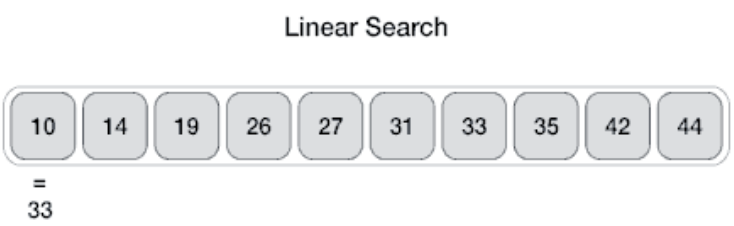
Tìm kiếm – Searching

- Tìm kiếm tuần tự - Sequential search
  - Còn gọi là tuyến tính – Linear search
  - Danh sách chưa sắp xếp hoặc đã sắp xếp
  - Thời gian tỉ lệ với n (số phần tử)
  - Độ phức tạp O(n)
- Tìm kiếm nhị phân – Binary search
  - Danh sách đã sắp xếp
  - Thời gian tỉ lệ với $\log_2 n$
  - Độ phức tạp O(log n)

Sequential search

- Duyệt danh sách từ đầu đến cuối
  - Dừng khi tìm thấy hoặc kết thúc danh sách
  - Nếu tìm thấy: Trả về kết quả tìm thấy
    - True hoặc vị trí được tìm thấy hoặc thông báo
  - Nếu không tìm thấy: Trả về kết quả không tìm thấy
    - False hoặc một giá trị như -1 hoặc thông báo



Sequential search – Vòng lặp

- Trả về vị trí khi tìm thấy
- Trả về -1 khi không tìm thấy
- Lưu ý: ***Các code chỉ mang tính minh hoạ cho giải thuật***
  - Có nhiều cách diễn đạt và cải tiến thuật toán

```
1. int linearSearch(int a[], int n, int x)
2. {
3.    int i;
4.    for(i=0; i<n; i++)
5.         if(a[i] == x)
6.              return i;
7.    return -1;
8. }
```


Sequential search – Vòng lặp

- Trả về kiểu bool
  - True: Tìm thấy
  - False: Không tìm thấy

```
1. bool linearSearch(int a[], int n, int x)
2. {
3.    int i;
4.    for(i=0; i<n; i++)
5.         if(a[i] == x)
6.              return true;
7.    return false;
8. }
```

Sequential search – Thông báo

- Xuất ra màn hình kết quả

```
1. void linearSearch(int a[], int n, int x)
2. {
3.    int i;
4.    for(i=0; i<n; i++)
5.      if(a[i] == x)
6.      {
7.         printf("Tim thay o vi tri %d", i);
8.         break;
9.      }
10.   if(i==n)
11.     printf("Khong tim thay");
12.}
```


Sequential search – Cờ hiệu

- Dùng cờ hiệu: Chương trình rõ ràng, dễ hiểu

```
1. void linearSearch(int a[], int n, int x)
2. {
3.    int i, flag = 0;     // Chưa tìm thấy
4.    for(i=0; i<n; i++)
5.      if(a[i] == x){
6.         printf("Tim thay o vi tri %d", i);
7.         flag = 1;        // Đã tìm thấy
8.         break;
9.      }
10.   if(!flag)
11.     printf("Khong tim thay");
12.}
```

Sequential search – Đệ quy

- Dùng đệ quy
  - Thực hiện gọi hàm nhiều lần

```
1. int linearSearch(int a[], int n, int x)
2. {
3.    if(n<0)
4.         return -1;
5.    else if(a[n-1] == x)
6.         return n-1;
7.    else
8.         return linearSearch(a, n-1, x);
9. }
```


Sequential search – Cầm canh

- Dùng phần tử cầm canh
  - Giảm bớt số lần so sánh

```
1. int linearSearch(int a[], int n, int x)
2. {
3.    int i = 0;
4.    a[n]=x;          // Phần tử cầm canh
5.    while(a[i]!=x)
6.         i++;
7.    if(i<n)
8.         return i;
9.    return -1;
10.}
```

Sequential search – Rút gọn

- Giảm thiểu số phép toán

```
1. int linearSearch(int a[], int n, int x)
2. {
3.    do{
4.         n--;
5.    }while(a[n]!=x && n>=0);
6.    return n;
7. }
```


Sequential search – Hai chiều

- Tìm cả hai chiều

```
1. int doubleSearch(int a[], int n, int x)
2. {
3.    int i=-1;
4.    do{
5.         if(a[--n]==x) return n;
6.         if(a[++i]==x) return i;
7.    }while(i<n);
8.    return -1;
9. }
```

So sánh thực nghiệm

- Thực hiện 1 triệu phép lặp cho mỗi hàm
  - Cơ bản
  - Đệ quy
  - Cầm canh
  - Rút gọn

```
1. for(int i=0;i<1000; i++)
2.   for(int j=0;j<1000; j++)
3.     k = linearSearch(a, n, x);
```

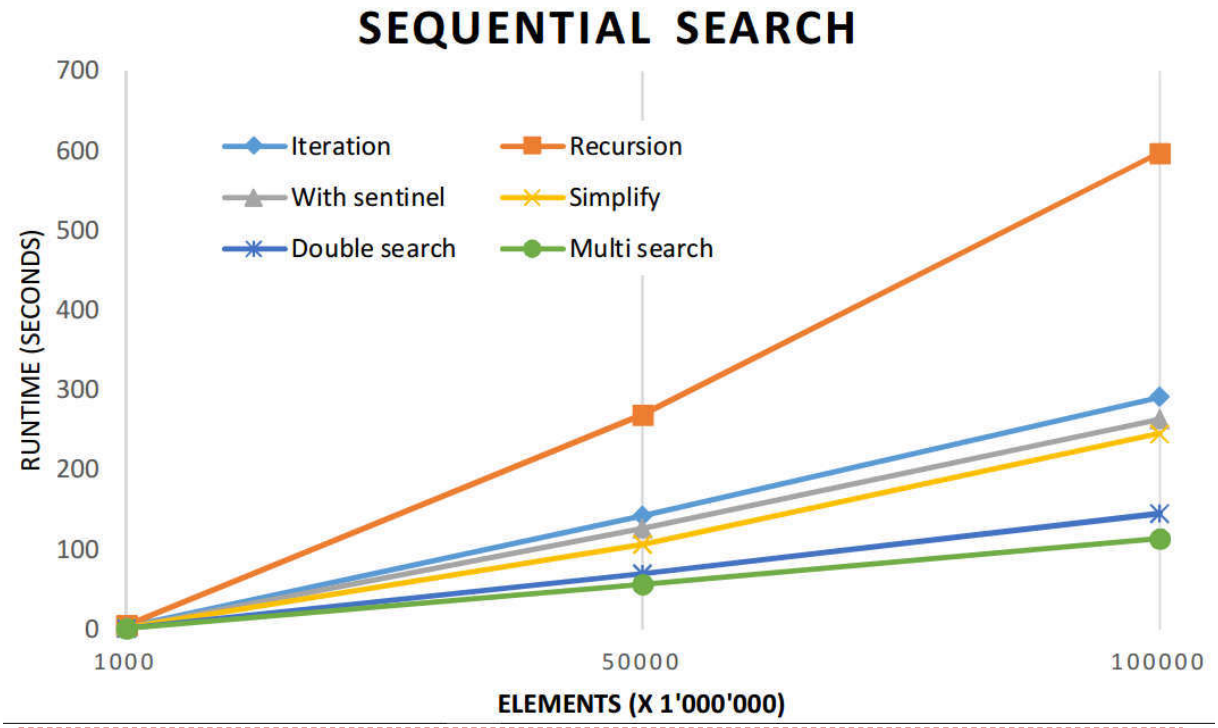
- Phần tử cần tìm nằm ở vị trí trong trường hợp xấu nhất (worst case)
- Đo thời gian thực hiện của mỗi hàm để so sánh kết quả

Cách đo thời gian

- Một số IDE có sẵn chức năng đo thời gian

```
1. #include <stdio.h>
2. #include <time.h>
3. int main()
4. {
5.     clock_t t = clock();

6.     // Đoạn code cần đo thời gian

7.     t = clock()-t;
8.     printf("Time: %.2fs\n",(float)t/CLOCKS_PER_SEC);
9.     return 0;
10.}
```

Đánh giá



Bài tập vận dụng

- Viết hàm tìm kiếm phần tử x trong khoảng từ *left* đến *right* trong mảng.
- Nguyên mẫu hàm?
- Sử dụng hàm?

Binary search

- Danh sách đã được sắp xếp, giả sử tăng dần
  - So sánh X với phần tử ở giữa danh sách
    - Nếu bằng nhau: Tìm kiếm thành công
    - Nếu X nhỏ hơn: Tiếp tục tìm bên trái danh sách
    - Nếu X lớn hơn: Tiếp tục tìm bên phải danh sách

www.penjee.com

Binary search – Iteration

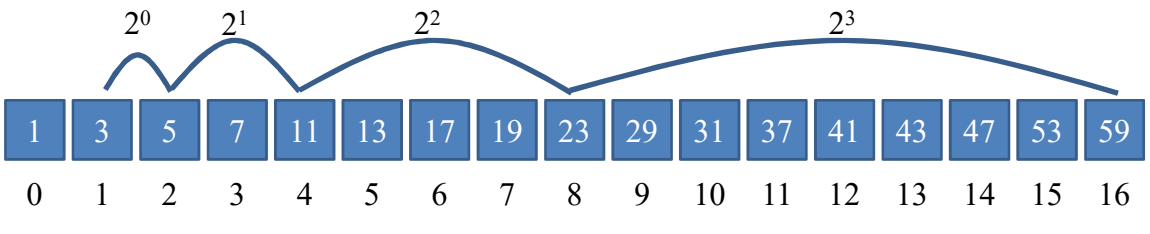
```
1. int binarySearch(int a[],int left,int right,int x)
2. {
3.    while(left<=right)
4.    {
5.          int mid = (left + right)/2;
6.          if(x==a[mid])
7.                return mid;
8.          if(x<a[mid])
9.                right = mid-1;
10.         else
11.               left = mid+1;
12.   }
13.   return -1;
14.}
```

Binary search – Recursion

```
1. int binarySearch(int a[],int left,int right,int x)
2. {
3.    if (left <= right)
4.    {
5.       int mid = left + (right - left)/2;      //?
6.       if (x == a[mid])
7.          return mid;
8.       if (x < a[mid])
9.          return binarySearch(a, left, mid-1, x);
10.      else
11.         return binarySearch(a, mid+1, right, x);
12.   }
13.   return -1;
14.}
```


Exponential Search

- Bao gồm hai bước
  - Xác định vùng chứa X trong mảng
    - Lần lượt so sánh X với các phần tử i bắt đầu từ 1, 2, 4, 8, 16… tăng dần theo lũy thừa 2.
    - Khi tìm được vị trí của phần tử i có giá trị lớn hơn X, vùng cần tìm là từ i/2 đến min(i,n)
  - Dùng binary search để tìm trong vùng đã xác định



Exponential Search

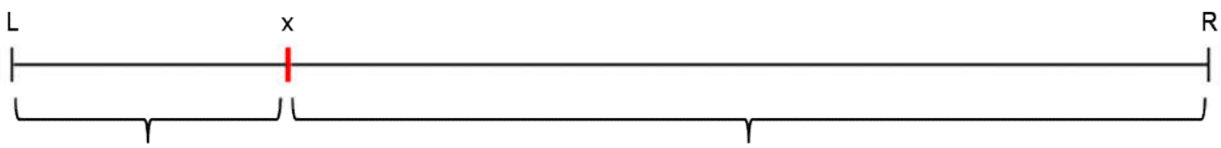
```
1. int exponentialSearch(int a[], int n, int x)
2. {
3.     if (a[0] == x)
4.          return 0;
5.
6.     int i=1;
7.     while (i < n && a[i] <= x)
8.           i = i*2;

9.     return binarySearch(a, i/2, (i<n)?i:n, x);
10.}
```


Interpolation Search

- Điểm mid không nhất thiết chính giữa
- Cách tính điểm ở giữa

```
mid = low+(x-a[low])*(high-low)/(a[high]-a[low]);
```

- mid sẽ gần điểm low khi x gần a[low] hơn
- mid sẽ gần điểm high khi x gần a[high] hơn

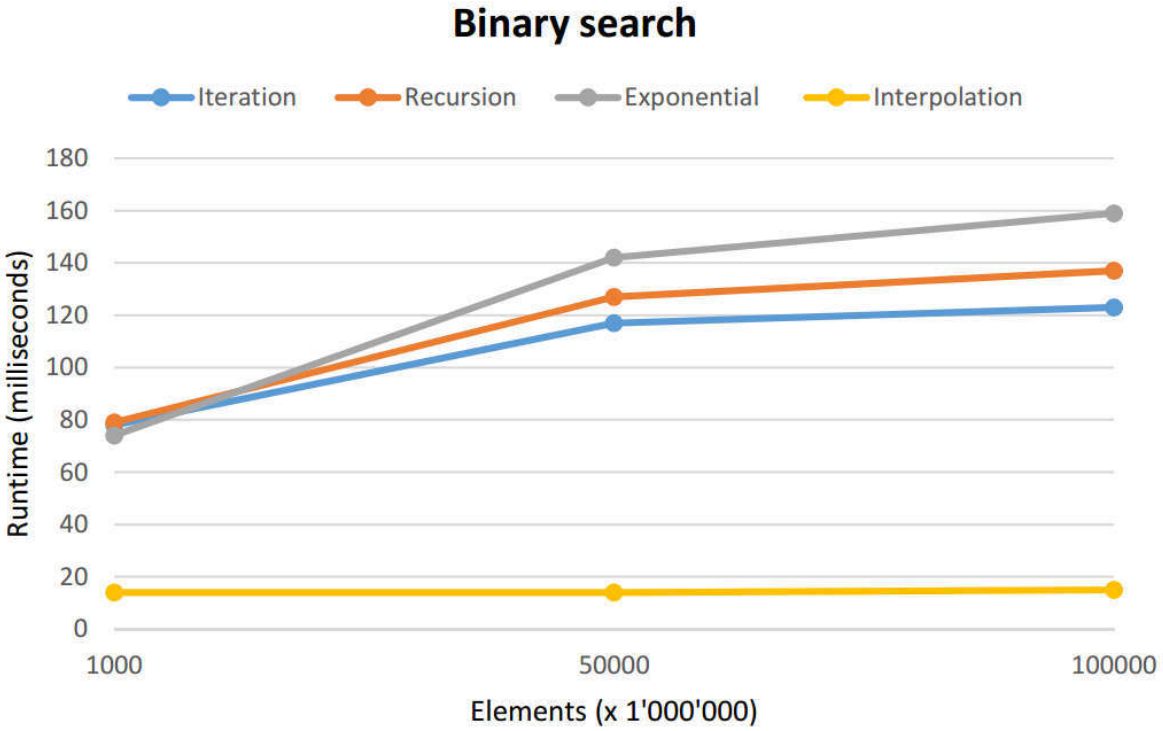


C = (x - L)/(R-L)

Interpolation Search

```
1. int interpolationSearch(int a[], int size, int x)
2. {
3.   int low = 0, high = size - 1, mid;
4.   while(high>=low && x>=a[low] && x<=a[high])
5.   {
6.     mid = low+(x-a[low])*(high-low)/(a[high]-a[low]);
7.     if (a[mid] < x)
8.        low = mid + 1;
9.     else if (x < a[mid])
10.       high = mid - 1;
11.    else
12.       return mid;
13.  }
14.  return -1;
15.}
```


Một kết quả so sánh



Độ phức tạp?

- Một trường hợp so sánh không đánh giá đầy đủ về các thuật toán
  - Cần nhiều trường hợp hơn

Algorithm	Best case	Average case	Worst case
Linear search	O(1)	O(n)	O(n)
Binary search	O(1)	O(log n)	O(log n)
Exponential Search	O(1)	O(log i)	O(log i) Với i là vị trí cần tìm
Interpolation Search	O(1)	O(log(log n))	O(n)

Bài tập vận dụng

- Viết chương trình
  - Phát sinh ngẫu nhiên một mảng tăng dần
  - Cài đặt các hàm tìm kiếm
  - Tìm giá trị x nhập từ bàn phím
  - Xuất ra số lần so sánh của mỗi phương pháp
  - Đánh giá các phương pháp
- Tìm hiểu hoặc đề xuất phương pháp mới