j) swap(A[i], A[j]);
   }
   swap(A[first], A[j]);
   return j;
}
```

- Hàm sắp xếp

```
void QuickSort(int A[], int first, int last){
   int j;
    if( first < last )
    {
        j = Partition2( A, first, last);
        QuickSort(A, first, j-1);
        QuickSort(A, j+1, last);
    }
}
```

Độ phức tạp của giải thuật Quick_sort

- Độ phức tạp trong trường hợp xấu nhất là

  T(n) = O(n2)

- Độ phức tạp trong trường hợp tốt nhất là

  T(n) = O(nlog2n)

- Độ phức tạp trung bình

  T(n) = O(nlog2n)

- Khi n lớn thì Quick_sort có hiệu năng tốt hơn các phương pháp còn lại

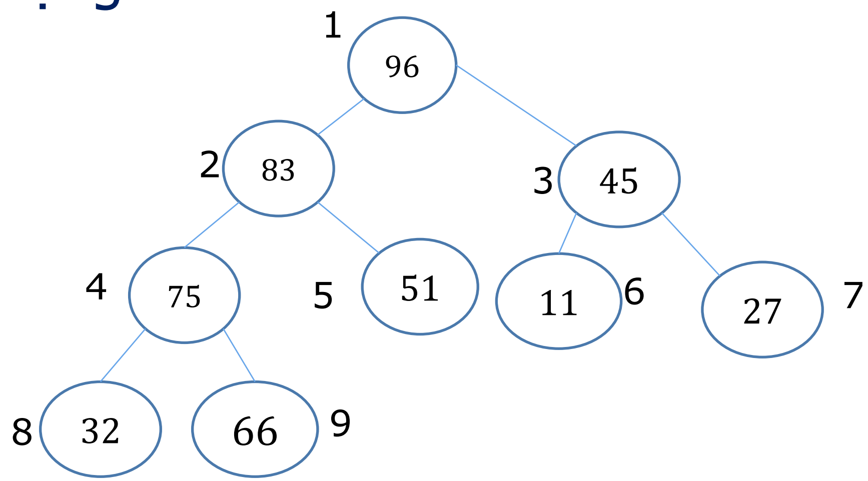
Sắp xếp vun đống (Heap-sort)

- Cấu trúc đống
- Phép tạo đống
- Sắp xếp kiểu vun đống (Heap – sort)

Cấu trúc đống

- Đống (dạng max-heap): là một cây nhị phân mà mỗi nút gắn với một số sao cho số ở nút cha không nhỏ hơn số ở nút con
- Đống (dạng min-heap): là một cây nhị phân mà mỗi nút gắn với một số sao cho số ở nút cha không lớn hơn số ở nút con
- Ví dụ: dùng cây nhị phân hoàn chỉnh
  – Số ứng với gốc của đống chính là số lớn nhất
  – Biểu diễn trong máy dưới dạng vector như sau

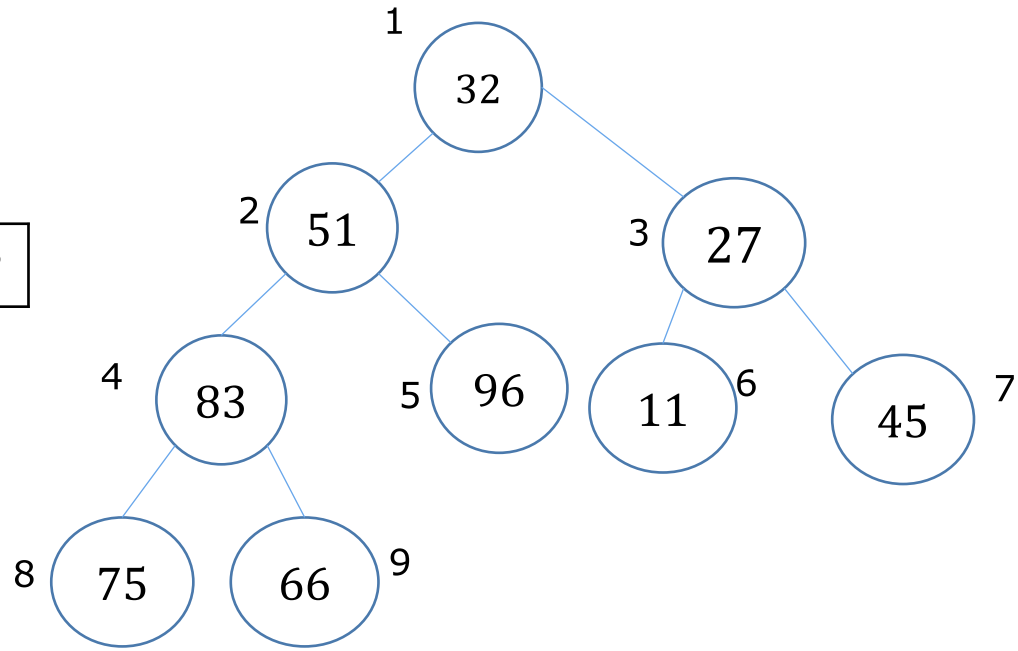
96	83	45	75	51	11	27	32	66
1	2	3	4	5	6	7	8	9



Phép tạo đống

- Đặt vấn đề: Một dãy số biểu diễn dạng vector trong máy (lưu trữ tuần tự) có thể biểu diễn dưới dạng cây nhị phân hoàn chỉnh và ngược lại. Tuy nhiên cây này chưa phải là đống
- Tạo đống như thế nào ?

32	51	27	83	96	11	45	75	66
1	2	3	4	5	6	7	8	9



Phép tạo đống

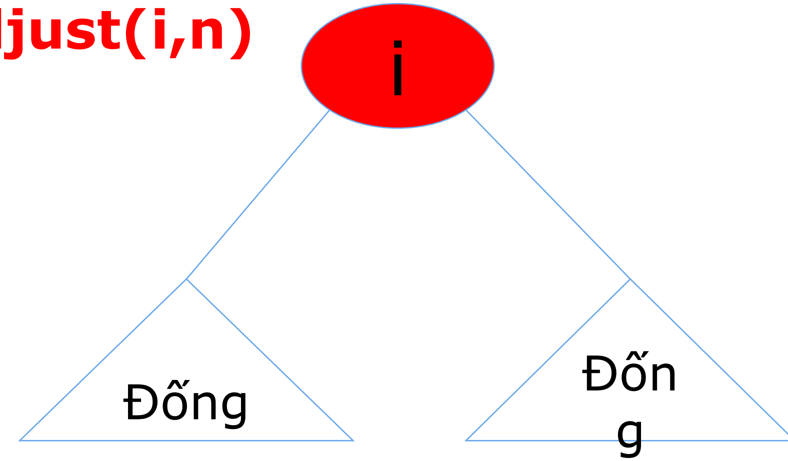
- Nếu một cây nhị phân hoàn chỉnh là đống thì cây con cũng là đống
  - Cây nhị phân hoàn chỉnh có n nút thì chỉ có n/2 nút cha.
  - Nút lá bao giờ cũng coi là đống
- Bài toán: tạo đống cho cây nhị phân hoàn chỉnh, gốc cây này có thứ tự là i (theo cách lưu trữ tuần tự) và hai nút con của nút gốc này đã là đống rồi

Giải thuật tạo đống

- Tiến hành theo kiểu từ dưới lên (bottom up)
- Lá là đống, nên tạo đống cho cây con mà gốc của nó có số thứ tự từ n/2 trở xuống
  - i: là thứ tự của nút gốc cây con cần xét
  - n: là số nút trên cây nhị phân hoàn chỉnh
- Có n nút biểu diễn bởi vector A, lệnh sau đây thực hiện tạo cây nhị phân hoàn chỉnh thành đống

For i = n/2 down to 1 Call Adjust(i,n)

- Hàm tạo đống: Adjust(i,n)



Giải thuật tạo đống

```
void adjust(int A[], int i, int N)
{
    int max = i; // khoi tao max nhu la root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2
    // Neu nut con trai lon hon so voi root
    if (l < N && A[l] > A[max])
        max = l;
    // Neu nut con phai lon hon so voi root
    if (r < N && A[r] > A[max])
        max = r;
    // Neu root khong phai la lon nhat
    if (max != i)
    {
        swap(A[i], A[max]);

        // De quy lai ham adjust
        adjust(A, max, N);
    }
}
```

Giải thuật sắp xếp vun đống

- Từ dãy số đã cho, quan niệm như một vector biểu diễn cho một cây hoàn chỉnh, tạo nên đống đầu tiên -> giai đoạn tạo đống ban đầu
- Đổi chỗ giữa số ở đỉnh đống với số ở cuối đống sau đó vun đống mới
- Quá trình này sẽ được lặp lại cho đến khi đống chỉ còn 1 nút -> giai đoạn sắp xếp

Giải thuật sắp xếp vun đống

```
Procedure HEAP-SORT(A, N){
      for i = [n/2] down to 1
            Call Adjust(A, i, N);
      for i = N down to 1 {
      swap(A[1],A[i]); call Adjust(A, 1, i-1);
      }
}
```

Giải thuật sắp xếp vun đống

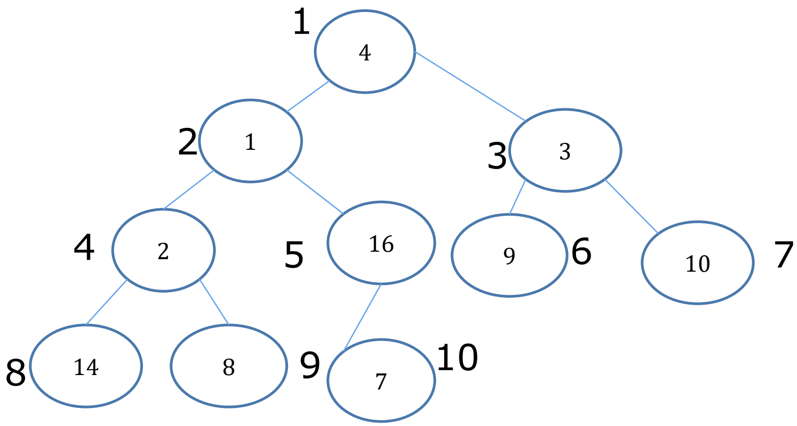
```
//giai thuat sap xep vun dong
void heap_sort(int A[], int N) {
    // Tao dong ban dau
    for (int i = N / 2 - 1; i >= 0; i--)
        adjust(A, i, N);
    // Trích xuất từng phần tử một từ heap
    for (int i = N - 1; i >= 0; i--)
    {
        // Di chuyen root ve cuoi cung
        swap(A[0], A[i]);

        // vun lại đống cho cây con gồm i phần tử
        adjust(A, 0, i);
    }
}
```

Sắp xếp dãy số sau

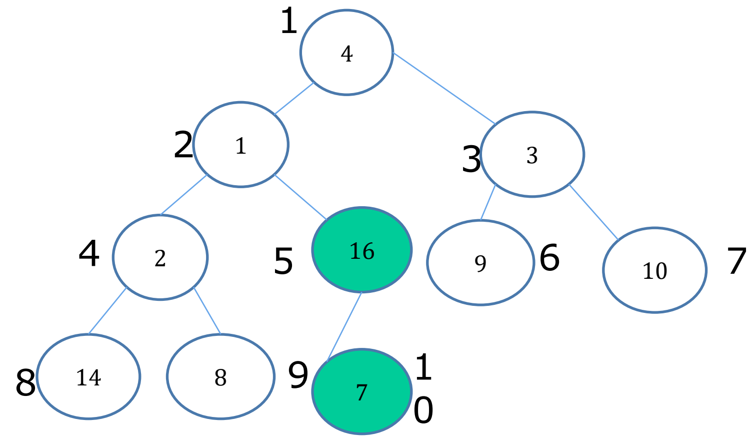
4	1	3	2	16	9	10	14	8	7

A[1] A[10]

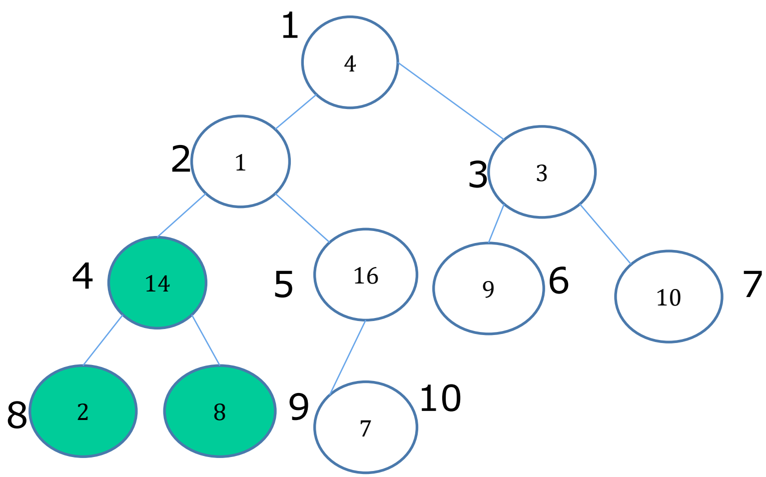
Mô tả các bước vun đống



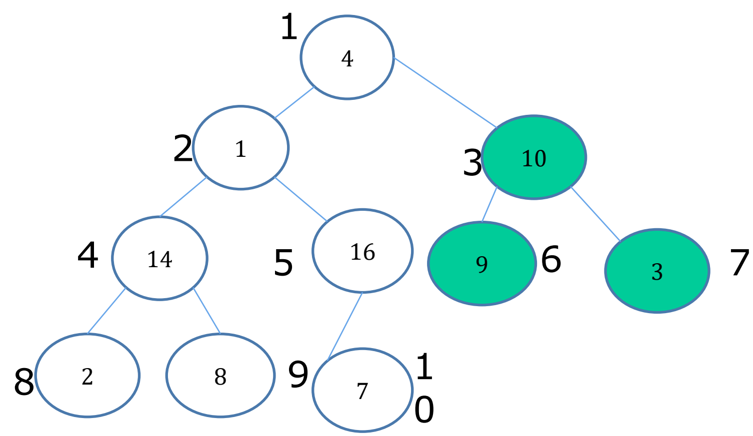
Cây nhị phân ban đầu



Thực hiện Adjust(5,10)

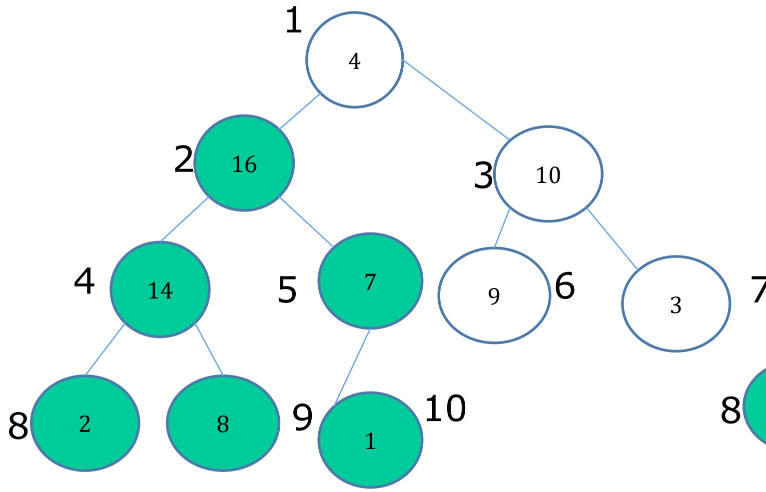


Thực hiện Adjust(4,10)

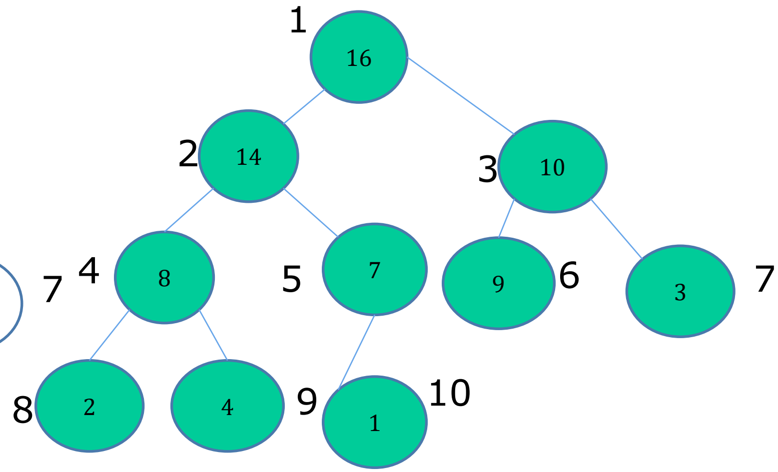


Thực hiện Adjust(3,10)

Mô tả các bước vun đống



Thực hiện Adjust(2,10)

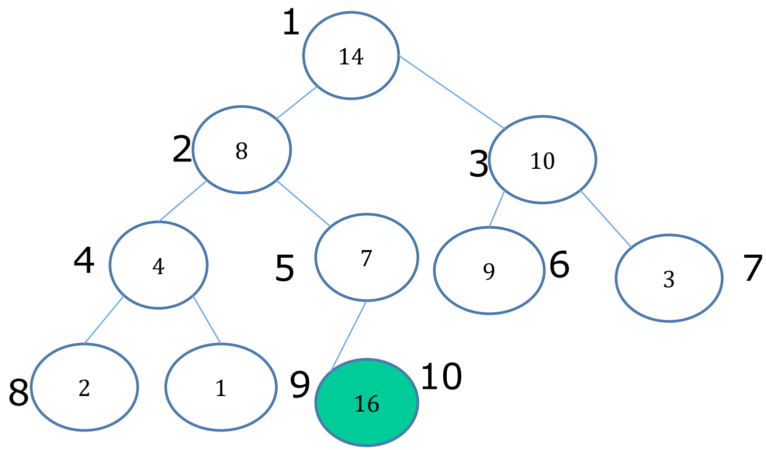


Thực hiện Adjust(1,10)

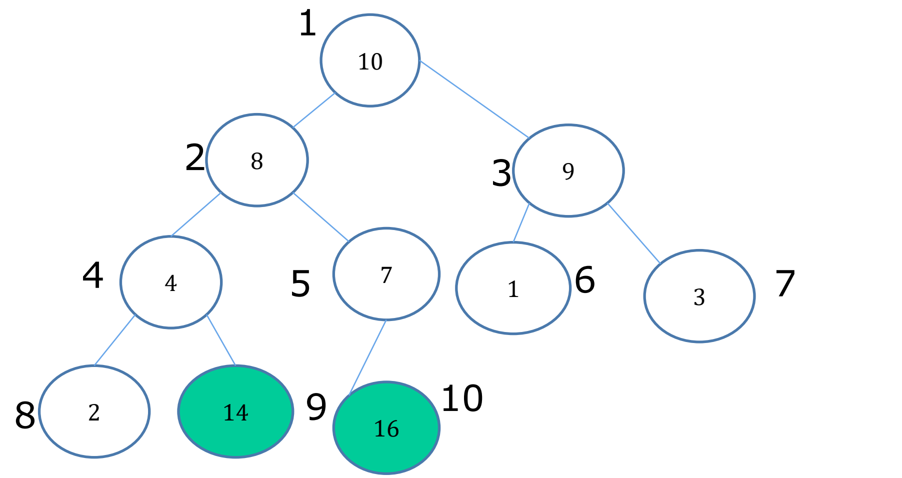
Lúc này cây đã là đống và biểu diễn trong máy là vector A như sau

16	14	10	8	7	9	3	2	4	1

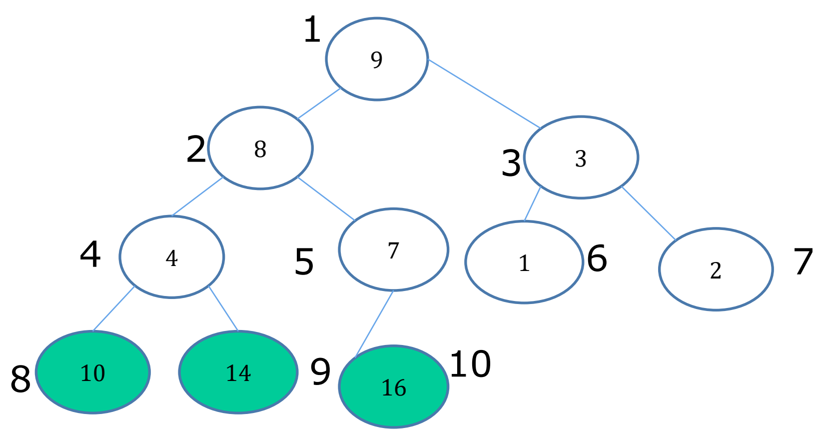
Sắp xếp



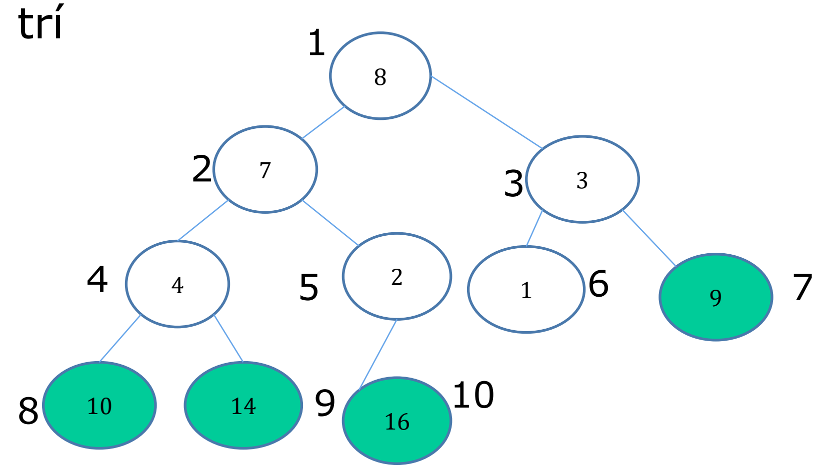
Đổi lần 1: giữa A[1] và A[10], vun đống cho cây với 9 nút còn lại, 16 đã vào đúng vị trí



Đổi lần 2: giữa A[1] và A[9], vun đống cho cây với 8 nút còn lại, 14, 16 vào đúng vị trí

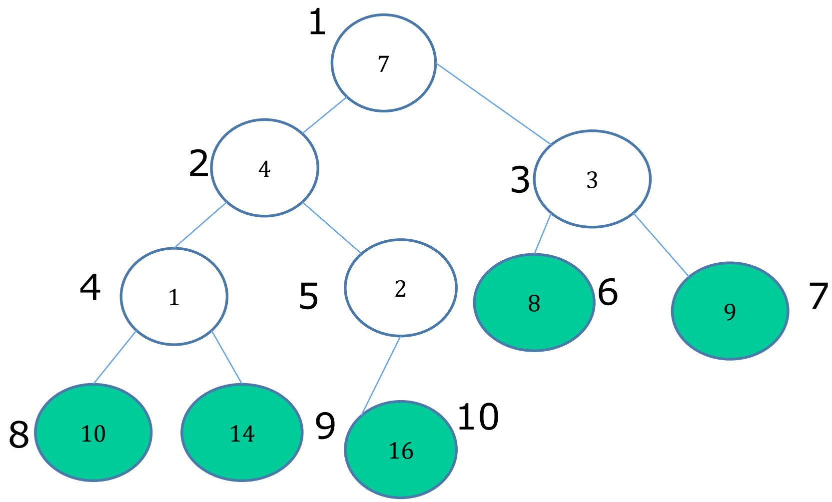


Đổi lần 3: giữa A[1] và A[8], vun đống cho cây với 7 nút còn lại, 10,14, 16 vào đúng vị trí

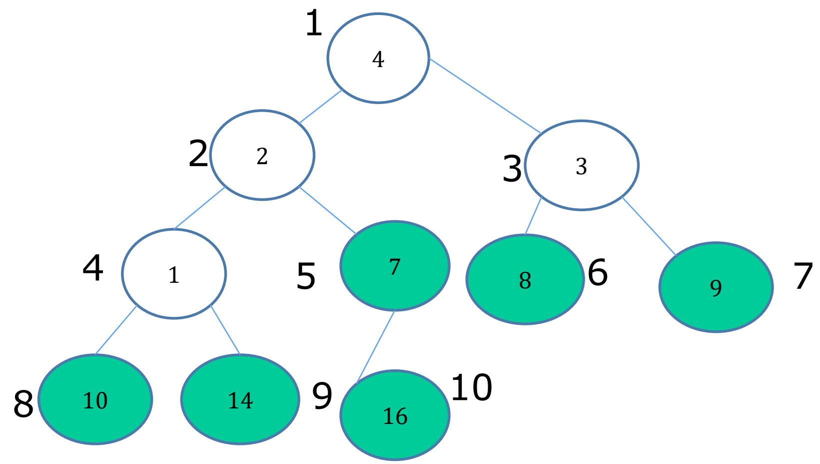


Đổi lần 4: giữa A[1] và A[7], vun đống cho cây với 6 nút còn lại

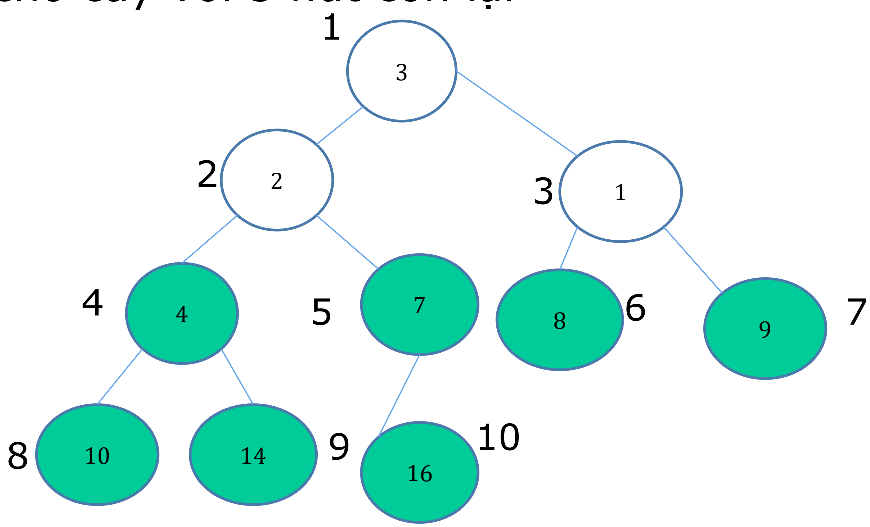
Sắp xếp



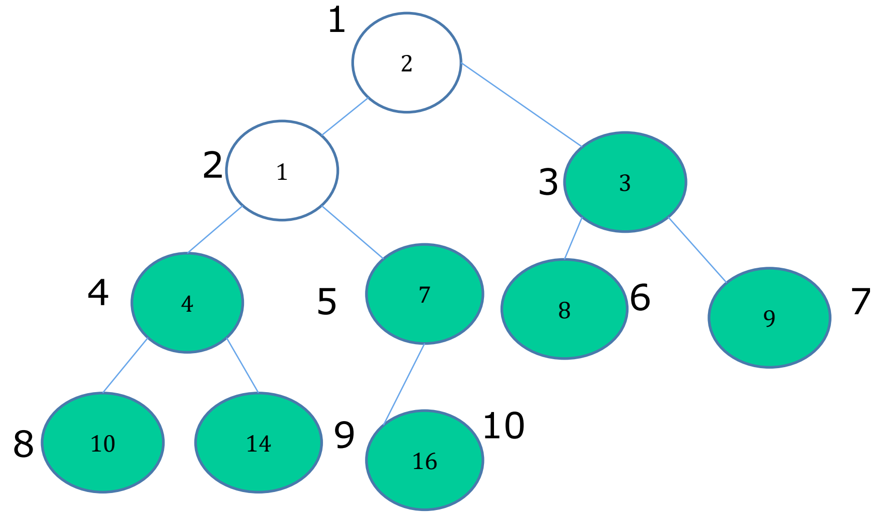
Đổi lần 5: giữa A[1] và A[6], vun đống cho cây với 5 nút còn lại



Đổi lần 6: giữa A[1] và A[5], vun đống cho cây với 4 nút còn lại

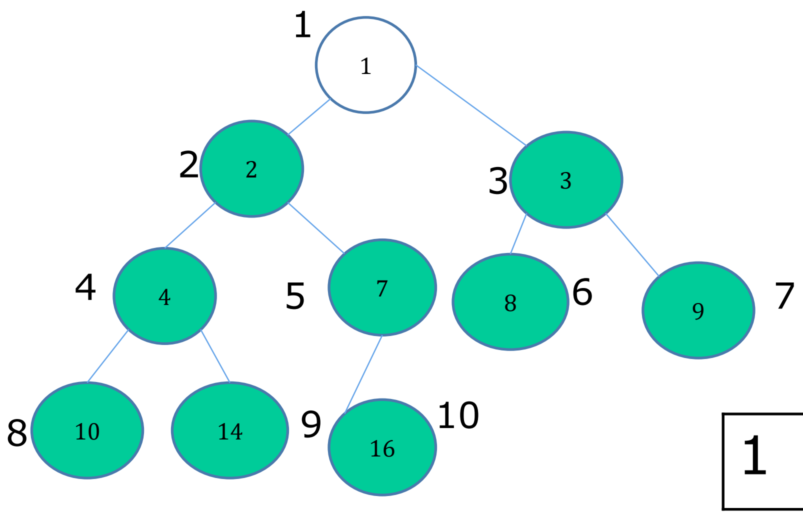


Đổi lần 7: giữa A[1] và A[4], vun đống cho cây với 3 nút còn lại



Đổi lần 8: giữa A[1] và A[3], vun đống cho cây với 2 nút còn lại

Sắp xếp



1	2	3	4	7	8	9	10	14	16

Đổi lần 9: giữa A[1] và A[2], đống chỉ còn 1 nút. Dãy A đã được sắp xếp

$$T(n) = O(n\log n)$$

Giải thuật sắp xếp trộn (Merge Sort)

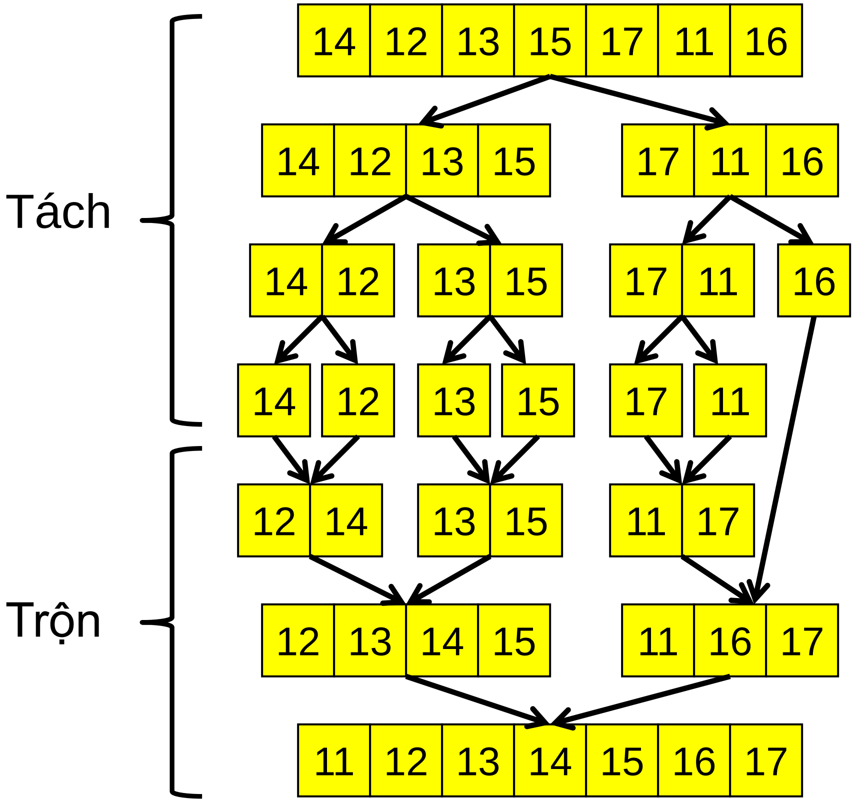
- Ý tưởng giải thuật: sử dụng giải thuật đệ quy như sau:
  - Chia dãy ban đầu thành hai dãy con có kích thước khác nhau không quá 1
  - Sắp xếp hai dãy con trên
  - Trộn hai dãy con đã sắp xếp để được dãy ban đầu cũng được sắp xếp
  - Điểm dừng: khi kích thước của dãy không > 1

Giải thuật sắp xếp trộn (Merge Sort)

- Đầu vào:
  – 2 dãy $A=(a_0, a_1, \ldots, a_{m-1})$ và $B=(b_0, b_1, \ldots, b_{n-1})$ đã được sắp xếp tăng dần
- Đầu ra:
  – Dãy $C=(c_0, c_1, \ldots, c_{m+n-1})$ là kết quả trộn của A và B và cũng được sắp xếp
- Giải thuật:
  – Khởi tạo: dùng 2 biến chạy i=j=0; C = rỗng
  – Chừng nào (i<m và j<n) // 2 dãy A và B còn chưa được chạy hết
    - Nếu $a_i \le b_j$ thì { $C = C + a_i$; i++; }
    - Trái lại, thì $C=C+(b_j, b_{j+1}, \ldots, b_{n-1})$
  – Nếu $i \geq m$ (dãy A đã được chạy hết) thì $C=C+(b_j, b_{j+1}, \ldots, b_{n-1})$
  – Trái lại thì $C=C+(a_i, a_{i+1}, \ldots, a_{m-1})$

Giải thuật sắp xếp trộn (Merge Sort)

- Ví dụ minh họa



Giải thuật sắp xếp trộn (Merge Sort)-Cài đặt

- Thủ tục trộn

```
//Tron 2 day con ma da duoc sap xep trong A
//L1=A[m],A[m+1],...,A[n]; //L2=A[n+1],A[n+2],...,A[p]
void MergeArrays(int A[],int first, int mid, int last)
{
      int i= first,j= mid +1;
      while (i<j && j<=last){
            if (A[i]<=A[j]) i++;
            else {//chen Aj vao vi tri i
                  int x=A[j];
                  for (int k=j-1;k>=i;k--)
                        A[k+1]=A[k];
                  A[i]=x;
                  i++; j++;
            }
      }
}
```

Giải thuật sắp xếp trộn (Merge Sort)-Cài đặt

- Các thủ tục sắp xếp trộn

```
void Merge(int A[], int first, int last){
      if (first>=last) return;
      int mid=(first+last)/2;
      Merge(A,first,mid);
      Merge(A,mid+1,last);
      MergeArrays(A,first,mid,last);
}

void MergeSort(int A[], int N){
      if (N<2) return;
      Merge(A,0,N-1);
}
```

Bài toán tìm kiếm trên cấu trúc mảng

- Tìm kiếm một phần tử theo một tiêu chí nào đó
- Tìm kiếm "được thỏa" khi có hoặc "không thỏa" khi không có phần tử nào
  - Tìm kiếm tuần tự
  - Tìm kiếm nhị phân
- Ví dụ: mảng A gồm n phần tử A[1], A[2]…A[n] Cho số X, tìm xem có giá trị nào trong A bằng X hay không?

Tìm kiếm tuần tự

- Duyệt tất cả các phần tử trong mảng A
  - Nếu có phần tử nào bằng X thì ghi nhận lại chỉ số của phần tử đó,
  - Nếu không có thì ghi nhận bằng 0.

Function Sequential(A,n,X)

1. i = 1;
2. While A[i] =! X do i = i+1;
3. If i = n+1 {

   Sequential = 0;

   Return "không tìm thấy"}
4. Else {

   Sequential = 1;

   Return "tìm thấy"}

Tìm kiếm nhị phân

- Tìm kiếm với mảng A đã được sắp xếp theo thứ tự tăng hoặc giảm dần.
- Ví dụ: xét mảng A có các phần tử được sắp xếp tăng dần
- So sánh X với phần tử A[k] ở giữa mảng
  - Nếu X< A[k] tìm kiếm với nửa đầu của mảng A
  - Nếu X> A[k] tìm kiếm với nửa cuối của mảng A
  - Nếu X = A[k] tìm kiếm được thỏa

Giải thuật tìm kiếm nhị phân

```
Function Binary_search(A,n,X)
1.  F=1; L = n;
2.  While F<=L {
        m = (F+L)div 2;
        if X=A[m] then return m;
        else If X< A[m] then L = m-1;
        else  F = m+1;
        }
3.   return (0);
```

Cài đặt giải thuật tìm kiếm nhị phân

```
int Binary_search(int A[],int n,int X)
{
        int F=0, L = n-1;
        while (F<=L) {
                int m = (F+L)/2;
                if (X==A[m])
                return m;
                else if (X< A[m])
                L = m-1;
                else
                F = m+1;
        }
        return -1;
  }
```

Giải thuật tìm kiếm nhị phân (Đệ quy)

```
Function Binary_search(A,F,L,X)
//F, L là chỉ số đầu và cuối của dãy
//LOC là chỉ số ứng với khóa cần tìm, nếu tìm không
if L>=F then {
      m = (F+L)/2;
      if X=A[m] then LOC=m
      else if X<A[m] then LOC= Binary_search(A,F,m-1,X);
      else  LOC= Binary_search(A,m+1,L,X);
      return LOC;
}
return -1;
```

Cài đặt giải thuật tìm kiếm nhị phân (Đệ quy)

```
int Binary_search_DQ(int A[],int F, int L,int X)
{
      int LOC,m;
      if (L>=F){
            m = (F+L)/2;
            if (X== A[m])
                  LOC=m;
            else if (A[m]>X)
                  LOC = Binary_search_DQ(A, F,  m-1, X);
            else
                  LOC = Binary_search_DQ(A, m+1,  L, X);
            return LOC;
      }
      return -1;
 }
```

Tìm kiếm nhị phân

Giả sử X = 75

0	1	2	3	4	5	6	7	8	9	10	11	12	l	r
11	22	30	33	40	44	55	60	66	77	80	88	99	0	12
							60	66	77	80	88	99	7	12
							60	66					7	8
								66					8	8

Chỉ phần tử ứng với m

l = r = 8 giải thuật kết thúc, giá trị trả về bằng 0, tìm kiếm không thỏa.

Đánh giá độ phức tạp

- Giải thuật tìm kiếm tuần tự
- $T(n) = O(n)$
- Giải thuật tìm kiếm nhị phân
- $T(n) = O(log2(n))$

Tổng kết

- Độ phức tạp của các thuật toán sắp xếp và tìm kiếm

Thuật toán	Độ phức tạp
SelectSort	$O(n^2)$
InsertSort	$O(n^2)$
BubbleSort	$O(n^2)$
QuickSort	$O(n\log_2 n)$
HeapSort	$O(n\log_2 n)$
LinearSearch	$O(n)$
BinarySearch	$O(\log_2 n)$

Bài tập

- Bài 1: Cho dãy số A gồm 8 phần tử : 77,33,44,11,88,22,66,55. Áp dụng các phương pháp sắp xếp đã học để sắp xếp dãy A thành một dãy tăng dần và minh họa từng cách sắp xếp.
- Bài 2: Một đơn vị quan tâm về dân số lập một bảng thống kê số lượng người sinh trong từng năm, kể từ năm 1920 (X) đến 1970 (Y) và lưu trữ bảng đó trong máy tính bằng một mảng một chiều N với N[k] (k=0->(Y-X)) có giá trị bằng số người sinh ra tương ứng trong năm từ 1920 (X) đến 1970 (Y). Hãy viết giải thuật thực hiện
  - In ra những năm không có người nào được sinh ra
  - Tính số lượng những năm mà số người sinh ra không quá 10 (M)
  - Tính số người đã trên 50 (T) tuổi tính đến năm 1985 (Z).

Phương pháp tìm kiếm trực tiếp

- Tính trực tiếp địa chỉ của phần tử cần tìm từ giá trị của bản thân phần tử đó thông qua hàm băm
- Có thể dễ dàng tìm kiếm một cách trực tiếp và nhanh chóng một phần tử
- Việc xác định hàm băm thường không có phương pháp tổng quát
- Hiện tượng xung đột xảy ra khi hàm băm của hai phần tử cần tìm khác nhau lại có giá trị như nhau

Phương pháp tìm kiếm trực tiếp

- Hàm băm: (hash function) là một công thức cho phép tính một cách trực tiếp vị trí của phần tử cần tìm mà chỉ cần thông qua giá trị của nó. Như vậy, một hàm băm h là một ánh xạ có dạng như sau:

  $$h: D \rightarrow P;$$

  Trong đó: D là miền giá trị cho các phần tử cần tìm, P miền giá trị của các địa chỉ của các phần tử đó.
- Việc tìm một phần tử K cho trước chỉ đơn giản là tính h(K)
- Hiện tượng đụng độ: xảy ra khi giá trị hàm băm trên hai khóa tìm kiếm khác nhau lại bằng nhau, tức là: tồn tại $K1 \neq K2$ mà $h(K1) = h(K2)$.

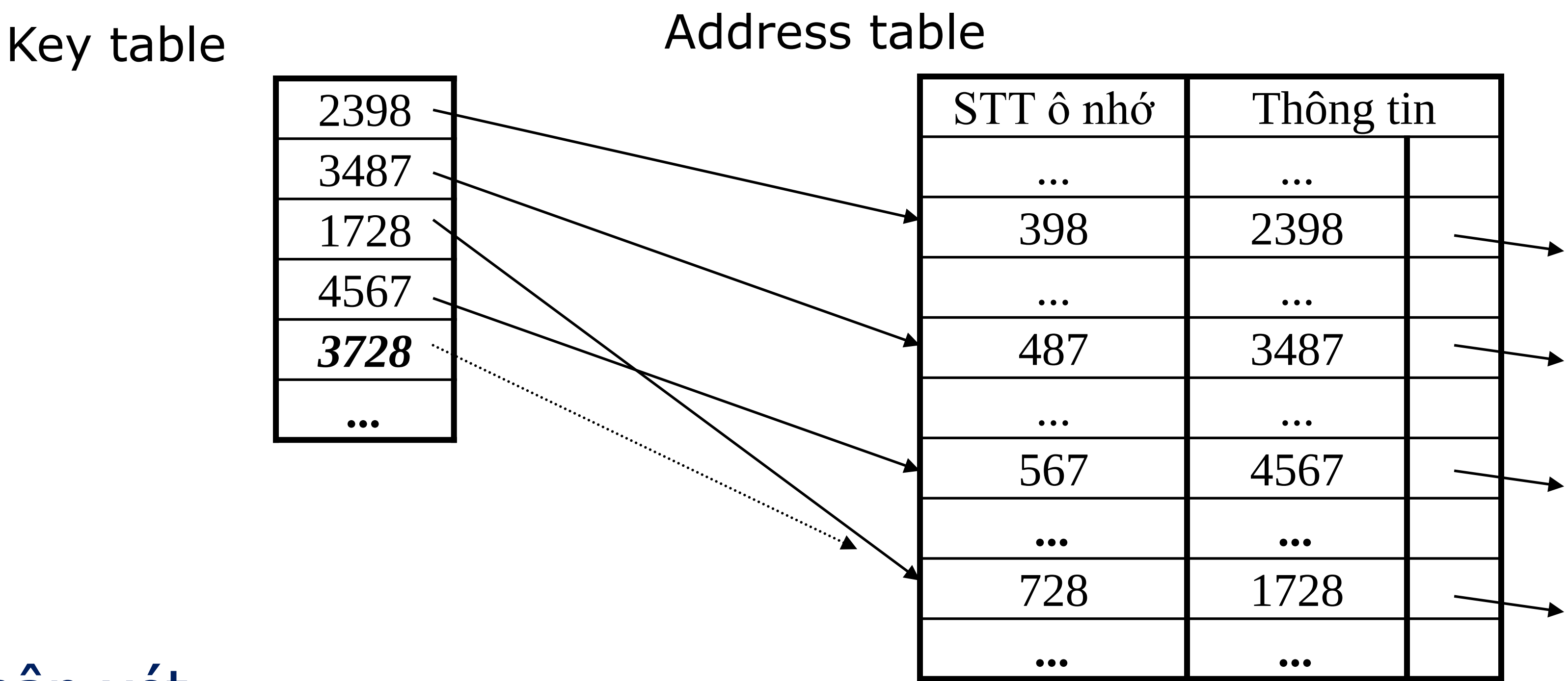
Phương pháp tìm kiếm trực tiếp

- Các tính chất của hàm băm:
  - Tính đầy đủ: là khi h(D)=P, tức là miền xác định của h sẽ bao phủ hết miền giá trị các địa chỉ mà các phần tử có thể được lưu trữ (ánh xạ h là một toàn ánh)

    => có thể tìm được bất kỳ phần tử nào dù cho nó nằm ở đâu trong miền lưu trữ.
  - Tính không đụng độ: tính không đụng độ sẽ đảm bảo ánh xạ h là một đơn ánh, và tính chất này sẽ giúp cho việc tìm kiếm là nhanh nhất

Phương pháp tìm kiếm trực tiếp

- Ví dụ minh họa: m=1000, h(k) = k mod m



- Nhận xét
  - Kích thước của bảng địa chỉ có giới hạn => hiện tượng đụng độ địa chỉ
  - Yêu cầu: xây dựng hàm địa chỉ cho các giá trị "rải" đều trên bảng và cần đưa ra được các biện pháp khắc phục đụng độ

Các giải thuật tìm kiếm trực tiếp

- Các vấn đề cần giải quyết
  - Xây dựng hàm địa chỉ ("hàm rải") cho tốt thông qua các phương pháp toán học
    - Phương pháp chia
    - Phương pháp nhân
    - Phương pháp phân đoạn
  - Chuẩn bị các biện pháp khắc phục đụng độ (băm lại - rehashing)
    - Biện pháp địa chỉ mở
    - Biện pháp móc nối

- Xây dựng hàm địa chỉ - Phương pháp chia
  - Phương pháp đơn giản và dễ sử dụng
    - h(k) = k mod m
  - Nhận xét:
    - m = 2 => h(k) có 2 giá trị
    - m = 1000 => h(k) chỉ phụ thuộc vào 3 chữ số cuối của k
    - Số giá trị của h(k) phụ thuộc vào m
    - Nếu m nguyên tố => rải tốt nhất
  - Yêu cầu: cần chọn m sao cho h(k) rải đều
  - Cải tiến:
    - h(k) = k mod m*, với m* là số nguyên tố lớn nhất < m

Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp nhân
  - Nguyên tắc
    - B1: lấy giá trị k*k
    - B2: xác định h(k) thông qua các chữ số liên tục ở giữa số k^2
  - Ví dụ:
    - m < 1000
    - tính k*k và chọn 3 chữ số ở giữa, không lấy 2 chữ số đầu, 2 chữ số cuối

k	k*k	h(k)
2398	5750404	504
3487	12159169	159 or 591
1728	2985984	859
4567	20857489	857 or 574
3728	13897984	897 or 979

Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp phân đoạn (partitioning)
  - Nguyên tắc:
    - Áp dụng khi khóa có kích thước lớn
    - Chia khóa thành các đoạn có độ dài như nhau = độ dài địa chỉ
    - Phối hợp các đoạn
      - Ví dụ: cộng lại, chọn một vài vị trí và ghép lại
  - Các kỹ thuật phân đoạn
    - Tách (spliting)
      - Tách tự phải qua trái
      - Xếp thành hàng các đoạn
    - Gấp (folding)
      - Gấp theo các đường biên (giống gấp giấy)
      - Xếp thành hàng các đoạn

Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp phân đoạn (partitioning)
  - Ví dụ:
    - k = 34289421
    - Độ dài địa chỉ: 3 (m<1000)
    - Tách: 421, 289, 034
    - Gấp: gấp theo biên 9, 4 và biên 2, 8 => 124, 289, 430

	Tách		**Gấp**
	421		124
	289		289
	034		430
h(k)	744	**h(k)**	843