



# KỸ THUẬT XỬ LÝ LẶP

NHẬP MÔN LẬP TRÌNH

GVHD: Trương Toàn Thịnh

# NỘI DUNG

- Các thuật toán lặp cơ bản
- Điều kiện dừng vòng lặp
- Bài toán số nguyên tố
- Bài toán tính lũy thừa nhanh
- Bài toán phân tích ra thừa số nguyên tố
- Kỹ thuật tính toán lặp
- Xử lý lặp trên mảng
- Kỹ thuật đệ quy

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Thuật toán tính tổng/tích
  - Biến tổng có giá trị ban đầu là 0
  - Biến tích có giá trị ban đầu là 1
  - Lần lượt duyệt các phần tử trong tập hợp
  - Tích lũy vào biến tổng/tích

Tính tổng	Tính tích
$S \leftarrow 0$	$P \leftarrow 1$
<b>Duyệt</b> tập hợp	<b>Duyệt</b> tập hợp
$S \leftarrow S + x$	$P \leftarrow P \times x$
<b>Cuối</b> <b>duyệt</b>	<b>Cuối</b> <b>duyệt</b>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Cho dãy số 1, 2, ... n. Hãy viết hàm tính tổng và tích

Dòng	Cách 01	Cách 02
1	<code>void TinhTongTich(long n, long&amp; s, long&amp; p){</code>	<code>void TinhTongTich(long n, long&amp; s, long&amp; p){</code>
2	<code>s = 0, p = 1;</code>	<code>s = 0, p = 1;</code>
3	<code>for(int i = 1; i &lt; n; i++){</code>	<code>for(; n &gt; 0; n--){</code>
4	<code>s+=i;</code>	<code>s+=n;</code>
5	<code>p*=i;</code>	<code>p*=n;</code>
6	<code>}</code>	<code>}</code>
7	<code>}</code>	<code>}</code>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Cho số nguyên dương n, hãy tính:
  - Tổng s gồm các số có dạng  $m/(1 + m^2)$  với m chia 4 dư 1 và  $< n$
  - Tích p gồm các số m lẻ, chia 3 dư 1 và  $< n$

Dòng	Mô tả
1	<code>void main(){</code>
2	<code>long m, n; double s = 0, p = 1;</code>
3	<code>scanf("%d", &amp;n); m = n;</code>
4	<code>for(; m &gt; 0; m--) {</code>
5	<code>    if(m % 4 == 1) s+=m/(1.0+m*m);</code>
6	<code>    if(m % 2 == 1 &amp;&amp; m% 3 == 1) p*=m;</code>
7	<code>}</code>
8	<code>printf("%lf\n", s); printf("%lf\n", p);</code>
9	<code>}</code>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tính

- Nhận xét:

- $x^2 = x \times x, x^3 = x \times x^2, \dots, x^n = x \times x^{n-1}$ .

Dòng	Mô tả	
1	<code>double TinhTong(double x, int n){</code>	<code>void main(){</code>
2	<code>double s = 0, p = 1;</code>	<code>double x, s = 0; int n;</code>
3	<code>for(int i = 1; i &lt;= n; i++){</code>	<code>scanf("%lf", &amp;x); scanf("%d", &amp;n);</code>
4	<code>p*=x;</code>	<code>s = TinhTong(x, n);</code>
5	<code>s+=(p/i);</code>	<code>printf("%lf", s);</code>
6	<code>}</code>	<code>}</code>
7	<code>return s;</code>	
8	<code>}</code>	

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tính +

- Cách 1: Thay đổi dòng mã số 2 ( $p = -1$ ) & truyền  $-x$  trong main

Dòng	Mô tả	
1	<code>double TinhTong(double x, int n){</code>	<code>void main(){</code>
2	<code>double s = 0, p = -1;</code>	<code>double x, s = 0; int n;</code>
3	<code>for(int i = 1; i &lt;= n; i++) {</code>	<code>scanf("%lf", &amp;x); scanf("%d", &amp;n);</code>
4	<code>p*=x;</code>	<code>s = TinhTong(-x, n);</code>
5	<code>s+=(p/i);</code>	<code>printf("%lf", s);</code>
6	<code>}</code>	<code>}</code>
7	<code>return s;</code>	
8	<code>}</code>	

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tính  $\frac{1}{1+x}$ 
  - Cách 2: Chỉ thay đổi cách gọi trong hàm main

Dòng	Mô tả	
1	double TinhTong(double x, int n){	void main(){
2	double s = 0, p = 1;	double x, s = 0; int n;
3	for(int i = 1; i <= n; i++){	scanf("%lf", &x); scanf("%d", &n);
4	p*=x;	s = -TinhTong(-x, n);
5	s+=(p/i);	printf("%lf", s);
6	}	}
7	return s;	
8	}	

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tính
  - Có thể viết lại  $K_n(x)$  như sau
    - Vì vậy ta có thể thay  $x$  bằng  $(-x)$  để đơn giản hóa việc xử lý âm dương
    - Ta nhận thấy
      - Vì vậy việc tính thành phần thứ  $i$ , ta chỉ cần lấy kết quả trước đó  $\times$

# CÁC THUẬT TOÁN LẶP CƠ BẢN

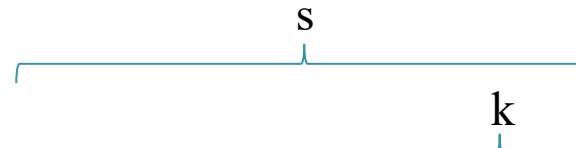
## • Tính

Dòng	Mô tả
1	<code>double TinhK(double x, int n){</code>
2	<code>double s = 0, p = 1;</code>
3	<code>for(int i = 1; i &lt;= n; i++){</code>
4	<code>p*=(x/i); s+=p;</code>
5	<code>}</code>
6	<code>return s;}</code>
7	<code>void main(){</code>
8	<code>double x, s, n;</code>
9	<code>scanf("%f", &amp;x); scanf("%d", &amp;n);</code>
10	<code>s = TinhK(-x, n);</code>
11	<code>}</code>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tính

- Công thức:  $1 + 3 + \dots + (2t - 1) = t^2$ .
- Cần tìm  $t$  sao cho  $t^2 \leq n$



Dòng	Mô tả
1	<code>double sqrtInt(long n){</code>
2	<code>long s = 1, k = 1;</code>
3	<code>while(s &lt; n){</code>
4	<code>    k += 2; s += k;</code>
5	<code>}</code>
6	<code>    if(s &gt; n) k = k - 2;</code>
7	<code>    return (k + 1)/2;</code>
8	<code>}</code>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm tất cả các số hoàn chỉnh nhỏ hơn m cho trước
  - Ví dụ 6 là số hoàn chỉnh vì:  $6 = 1 + 2 + 3$ , với  $\{1, 2, 3\}$  đều là ước số của 6
  - Ví dụ 28 là số hoàn chỉnh vì:  $28 = 1 + 2 + 4 + 7 + 14$ , với  $\{1, 2, 4, 14\}$  đều là ước số của 28
- Nhận xét thêm:
  - $6 = 2 \times 3 = 2^1 \times (2^2 - 1)$
  - $28 = 4 \times 7 = 2^2 \times (2^3 - 1)$
  - Vậy tất cả các số hoàn chỉnh chẵn đều có dạng  $2^{k-1} \times (2^k - 1)$

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm tất cả các số hoàn chỉnh nhỏ hơn m cho trước

Dòng	Mô tả	12	void main(){
1	long sumDivisors(long n){	13	long m;
2	long s = 0, k = n;	14	scanf("%ld", &m){
3	while(--k){	15	while(m--){
4	if(n % k) s+=k;	16	if(isPerfect(m))
5	}	17	printf("%ld\n", m)
6	return s;	18	}
7	}	19	}
8	int isPerfect(long n){		
9	if(n == sumDivisors(n)) return 1;		
10	return 0;		
11	}		

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Thuật toán đếm

- Khởi tạo biến ‘dem’ có giá trị ban đầu là 0
- Lần lượt duyệt các phần tử trong tập hợp
- Nếu phần tử thỏa điều kiện thì tăng biến ‘dem’ lên một đơn vị

Thuật toán đếm
Count ← 0
<b>Duyệt</b> trên tập hợp cần đếm
Lấy phần tử x hiện hành
Nếu x thỏa <b>điều kiện</b> thì
Count ← Count + 1

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Đếm số lượng ước số của số nguyên  $n > 0$

Dòng	Mô tả
1	<code>int DemUocSo(long n){</code>
2	<code>long count = 0;</code>
3	<code>for(int i = 1; i &lt;= n; i++) {</code>
4	<code>    if(n % i == 0)</code>
5	<code>        count++;</code>
6	<code>    }</code>
7	<code>}</code>

- Có thể tận dụng hàm này để kiểm tra xem một số nguyên  $n$  có phải là số nguyên tố hay không?

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Đếm số lượng các kí số, kí tự trong một chuỗi cho trước

Dòng	Mô tả	
1	void DemKiSoKiTu(char* s, int& c, int& n){	void main(){
2	c = n = 0;	char s[] = "Hello world";
3	for(int i = 0; i < strlen(s); i++){	int n, c;
4	if('a' <= s[i] && s[i] <= 'z')	DemKiSoKiTu(s, c, n);
5	c++;	printf("%d %d", n, c);
6	else if('A' <= s[i] && s[i] <= 'Z')	}
7	c++;	
8	else n++;	
9	}	
10	}	

H e l l o   w o r l d 0  
<10>  
<???>  
<10> S

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Đếm số lượng số âm, số dương và số không trong mảng các số thực

Dòng	Mô tả	
1	void DemSo(float a[], int n, int* nd, int* na, int* no){	void main(){
2	*nd = *na = *no = 0;	float b[] = {-2, 1, 4, -9, 0};
3	for(int i = 0; i < n; i++){	int nn = 5, dn, an, on;
4	if(a[i] > 0)	DemSo(s, c, n);
5	(*nd)++;	printf("%d %d %d", dn, an, on);
6	else if(a[i] < 0)	}
7	(*na)++;	
8	else (*no)++;	
9	}	
10	}	

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Cho mảng số thực  $a$  và biến số thực  $x$ . Hãy cho biết
  - $x$  có xuất hiện trong mảng  $a$  hay không?
  - $x$  có lớn hơn mọi số của mảng  $a$  hay không?
  - $x$  có nhỏ hơn một số nào đó trong mảng  $a$  hay không?
  - Tất cả các số trong mảng  $a$  có âm hay không?
- Dùng phương pháp đếm để xử lý
  - Đếm các số của mảng  $a$  trùng với  $x$ , kiểm tra kết quả đếm là dương hay âm
  - Đếm các số của mảng  $a$  nhỏ hơn  $x$ , kiểm tra kết quả đếm có bằng  $n$  hay không
  - Đếm các số của mảng  $a$  lớn hơn  $x$ , kiểm tra kết quả đếm có dương hay không

# CÁC THUẬT TOÁN LẮP CƠ BẢN

- Cho mảng số thực a và biến số thực x.

1	int Appear(float a[], int n, float x){	10	int GreaterAll(float a[], int n, float x){
2	int cnt = 0;	11	int cnt = 0, m = n;
3	while(n--){	12	while(m--){
4	if(x == a[n]) cnt++;	13	if(x > a[m]) cnt++;
5	}	14	}
6	return (cnt > 0); }	15	return (cnt == n); }
7	int AllNegative(float a[], int n){	16	int LessOne(float a[], int n, float x){
8	return GreaterAll(a, n, 0);	17	int cnt = 0;
9	}	18	while(n--){
		19	if(x < a[n]) cnt++;
		20	}
		21	return (cnt > 0); }

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Cho mảng thực a. Cần trả lời các câu sau
  - Mảng a có thứ tự tăng/giảm hay không
  - Mảng a có sắp thứ tự hay không
- Ý tưởng giải:
  - Theo đ/n mảng tăng:  $a[0] \leq a[1] \leq \dots \leq a[n - 1]$ .
  - Vậy ta có:  $a[i - 1] \leq a[i], \forall i \in [1, \dots, n - 1]$
  - Vậy ta có  $n - 1$  chỉ số i thỏa  $a[i - 1] \leq a[i]$
- Đoạn mã
  - `int isIncreasing(float a[], int n){  
 int cnt = 0, m = n;  
 while(--m){  
 if(a[m - 1] <= a[m]) cnt++;  
 }  
 return (cnt == n - 1);  
}`

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Thuật toán tìm phần tử nhỏ/lớn nhất
  - Ban đầu chọn một phần tử  $x_0$  nào đó trong tập hợp làm **kết quả mong đợi**.
  - Duyệt tập hợp  $\setminus \{x_0\}$  và nếu thấy có phần tử thỏa mãn hơn **kết quả mong đợi** thì cập nhật

**Thuật toán tìm phần tử mong đợi**

Trường hợp không có điều kiện K	Trường hợp có điều kiện K
<b>Chọn</b> $x_0 \in S$ và cho $kq \leftarrow x_0$	<b>Flag</b> $\leftarrow$ false
<b>Duyệt</b> $x \in S \setminus \{x_0\}$	<b>Duyệt</b> $x \in S$
<b>Nếu</b> $x$ thỏa điều kiện với $kq$ thì	<b>Nếu</b> $x$ thỏa điều kiện K
$kq \leftarrow x;$	<b>Nếu</b> not Flag thì
	$kq \leftarrow x; Flag \leftarrow true$
	<b>Ngược lại</b> nếu $x$ thỏa điều kiện với $kq$ thì $kq \leftarrow x$

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm phần tử nhỏ nhất trong mảng các số thực

Dòng	Mô tả	
1	<code>int PhanTuNhoNhat(float a[], int n){</code>	<code>int PhanTuNhoNhat(float a[], int n){</code>
2	<code>    int idx = 0;</code>	<code>    int idx = --n;</code>
3	<code>    for(int i = 0; i &lt; n; i++){</code>	<code>    while(n--){</code>
4	<code>        if(a[idx] &gt; a[i])</code>	<code>        if(a[idx] &gt; a[n])</code>
5	<code>            idx = i;</code>	<code>    idx = n;</code>
6	<code>}</code>	<code>}</code>
7	<code>    return idx;</code>	<code>    return idx;</code>
8	<code>}</code>	<code>}</code>

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm phần tử **DƯƠNG** nhỏ nhất trong mảng các số thực

Dòng	Phiên bản như thuật toán	Phiên bản cải tiến
1	int PhanTuDuongNhoNhat(float a[], int n){	int PhanTuDuongNhoNhat(float a[], int n){
2	int idx = -1; bool flag = false;	int idx = -1;
3	for(int i = 0; i < n; i++){	for(int i = 0; i < n; i++){
4	if(a[i] > 0){	if(a[i] > 0){
5	if(flag == false){	if(idx == -1    a[idx] > a[i]){
6	idx = i; flag = true;	idx = i;
7	}	}
8	else if(a[idx] > a[i]) idx = i;	
9	}	}
10	}	}
11	return idx;}	return idx;}

# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm phần tử  $\neq 0$  có trị tuyệt đối nhỏ nhất trong mảng các số thực

Dòng	Mô tả
1	<code>int absMinNotZero(float a[], int n){</code>
2	<code>    int idx = -1;</code>
3	<code>    for(int i = 0; i &lt; n; i++){</code>
4	<code>        if(a[i] != 0)</code>
5	<code>            if(idx == -1    fabs(a[idx]) &gt; fabs(a[i]))</code>
6	<code>                idx = i;</code>
7	<code>}</code>
8	<code>return idx;</code>
9	<code>}</code>

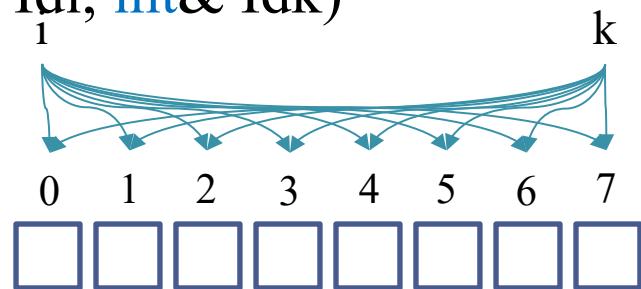
# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm cặp phần tử  $a[i]$  &  $a[j]$  sao cho  $a[i] \neq a[j]$  và  $|a[i] - a[j]|$  đạt giá trị nhỏ nhất

Dòng	Mô tả
1	<code>double minPair(double a[], int n, int&amp; idi, int&amp; idj){</code>
2	<code>    double dmin= -1, d; idi = idj = -1;</code>
3	<code>    for(int j = n - 1; j &gt;= 1; j--){</code>
4	<code>        for(int i = j - 1; i &gt;= 0; i--)</code>
5	<code>            if(a[i] != a[j]) {</code>
6	<code>                d = fabs(a[i] - a[j]);</code>
7	<code>            if(dmin == -1    dmin &gt; d){dmin = d; idi = i; idj = j;}</code>
8	<code>            }</code>
9	<code>    }</code>
10	<code>    return dmin;}</code>

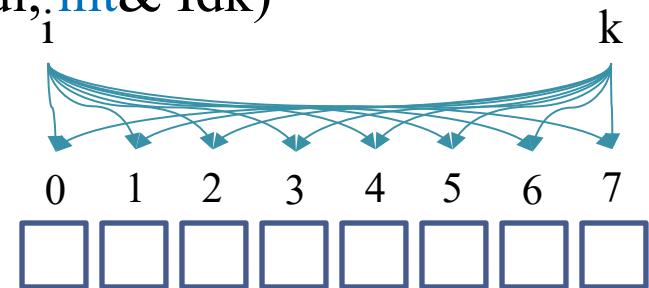
# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm cặp chỉ số  $(i, k)$  sao cho  $0 \leq i \leq k < n$  và  $a[i] + \dots + a[k]$  đạt giá trị lớn nhất
  - Đây là bài toán kết hợp tìm kiếm và tính tổng
  - Nếu mảng thực chỉ chứa toàn số không âm thì giá cặp chỉ số là  $(0, n - 1)$
- Đoạn mã
  - `float maxS(float a[], int n, int& Idi, int& Idk)`
    - `float lc = a[0]; Idi = Idk = 0;`
    - `for(int i = 0; i < n; i++)`
      - `for(int k = i; k < n; k++)`
        - `float s = 0;`
        - `for(int j = i; j <= k; j++) s+=a[j];`
        - `if(s > lc)`
          - `lc = s;`
          - `Idi = i; Idk = k;`
      - `return lc;`



# CÁC THUẬT TOÁN LẶP CƠ BẢN

- Tìm cặp chỉ số  $(i, k)$  sao cho  $0 \leq i \leq k < n$  và  $a[i] + \dots + a[k]$  đạt giá trị lớn nhất
  - Đây là bài toán kết hợp tìm kiếm và tính tổng
  - Nếu mảng thực chỉ chứa toàn số không âm thì giá cặp chỉ số là  $(0, n - 1)$
- Đoạn mã cải tiến
  - `float maxS(float a[], int n, int& Idi, int& Idk)`
    - `float lc = a[0]; Idi = Idk = 0;`
    - `for(int i = 0; i < n; i++)`
      - `float s = 0;`
      - `for(int k = i; k < n; k++)`
        - `s+=a[k];`
        - `if(s > lc)`
          - `lc = s;`
          - `Idi = i; Idk = k;`
      - `return lc;`



# NỘI DUNG

- Các thuật toán lặp cơ bản
- Điều kiện dừng vòng lặp
- Bài toán số nguyên tố
- Bài toán tính lũy thừa nhanh
- Bài toán phân tích ra thừa số nguyên tố
- Kỹ thuật tính toán lặp
- Xử lý lặp trên mảng
- Kỹ thuật đệ quy

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- ‘Bát biến’ của vòng lặp là một biểu thức hay một điều kiện logic
- ‘Bát biến’ vòng lặp luôn có giá trị đúng lúc đầu và sau khi vòng lặp kết thúc
- Ví dụ xét bài toán

Dòng	Mô tả
1	<code>double sum = 0;</code>
2	<code>for(int i = 1; i &lt;= n; i++){</code>
3	<code>    sum+=1.0/i;</code>
4	<code>}</code>

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ xét bài toán
  - Từ chương trình ta suy ra biểu thức
  - Vậy nếu kiểm tra  $i \in [1, n]$  thì ta thấy mệnh đề trên thỏa

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ xét bài toán tính  $x^n$ .

Dòng	Mô tả
1	<code>double y = 1;</code>
2	<code>for(int i = 0; i &lt; n; i++)</code>
3	<code>y*=x;</code>

- Từ chương trình ta suy ra biểu thức

$$0 \leq i < n \Rightarrow y = x^i$$

$$\Leftrightarrow \neg(0 \leq i < n) \vee y = x^i$$

$$\Leftrightarrow (i \geq 0) \vee (i \geq n) \vee y = x^i$$

- Vậy nếu kiểm tra  $i \in [0, n)$  thì ta thấy mệnh đề trên thỏa

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Mệnh đề lượng từ tồn tại ( $\exists$ ), với mọi ( $\forall$ )
  - Mệnh đề  $(\forall x \in S)(P(x))$  đúng khi đúng với tất cả các  $x$ .
  - Mệnh đề  $(\exists x \in S)(P(x))$  đúng khi tồn tại một  $x$  làm cho  $P(x)$  đúng.

Thuật toán kiểm tra	
Lượng từ với mọi	Lượng từ tồn tại
Kiểm tra: $(\forall x \in S)(P(x))$	Kiểm tra: $(\exists x \in S)(P(x))$
Flag $\leftarrow$ true	Flag $\leftarrow$ false
Duyệt $x \in S$	Duyệt $x \in S$
Nếu $P(x)$ sai	Nếu $P(x)$ đúng
Flag $\leftarrow$ false	Flag $\leftarrow$ true
Dừng duyệt	Dừng duyệt

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ: xác định xem x có lớn hơn **số nhỏ nhất** trong mảng n số thực hay không
- Bài toán  $\Leftrightarrow \exists i \in \{0, 1, \dots, n - 1\} \mid x > a[i]$

Dòng	Mô tả
1	<code>int greaterMin(double a[], int n, double x){</code>
2	<code>int kt = 0;</code>
3	<code>for(int i = 0; i &lt; n; i++){</code>
4	<code>if(x &gt; a[i]){</code>
5	<code>kt = 1;</code>
6	<code>break;</code>
7	<code>}</code>
8	<code>}</code>
9	<code>return kt;</code>

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ: xác định xem x có lớn hơn **số lớn nhất** trong mảng n số thực hay không
- Bài toán  $\Leftrightarrow \forall i \in \{0, \dots, n - 1\} \mid x > a[i]$

Dòng	Mô tả
1	<code>int greaterMax(double a[], int n, double x){</code>
2	<code>    int kt = 1;</code>
3	<code>    for(int i = 0; i &lt; n; i++){</code>
4	<code>        if(x &lt;= a[i]){</code>
5	<code>            kt = 0;</code>
6	<code>        break;</code>
7	<code>    }</code>
8	<code>}</code>
9	<code>return kt;}</code>

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ: xác định xem x tồn tại trong mảng n số hay không
- Bài toán  $\Leftrightarrow \exists i \in \{0, \dots, n - 1\} \mid x = a[i]$

Dòng	Mô tả
1	<code>int Find(double a[], int n, double x){</code>
2	<code>    int vt = -1;</code>
3	<code>    for(int i = 0; i &lt; n; i++){</code>
4	<code>        if(x == a[i]){</code>
5	<code>            vt = i;</code>
6	<code>        break;</code>
7	<code>}</code>
8	<code>}</code>
9	<code>return vt;</code>

# ĐIỀU KIỆN DÙNG VÒNG LẶP

- Ví dụ: xác định mảng thực a tăng?
- Bài toán  $\Leftrightarrow \forall i \in \{1, \dots, n - 1\} \mid a[i - 1] \leq a[i]$

Dòng	Mô tả
1	<code>int Increasing(double a[], int n){</code>
2	<code>    int KT = 1;</code>
3	<code>    for(int i = 1; i &lt; n; i++){</code>
4	<code>        if(a[i - 1] &gt; a[i]) {</code>
5	<code>            KT = 0;</code>
6	<code>        break;</code>
7	<code>}</code>
8	<code>}</code>
9	<code>    return KT;}</code>

# NỘI DUNG

- Các thuật toán lặp cơ bản
- Điều kiện dừng vòng lặp
- Bài toán số nguyên tố
- Bài toán tính lũy thừa nhanh
- Bài toán phân tích ra thừa số nguyên tố
- Kỹ thuật tính toán lặp
- Xử lý lặp trên mảng
- Kỹ thuật đệ quy

# BÀI TOÁN SỐ NGUYÊN TỐ

- Số nguyên tố là số có hai ước số.
- Số 1 không phải là số nguyên tố
- Số 0 là số nguyên tố đặc biệt
- Bài toán xác định số nguyên  $a > 1$  có phải là số nguyên tố hay không
  - $\Leftrightarrow$  Xác định  $a > 1$  có phải là hợp số hay không
  - $\Leftrightarrow \exists i \in \{2, \dots, a-1\}, i | a$

# BÀI TOÁN SỐ NGUYÊN TỐ

- Thuật toán xác định số nguyên tố (chậm)

Dòng	Mô tả
1	<code>int isPrime(long a){</code>
2	<code>int kq;</code>
3	<code>if(a &lt; 0) a = -a;</code>
4	<code>switch(a){</code>
5	<code>case 0: kq = 1; break;</code>
6	<code>case 1: kq = 0; break;</code>
7	<code>default:{</code>
8	<code>long i = 2; kq = 1</code>
9	<code>while(i &lt; a){</code>
10	<code>if(a % i == 0){kq = 0; break;}</code>
11	<code>i++;</code>
12	<code>}}}} return kq;}</code>

# BÀI TOÁN SỐ NGUYÊN TỐ

- Nếu  $m | a \Rightarrow \exists n | a$  và  $n \leq$
- Chỉ có 0 và 2 là số nguyên tố chẵn.

Dòng	Mô tả	Dòng	Mô tả
1	<code>int isPrime(long a){</code>	11	<code>while(i &lt;= sqrt(a)){</code>
2	<code>int kq;</code>	12	<code>if(a % i == 0){ kq = 0; break;}</code>
3	<code>if(a &lt; 0) a = -a;</code>	13	<code>i+=2;</code>
4	<code>switch(a){</code>	14	<code>}</code>
5	<code>case 1: kq = 0; break;</code>	15	<code>}</code>
6	<code>case 0: case 2: case 3: kq = 1; break;</code>	16	<code>}</code>
7	<code>default:{</code>	17	<code>return kq;</code>
8	<code>long i = 3;</code>	18	<code>}</code>
9	<code>if(a % 2 == 0){ kq = 0; break;}</code>		
10	<code>kq = 1;</code>		

# BÀI TOÁN SỐ NGUYÊN TỐ

## (Tìm ước số chung)

- Cho hai số **nguyên dương** a và b, thì d gọi là ước số chung lớn nhất của a và b khi:
  - d là ước của a và d là ước của b
  - Nếu tồn tại t là ước của a và t là ước của b thì t cũng phải là ước của d
- Phát biểu hình thức:  $\text{gcd}(a, b) = d \wedge \{\forall t: t \mid a \wedge t \mid b \Rightarrow t \mid d\}$
- Thuật toán Euclide tìm UCSLN của a và b
  - **while**( $a \neq 0 \wedge b \neq 0$ )
    - **if**  $a > b$  **then**  $a \leftarrow a \bmod b$
    - **else**  $b \leftarrow b \bmod a$
  - **return**  $a + b;$
- Hai số nguyên dương a và b gọi là nguyên tố cùng <sub>41</sub>

# BÀI TOÁN TÍNH LŨY THỪA NHANH

- Bài toán tính  $a^n$ .

- Có thể dùng  $\text{pow}(a, n)$
- Vòng lặp n lần phép tính nhân
- Cải tiến nhờ áp dụng tính chất sau:

nếu  $n$  chẵn

nếu  $n$  lẻ

- Từ vòng lặp  $n$ , ta chỉ cần lặp  $n/2$  hoặc  $(n - 1)/2$
- Dù  $n$  là chẵn hay lẻ thì vẫn cần bình phương  $a$
- Trường hợp  $n$  lẻ, cần nhân thêm đại lượng  $a$  (trước khi bình phương)

# BÀI TOÁN TÍNH LŨY THỪA NHANH

- Thuật toán tính lũy thừa

Dòng	Mô tả
1	<code>double power_n(double a, long n){</code>
2	<code>    double kq = 1;</code>
3	<code>    for(int i = 0; i &lt; n; i++)</code>
4	<code>    {</code>
5	<code>        kq*=a;</code>
6	<code>    }</code>
7	<code>    return kq;</code>
8	<code>}</code>

Dòng	Mô tả
1	<code>double power_n(double a, long n){</code>
2	<code>    double kq = 1;</code>
3	<code>    for(; n &gt; 0; n/=2){</code>
4	<code>        if(n % 2 == 1) kq*=a;</code>
5	<code>        a*=a;</code>
6	<code>    }</code>
7	<code>    return kq;</code>
8	<code>}</code>

- Nếu  $n \approx 2$  tỷ thì thuật toán phía trái chạy khá chậm

# BÀI TOÁN TÍNH LŨY THỪA NHANH

- Ví dụ chạy thuật toán tính lũy thừa nhanh

Dòng	Mô tả	
1	double power_n(double a, long n){	<u>Ví dụ a = 3 và n = 12</u>
2	double kq = 1;	Lần 0: kq = 1 & a = 3 & n = 12
3	for(; n > 0; n/=2){	Lần 1: kq = 1 & a = 9 & n = 6
4	if(n % 2 == 1) kq*=a;	Lần 2: kq = 1 & a = 81 & n = 3
5	a*=a;	Lần 3: kq = 81 & a = 6561 & n = 1
6	}	Lần 4: kq = 81*6561 & a = 43046721 & n = 0
7	return kq;	
8	}	

Cuối cùng kq = 81\*6561 = 531441

# BÀI TOÁN TÍNH LŨY THỪA NHANH

- Bài toán  $a^n \bmod b = r = a^n - k \times b$  ( $k \in \mathbb{Z}$ )
  - Không thể tính  $a^n$  trước sau đó mod b (vì  $a^n$  có thể rất lớn)
  - Có thể tính nhờ nhận xét:  $a^n \bmod b = [(a^{n-1} \bmod b) \times a] \bmod b$ .
  - Có ngay thuật toán (C/C++)
    - long kq = 1;
    - while(n--){ // lặp n lần (kiểm tra trước sau đó giảm)
      - kq = kq\*a % b;
      - }
- Cải tiến từ nhận xét:

$$a^n \bmod b = \begin{cases} \text{ nếu } n \text{ chẵn} \\ \text{ nếu } n \text{ lẻ} \end{cases}$$

# BÀI TOÁN TÍNH LŨY THỪA NHANH

- Ví dụ chạy thuật toán  $a^n \bmod b$  nhanh

Dòng	Mô tả
1	long modPow(long a, long b, long n){
2	long kq = 1;
3	for(; n > 0; n/=2){
4	if(n % 2 == 1) kq = kq*a % b;
5	a = a*a % b;
6	}
7	return kq;
8	}

Ví dụ  $a = 3, n = 5$  và  $b = 12$

Lần 0:  $kq = 1$  &  $a = 3$  &  $n = 5$

Lần 1:  $kq = 3$  &  $a = 4$  &  $n = 2$

Lần 2:  $kq = 3$  &  $a = 1$  &  $n = 1$

Lần 3:  $kq = 3$  &  $a = 1$  &  $n = 0$

Cuối cùng  $kq = 3$

# BÀI TOÁN PHÂN TÍCH THÙA SỐ NGUYÊN TỐ

- Luôn phân tích được
- Trong đó các  $p_i$  là các số nguyên tố và  $e_j$  là các số tự nhiên.  
Ví dụ:  $315 = 3^2 \times 5 \times 7$
- Thuật toán (ví dụ chỉ xét  $n \geq 4$ )
  - `void factors(long n){`
    - `long p = 2, c = 0;`
    - `while(n > 1) {`
      - `if(n % p == 0) {`
        - `c++; n = n/p;`
        - `}`
        - `else{`
          - `if(c > 0) printf("%ld ^ %ld x ", p, c);`
          - `p++; c = 0;`
          - `}`
        - `}`
        - `if(c > 0) printf("%ld ^ %ld", p, c);`
    - `}`

$p = 3$        $c = 0$

$n = 315$

$3^2 \times 5 \times 7^1$

# BÀI TOÁN PHÂN TÍCH THÙA SỐ NGUYÊN TỐ

- Luôn phân tích được
- Cần mảng pr[] để lưu các  $p_i$  và e[] để lưu các  $e_j$
- Thuật toán phiên bản lưu trữ (ví dụ chỉ xét  $n \geq 4$ )
  - void factors(**long** n, **long** pr[], **long** e[], **long&** np){
    - **long** p = 2, c = 0, np = 0;
    - **while**(n > 1) {
      - **if**(n % p == 0) {
        - **if**(c == 0){ pr[np] = p; np++; }
        - c++;
        - e[np - 1] = c;
        - n /= p;
      - }
        - **else** {p++; c = 0;}
      - }
        - }

e      2      1      1

pr    3      5      7

np = 0

n = 35

p = 2

c = 0

# BÀI TOÁN PHÂN TÍCH THÙA SỐ NGUYÊN TỐ

- Luôn phân tích được
- Ta luôn thấy  $p_k \leq$
- Thuật toán cải tiến (ví dụ chỉ xét  $n \geq 4$ )
  - `void factors(long n, long pr[], long e[], long& np){`
    - `long p = 2, c = 0, np = 0;`
    - `double sqrtN = sqrt(n); // vd: n = 51 và sqrtN = 7.14...`
    - `while(n > 1) {`
    - `if(n % p == 0) {`
    - `if(c == 0){ pr[np] = p; np++; }`
    - `c++;`
    - `e[np - 1] = c;`
    - `n /= p;`
    - `}`
    - `else {`
    - `p++; c = 0;`
    - `if(p > sqrtN){`
    - `pr[np] = n; e[np] = 1; np++; break;`
    - `}`
    - `}`
    - `}`
    - `}`

e 

1	1
---	---

pr 

3	17
---	----

np =  $\emptyset$

n = 51

p = 2

c =  $\emptyset$

# NỘI DUNG

- Các thuật toán lặp cơ bản
- Điều kiện dừng vòng lặp
- Bài toán số nguyên tố
- Bài toán tính lũy thừa nhanh
- Bài toán phân tích ra thừa số nguyên tố
- Kỹ thuật tính toán lặp
- Xử lý lặp trên mảng
- Kỹ thuật đệ quy

# KỸ THUẬT TÍNH TOÁN LẶP

- Xét bài toán dãy Fibo

$$\begin{cases} F_0 = F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad (n \geq 2) \end{cases}$$

- Thuật toán

- Fib(int n){
  - if  $n = 0$  or  $n = 1$  then return 1
  - while( $k < n$ )
    - $U \leftarrow F // F \Leftrightarrow F_n$
    - $F \leftarrow lastF + F // LastF \Leftrightarrow F_{n-1}$
    - $lastF \leftarrow U$
    - $k \leftarrow k + 1;$
  - }

Bỏ biến U

$$\begin{cases} F \leftarrow F + lastF \\ lastF \leftarrow F - lastF \end{cases}$$

# KỸ THUẬT TÍNH TOÁN LẶP

- Xét phương trình:  $x^3 + ax^2 + bx + c = 0$ . Hãy tính  $S_n = x^n + y^n + z^n$ .
- Dựa vào tính chất phương trình ta có hệ

$$\begin{aligned} S_0 &= 3 \\ S_1 &= x + y + z = -a \\ S_2 &= \cancel{x^2} + y^2 + z^2 = a^2 - 2b \\ S_n &= -aS_{n-1} - bS_{n-2} - cS_{n-3} \quad (n \geq 3) \end{aligned}$$

- Thuật toán
  - **TinhS(int n){**
    - $lastS \leftarrow 3, S \leftarrow -a, nextS \leftarrow a*a - 2*b$
    - **while**( $k < n$ )
      - $U \leftarrow nextS // nextS \Leftrightarrow S_{n+1}$
      - $nextS \leftarrow -a*nextS - b*S - c*lastS // lastS \Leftrightarrow S_{n-1}$
      - $lastS \leftarrow S$
      - $S \leftarrow U // S \Leftrightarrow S_n$
      - $k \leftarrow k + 1;$
  - **}**

# KỸ THUẬT TÍNH TOÁN LẶP

- Tính xấp xỉ  $\sin(x)$  và  $\cos(x)$ . Ta có ngay công thức
  - $\sin(x) = 2\sin(x/2)\cos(x/2)$  &  $\cos(x) = 1 - 2\sin^2(x/2)$
  - Tổng quát hóa ta có:
    - $\sin(x/2^{n-1}) = 2\sin(x/2^n)\cos(x/2^n)$  &  $\cos(x/2^{n-1}) = 1 - 2\sin^2(x/2^n)$
    - Nếu  $x \in [0, 2]$  và  $n \geq 20$  thì ta có
      - $\sin(x/2^n) \approx (x/2^n) - (x/2^n)^3/6$  &  $\cos(x/2^n) \approx 1 - (x/2^n)^2/2$
    - Thiết kế thuật toán:
      - `TinhSinCos(double x, double& sinx, double& cosx)`
        - Đặt hằng số n từ 20 trở lên ( $n = 20$ )
        - Tính  $x \leftarrow x - 2$  ( $0 \leq x < 2$ )
        - Gán  $x \leftarrow x/2^n$  và tính  $\sinx \leftarrow x - x^3/6$  &  $\cosx \leftarrow 1 - x^2/2$
        - `while(n ≠ 0)`
          - $sx \leftarrow 2\sinx\cosx$ ,  $cx \leftarrow 1 - 2\sinx\sinx$
          - $\sinx \leftarrow sx$ ,  $\cosx \leftarrow cx$
          - $n \leftarrow n - 1$

# KỸ THUẬT TÍNH TOÁN LẶP

- Tính căn bậc hai số thực a ( $a \in \mathbb{R}$ )
  - $x_0 = 1$  &  $x_n = x_{n-1}/2 + a/(2 \times x_{n-1})$  ( $n \geq 1$ )
  - Nhận xét:
    - Để tính thành phần n ta cần tính thành phần n – 1
    - Khi  $n \rightarrow +\infty$  thì  $|x_n - x_{n-1}| \rightarrow 0$
  - Thiết kế thuật toán:
    - `#define epsilon 0.0000001`
    - `double sqrtR(double a)`
      - `if a <= 0 then return 0`
      - `double xLast, x = 1`
      - `do {`
        - `xLast ← x`
        - `x ← xLast/2 + a/(2*xLast)`
      - `}while(|x - xLast| >= epsilon)`
      - `return x`

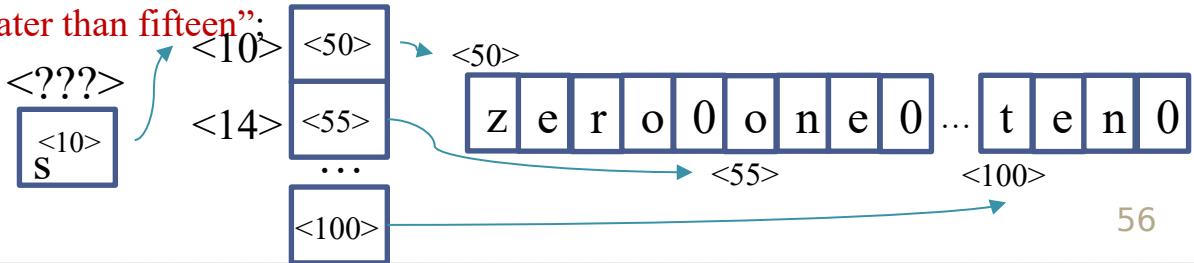
# NỘI DUNG

- Các thuật toán lặp cơ bản
- Điều kiện dừng vòng lặp
- Bài toán số nguyên tố
- Bài toán tính lũy thừa nhanh
- Bài toán phân tích ra thừa số nguyên tố
- Kỹ thuật tính toán lặp
- Xử lý lặp trên mảng
- Kỹ thuật đệ quy

# XỬ LÝ LẬP TRÊN MẢNG

## (Dùng làm bảng tra cứu)

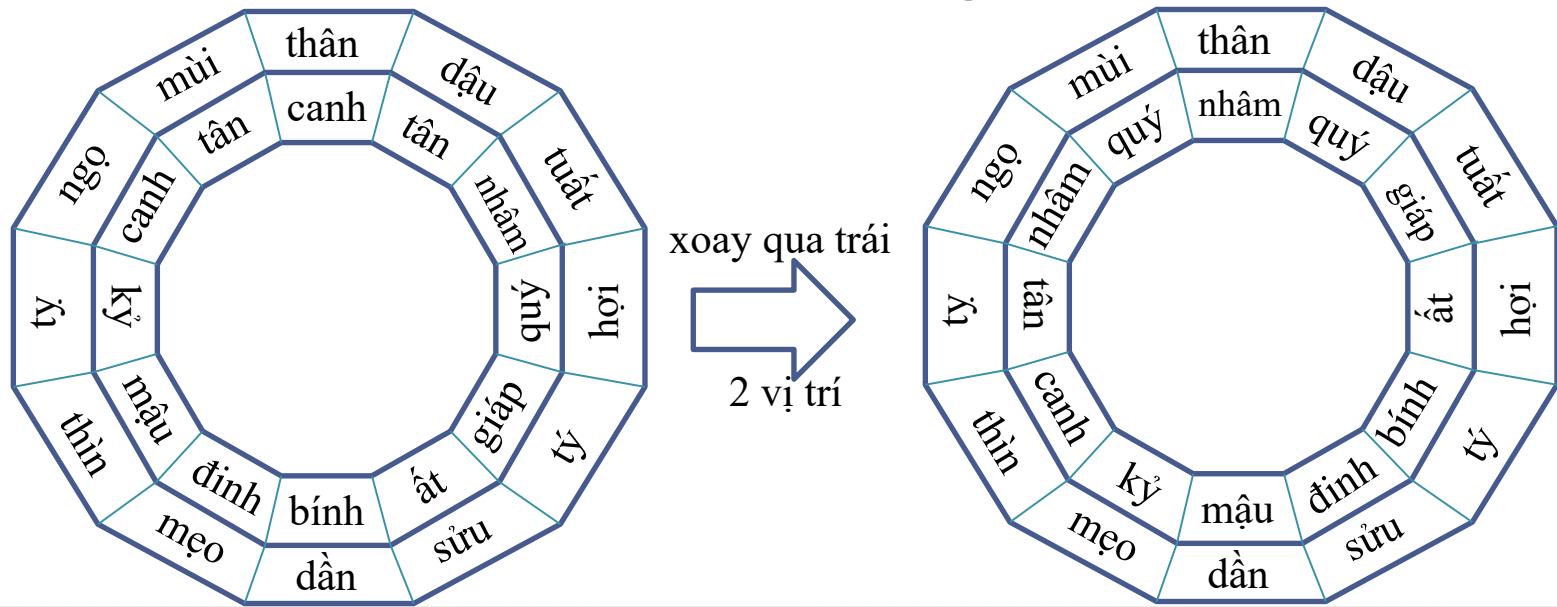
- Bài toán có quá nhiều trường hợp thì if và switch sẽ làm chương trình khó bảo trì
- Ví dụ 1: xây dựng hàm chuyển một số n thành chuỗi
  - $0 \rightarrow \text{“Zero”}$ ,  $1 \rightarrow \text{“One”}$  khi  $0 \leq n \leq 10$
  - Khi  $10 < n \leq 15 \rightarrow \text{“Eleven to fifteen”}$
  - Khi  $n > 15 \rightarrow \text{“Greater than fifteen”}$
- Thuật toán
  - `char* toString(int n){`
    - `char* s[] = {“zero”, “one”, …, “ten”}, *res = “Negative”;`
    - `if(n >= 0){`
      - `if(0 <= n && n <= 10) res = s[n];`
      - `else if(n <= 15) res = “Eleven to fifteen”;`
      - `else res = “Greater than fifteen”;`
    - `}`
    - `return res;`
  - `}`



# XỬ LÝ LẮP TRÊN MẢNG

## (Dùng làm bảng tra cứu)

- Bài toán xác định năm âm lịch
  - Ví dụ 1968 là năm mậu thân
  - “Mậu” gọi là can (có 10 can), “thân” gọi là chi/con giáp (có 12 chi)
- Cách xác định theo bảng sau



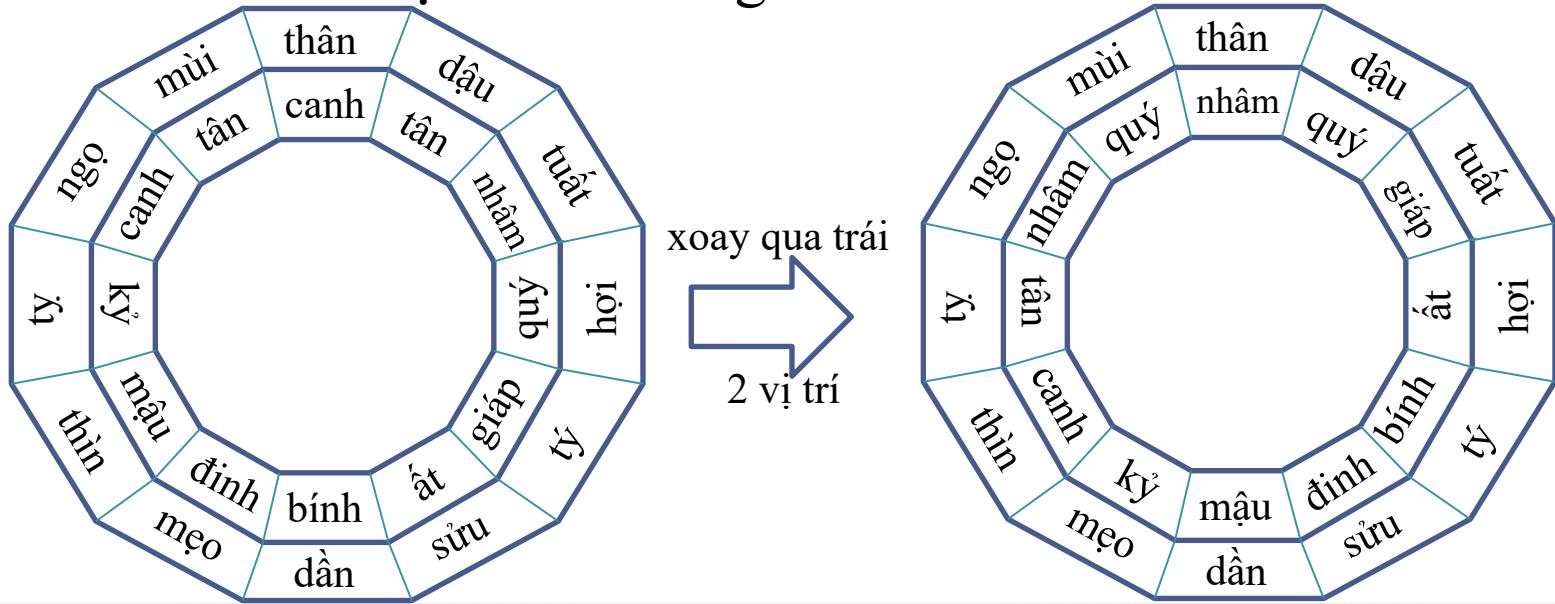
# XỬ LÝ LẬP TRÊN MẢNG

## (Dùng làm bảng tra cứu)

- Thuật toán

- ```
string toString(int y){  
    string c[] = {"canh", "tan", "nham", ..., "ky"};  
    string g[] = {"than", "dau", "tuat", ..., "mui"};  
    return c[y % 10] + " " + g[y % 12];  
}
```

- Cách xác định theo bảng sau



# XỬ LÝ LẬP TRÊN MẢNG

## (Dùng làm bảng tra cứu)

- Bài toán xác định số ngày theo tháng và năm
  - Đã giải quyết theo switch – case
  - Dùng mảng sẽ đơn giản thuật toán
- Thuật toán
  - `int checkLeap(int y){`
    - `return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);`
    - `}`
  - `int nDayOfMonth(int m, int y){`
    - `int Days[] = {-1, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};`
    - `int nDay = -1;`
    - `if(checkLeap(y)) Days[2] = 29;`
    - `nDay = Days[m];`
    - `return nDay;`
    - `}`

# XỬ LÝ LẬP TRÊN MẢNG

## (Dùng làm bảng tra cứu)

- Bài toán tính tiền điện: dùng mảng xử lý
- Thuật toán
  - `const int _ext = -1;`
  - `const int L[] = {0, 100, 150, 200, 300, 400, _ext};`
  - `const double P[] = {1242, 1304, 1651, 1788, 1912, 1962};`
  - `int nP = sizeof(P)/sizeof(P[0]), nL = sizeof(L)/sizeof(L[0]);`
  - `double subCompute(int Lm1, int Lm2, double Prc, int kWh){`
    - `if(kWh >= Lm1){`
      - `if(kWh < Lm2 || Lm2 == _ext) return (kWh - Lm1)*Prc;`
      - `else return (Lm2 - Lm1)*Prc;`
    - `}`
    - `return 0;`
  - `}`
  - `double TienDien(int kWh){`
    - `double s = 0;`
    - `for(int i = 0; i < nL - 1; i++) s+=subCompute(L[i], L[i+1], P[i], kWh);`
    - `return s*1.1; // thuế`
  - `}`

# XỬ LÝ LẬP TRÊN MẢNG

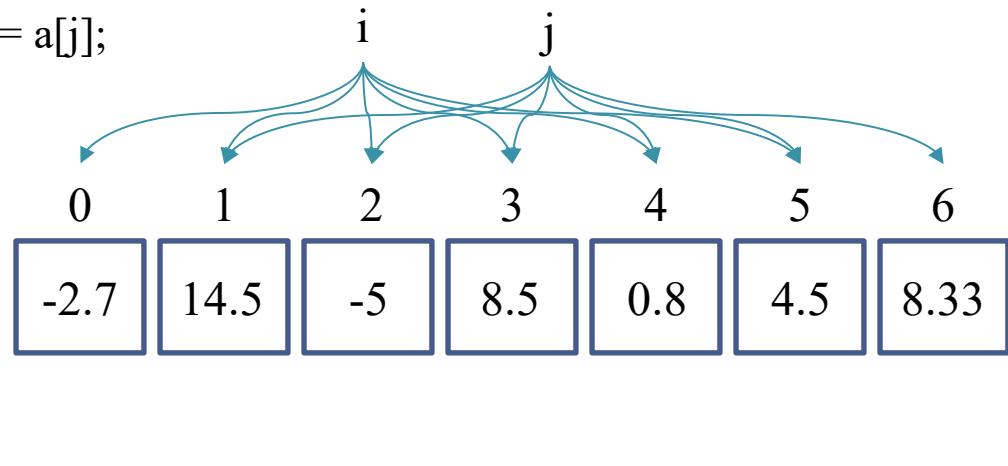
## (Sắp xếp theo thứ tự)

- Bài toán sắp xếp yêu cầu “xáo trộn” vị trí của các phần tử theo một thứ tự xác định
- Ví dụ xét mảng 8.5, -3.4, -7, 5, 8, 4.7, -2
  - Thứ tự tăng: -7, -3.4, -7, 5, 8, 4.7, -2
  - Thứ tự giảm: 8.5, 8, 5, 4.7, -2, -3.4, -7
  - Trị tuyệt đối tăng: -2, -3.4, 4.7, 5, -7, 8, 8.5
- Thuật toán đổi chỗ trực tiếp
  - Duyệt tất cả các cặp chỉ số ( $i, j: i < j$ )
    - Nếu  $a[i]$  và  $a[j]$  đứng sai vị trí
      - Đổi chỗ  $a[i]$  và  $a[j]$  // `swap(a[i], a[j])`

# XỬ LÝ LẬP TRÊN MẢNG

## (Sắp xếp theo thứ tự)

- Ví dụ sắp xếp trị tuyệt đối tăng
  - xét mảng  $\{-2.7, 14.5, -5, 8.5, 0.8, 4.5, 8.33\}$
  - Mảng sau sắp:  $\{0.8, -2.7, 4.5, -5, 8.33, 8.5, 14.5\}$
- Thuật toán:
  - `void absSort(double a[], int n){`
    - `for(int i = 0; i < n - 1; i++){`
      - `for(int j = i + 1; j < n; j++){`
        - `if(fabs(a[i]) > fabs(a[j])){`
          - `double temp = a[j];`
          - `a[j] = a[i];`
          - `a[i] = temp;`
    - `}`
    - `}`
    - `}`
  - `}`



# XỬ LÝ LẮP TRÊN MẢNG

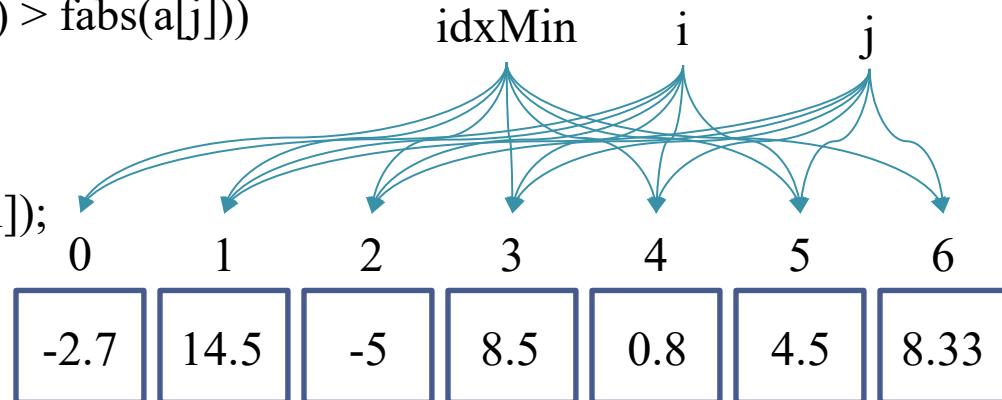
## (Sắp xếp theo thứ tự)

- Bài toán sắp xếp yêu cầu “xáo trộn” vị trí của các phần tử theo một thứ tự xác định
- Thuật toán sắp xếp chọn
  - Ý tưởng:
    - Từ vị trí  $i \in [0, n - 2]$  tìm phần tử nhỏ nhất từ  $i \rightarrow n - 1$
    - Khi xác định phần tử nhỏ nhất  $\Rightarrow$  đổi chỗ với  $a[i]$
    - Số lần đổi chỗ đúng  $n - 1$  lần
  - Mã giả
    - Lặp  $i = 0 \rightarrow n - 2 // Lặp n - 1 lần$
    - $idxMin = i;$
    - $Lặp j = i + 1 \rightarrow n - 1$ 
      - Nếu  $a[j]$  và  $a[idxMin]$  đứng sai vị trí
        - $idxMin = j;$
        - Đổi chỗ  $a[i]$  và  $a[idxMin]$

# XỬ LÝ LẬP TRÊN MẢNG

## (Sắp xếp theo thứ tự)

- Ví dụ lại sắp xếp trị tuyệt đối tăng
  - xét mảng  $\{-2.7, 14.5, -5, 8.5, 0.8, 4.5, 8.33\}$
  - Mảng sau sắp:  $\{0.8, -2.7, 4.5, -5, 8.33, 8.5, 14.5\}$
- Thuật toán sắp xếp chọn:
  - `void absSelectionSort(double a[], int n){`
    - `for(int i = 0; i < n - 1; i++){`
      - `int idxMin = i;`
      - `for(int j = i + 1; j < n; j++){`
        - `if(fabs(a[idxMin]) > fabs(a[j]))`
          - `idxMin = j;`
        - `}`
        - `swap(a[idxMin], a[i]);`
      - `}`
    - `}`



# XỬ LÝ LẬP TRÊN MẢNG

## (Thao tác trên mảng hai chiều)

- Trường hợp mảng hai chiều có kích thước nhỏ. Ta có thể dùng **typedef** định nghĩa sẵn kiểu:
- Ví dụ:
  - `#define MaxSz 20`
  - `typedef double MATRIX[MaxSz][MaxSz];`
  - `void mtPrint(MATRIX& A, int n){`
    - `for(int i = 0; i < n; i++){`
      - `for(int j = 0; j < n; j++)`
        - `printf("%lf ", A[i][j]);`
      - `}`
      - `printf("\n");`
    - `}`
  - `void main(){`
    - `MATRIX A = {{1, 2, 3}, {0, 4, 1}, {1, 5, 6}};`
    - `mtPrint(A, 3);`
  - `}`

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Có hai cách duyệt cơ bản
  - Cách 1:
    - Duyệt theo từng dòng (từ trên xuống dưới)
    - Mỗi dòng duyệt theo từng cột (từ trái qua phải)
    - Ví dụ: mảng a gồm m dòng và n cột
      - `for(int i = 0; i < m; i++)`
      - `for(int j = 0; j < n; j++)`
  - Cách 2:
    - Duyệt theo từng cột (từ trái qua phải)
    - Mỗi cột duyệt theo từng dòng (từ trên xuống dưới)
    - Ví dụ: mảng a gồm m dòng và n cột
      - `for(int j = 0; j < n; j++)`
      - `for(int i = 0; i < m; i++)`

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

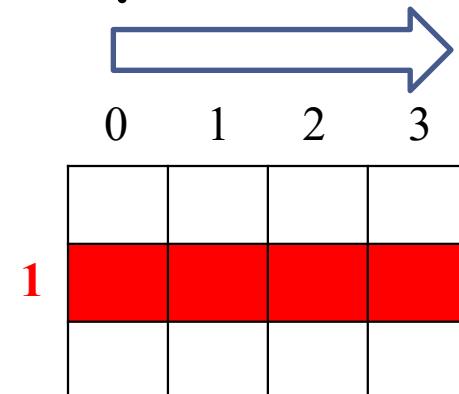
- Duyệt trên một dòng k cố định nào đó

- `for(int j = 0; j < n; j++)`

- {

- // Xử lý `a[k][j]`

- }



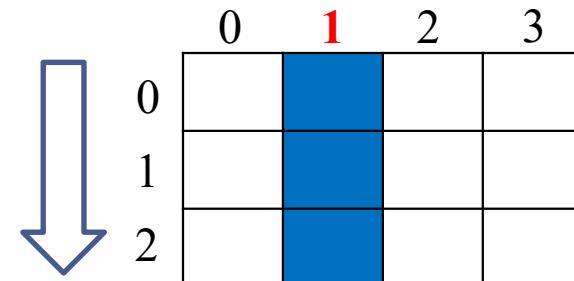
- Duyệt trên một cột k cố định nào đó

- `for(int i = 0; i < m; i++)`

- {

- // Xử lý `a[i][k]`

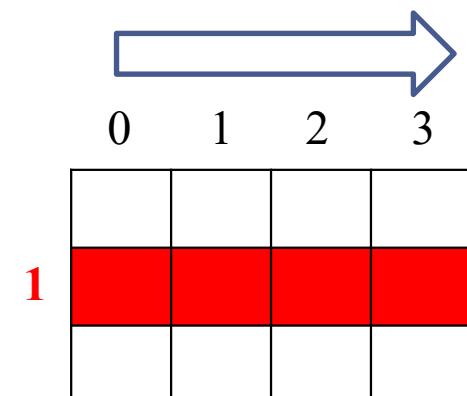
- }



# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ viết hàm tính tổng trên một dòng k
  - `typedef double MAT[10][10];`
  - `double sumR(MAT& A, int n, int k){`
    - `if(k < 0 || k >= n || n <= 0 || n > 10)`
      - `return 0;`
    - `double sum = 0;`
    - `while(n--){ // kiểm tra trước, trừ sau`
      - `sum+=A[k][n];`
    - `}`
    - `return sum;`
  - `}`



# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Cách duyệt đường chéo
  - Đường chéo chính:
    - Duyệt theo từng cột (từ trái qua phải)
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - // xử lý `a[i][i];`
  - Đường chéo phụ:
    - Duyệt theo từng cột (từ trái qua phải)
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - // xử lý `a[i][n - 1 - i];`

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 | ■ |   |   |
| 1 |   | ■ |   |
| 2 |   |   | ■ |

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 |   |   | ■ |
| 1 |   | ■ |   |
| 2 | ■ |   |   |

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Cách duyệt phần tử trên/dưới đường chéo
  - Trên đường chéo chính:
    - Duyệt theo từng dòng (từ trên xuống dưới)
    - Ứng với từng dòng, duyệt theo cột mong muốn
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - `for(int j = i + 1; j < n; j++)`
      - `// xử lý a[i][j];`
  - Dưới đường chéo chính:
    - Duyệt theo từng dòng (từ trên xuống dưới)
    - Ứng với từng dòng, duyệt theo cột mong muốn
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - `for(int j = 0; j < i; j++)`
      - `// xử lý a[i][j];`

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 |   | ■ | ■ |
| 1 |   |   | ■ |
| 2 |   |   |   |

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 | ■ |   |   |
| 1 |   | ■ |   |
| 2 | ■ | ■ |   |

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Cách duyệt phần tử trên/dưới đường chéo
  - Trên đường chéo phụ:
    - Duyệt theo từng dòng (từ trên xuống dưới)
    - Ứng với từng dòng, duyệt theo cột mong muốn
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - `for(int j = 0; j < n - 1 - i; j++)`
      - `// xử lý a[i][j];`
  - Dưới đường chéo phụ:
    - Duyệt theo từng dòng (từ trên xuống dưới)
    - Ứng với từng dòng, duyệt theo cột mong muốn
    - Ví dụ: mảng a gồm n dòng và n cột
      - `for(int i = 0; i < n; i++)`
      - `for(int j = n - i; j < n; j++)`
      - `// xử lý a[i][j];`

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 |   |   |   |
| 1 |   |   |   |
| 2 |   |   |   |

|   |   |   |   |
|---|---|---|---|
|   | 0 | 1 | 2 |
| 0 |   |   |   |
| 1 |   |   |   |
| 2 |   |   |   |

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 1 xét ma trận vuông ( $n \times n$ ): kiểm tra tổng các phần tử trên mỗi dòng có tăng dần từ **trên xuống** dưới hay **không**

|    |   |   |   |
|----|---|---|---|
| 6  | 1 | 2 | 3 |
| 15 | 4 | 5 | 6 |
| 24 | 7 | 8 | 9 |

|   |   |   |    |
|---|---|---|----|
| 4 | 5 | 6 | 15 |
| 1 | 2 | 3 | 6  |
| 7 | 8 | 9 | 24 |

- Phát biểu hình thức: cần kiểm tra mệnh đề

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 1: cần kiểm tra mệnh đề
- Đoạn mã
  - `typedef double MATRIX[3][3];`
  - `double sumR(MATRIX &A, int n, int r){ // r là dòng`
    - `double s = 0;`
    - `while(n--) s+=A[r][n];`
    - `return s;`
  - `}`
  - `bool isIncrease(MATRIX &A, int n){`
    - `bool result = true;`
    - `int i = n;`
    - `while(--i){`
      - `double s1 = sumR(A, n, i - 1), s2 = sumR(A, n, i);`
      - `if(s1 > s2) result = false;`
    - `}`
    - `return result;`

Ví dụ  $n = 3$

Cần tính:

- `sumR(1) & sumR(2)`
- `sumR(0) & sumR(1)`

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 1: cần kiểm tra mệnh đề
- Đoạn mã cài tiến
  - `typedef double MATRIX[3][3];`
  - `double sumR(MATRIX &A, int n, int r){ // r là dòng`
    - `double s = 0;`
    - `while(n--) s+=A[r][n];`
    - `return s;`
  - `}`
  - `bool isIncrease(MATRIX &A, int n){`
    - `bool result = true;`
    - `int i = n - 1; double s2 = sumR(A, n ,i);`
    - `while(i--){`
      - `double s1 = sumR(A, n, i);`
      - `if(s1 > s2) result = false;`
      - `s2 = s1; // Lưu lại s1 để so sánh tiếp với dòng trên`
    - `}`

Ví dụ  $n = 3$

Cần tính trước  $\text{sumR}(2)$ , sau đó

- $\text{sumR}(1)$  & lưu  $\text{sumR}(1)$
- $\text{sumR}(0)$  & lưu  $\text{sumR}(0)$

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 2 sắp xếp ma trận vuông ( $n \times n$ ) sao cho tổng các phần tử trên mỗi dòng giảm dần từ trên xuống

|    |   |   |   |
|----|---|---|---|
| 6  | 1 | 2 | 3 |
| 15 | 4 | 5 | 6 |
| 24 | 7 | 8 | 9 |



|    |   |   |   |
|----|---|---|---|
| 24 | 7 | 8 | 9 |
| 15 | 4 | 5 | 6 |
| 6  | 1 | 2 | 3 |

- Phát biểu hình thức: cần thỏa mệnh đề

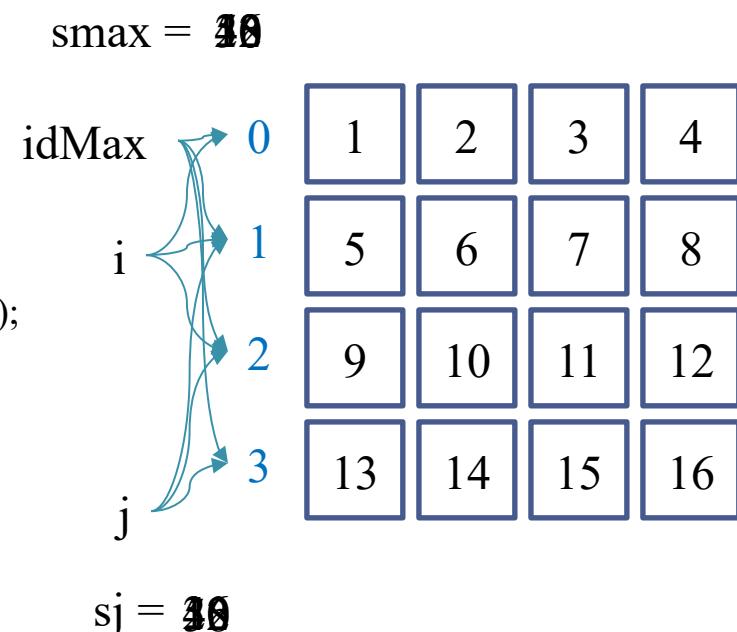
# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 2: cần thỏa mệnđ đề

- Đoạn mã

- //...
- `void swapR(MATRIX &A, int n, int i, int k){` //i,k là dòng cần swap
  - `for(int j = 0; j < n; j++)`
  - `swap(A[i][j], A[k][j]);`
- }
- `bool sortR(MATRIX &A, int n){`
  - `for(int i = 0; i < n - 1; i++){`
  - `int idxMax = i; double smax = sumR(A, n, i);`
  - `for(int j = i + 1; j < n; j++){`
  - `double sj = sumR(A, n, j);`
  - `if(sj > smax) {idMax = j; smax = sj;}`
  - }
  - `swapR(A, n, i, idMax);`
- }



# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 3 kiểm tra ma trận vuông ( $n \times n$ ) có chứa hai dòng nào trùng nhau hay không



|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |



|   |   |   |
|---|---|---|
| 7 | 8 | 9 |
| 7 | 8 | 9 |
| 1 | 2 | 3 |

- Phát biểu hình thức:

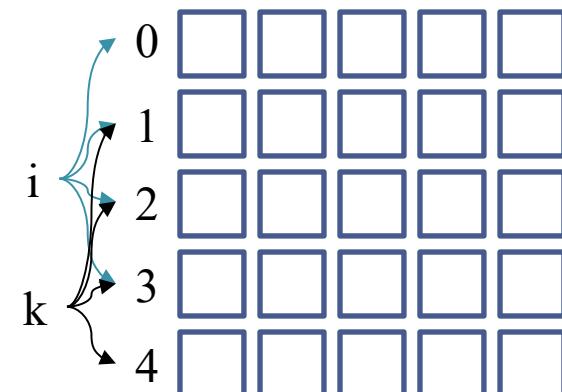
# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 3: cân thỏa mệnh đề

- Đoạn mã

- `bool rowsEqual(MATRIX &A, int n, int i, int k){`
  - `while(n--)`
  - `if(A[i][n] != A[k][n])`
  - `return false;`
  - `return true;`
- `}`
- `bool rowsDup(MATRIX &A, int n){`
  - `for(int i = 0; i < n - 1; i++)`
  - `for(int k = i + 1; k < n; k++)`
  - `if(rowsEqual(A, n, i, k)) return true;`
  - `return false;`
- `}`



# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 4 tìm số dương nhỏ nhất trên đường chéo chính trong ma trận vuông ( $n \times n$ )

|   | 0 | 1  | 2 |
|---|---|----|---|
| 0 | 7 | 8  | 9 |
| 1 | 4 | -5 | 6 |
| 2 | 1 | 2  | 3 |

- Đoạn mã
  - `int PositiveMinOnDiag(MATRIX& A, int n){`
    - `int idx = -1;`
    - `for(int i = 0; i < n; i++){`
      - `if(A[i][i] > 0)`
      - `if(idx == -1 || A[idx][idx] > A[i][i])`
        - `idx = i;`
    - `}`
    - `return idx;`
  - `}`

# XỬ LÝ LẬP TRÊN MẢNG

## (Duyệt trên mảng hai chiều)

- Ví dụ 5 kiểm tra xem có tồn tại x ở phần phía dưới đường chéo phụ trong ma trận vuông ( $n \times n$ )

|   |   |    |   |
|---|---|----|---|
| 0 | 7 | 8  | 9 |
| 1 | 4 | -5 | 6 |
| 2 | 1 | 2  | 3 |

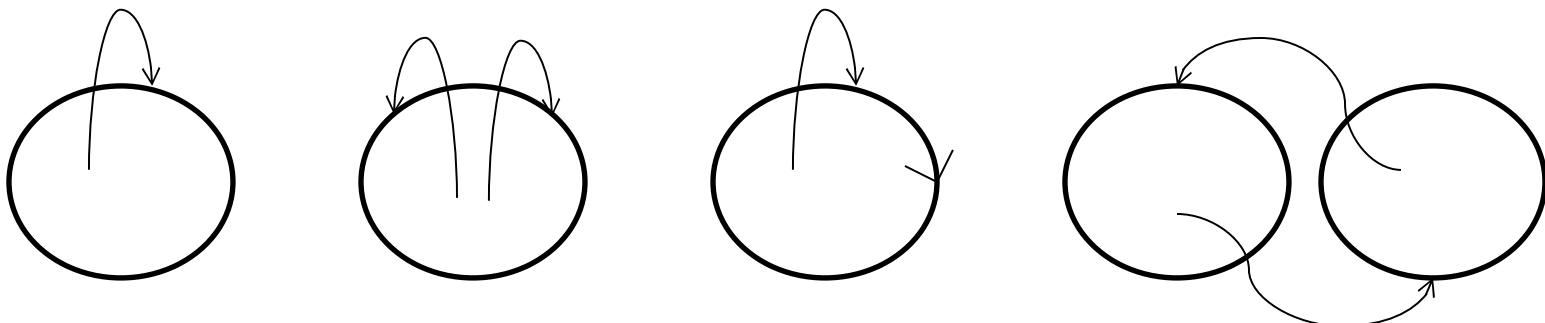
- Phát biểu hình thức:

- Đoạn mã

- `bool TriangleSearch(MATRIX& A, int n, double x){`
  - `for(int i = 0; i < n; i++){`
    - `for(int j = n - i; j < n; j++)`
      - `if(A[i][j] == x) return true;`
      - `}`
      - `return false;`
  - `}`

# KỸ THUẬT ĐỆ QUY

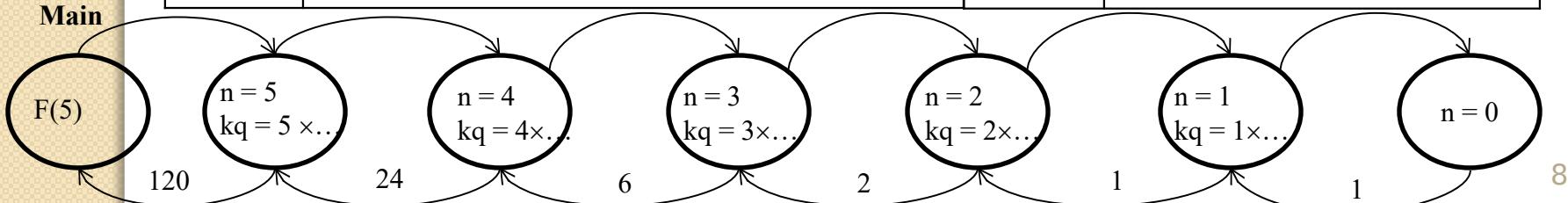
- Thuật toán dạng đệ quy có bốn loại chính
  - Đệ quy tuyến tính
  - Đệ quy nhị phân
  - Đệ quy phi tuyến
  - Đệ quy hõi tương
- Hình ảnh



# KỸ THUẬT ĐỆ QUY

- Ví dụ 1: bài toán tính  $n! = 1 \times \dots \times n$
- Nhận xét:
  - $n! = (1 \times \dots \times n - 1) \times n = (n - 1)! \times n$
  - $n = 0 \Rightarrow 0! = 1$

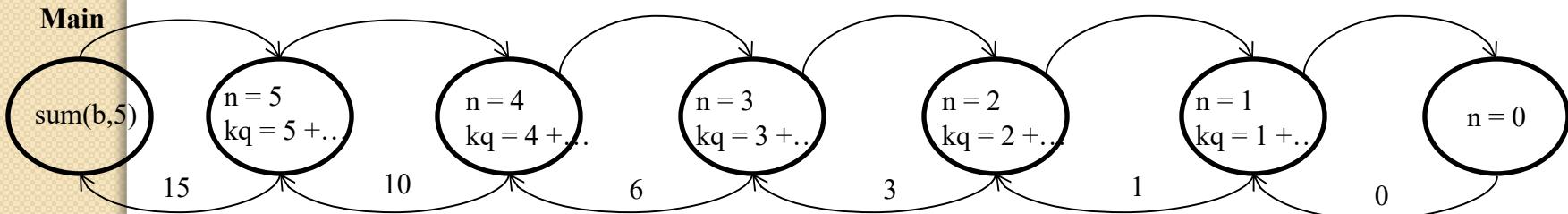
| Dòng | Mô tả                                          |    |                                             |
|------|------------------------------------------------|----|---------------------------------------------|
| 1    | <code>long Factorization(long n){</code>       | 6  | <code>void main(){</code>                   |
| 2    | <code>if(n &lt;= 0) return 1;</code>           | 7  | <code>long r;</code>                        |
| 3    | <code>long kq = n*Factorization(n - 1);</code> | 8  | <code>r = Factorization(5);</code>          |
| 4    | <code>return kq;</code>                        | 9  | <code>cout &lt;&lt; r &lt;&lt; endl;</code> |
| 5    | <code>}</code>                                 | 10 | <code>}</code>                              |



# KỸ THUẬT ĐỆ QUY

- Ví dụ 2: tổng mảng  $S = a[0] + \dots + a[n - 1]$
- Nhận xét:
  - Nếu  $n \geq 1 \Rightarrow S_{n-1} = (a[0] + \dots + a[n - 2]) + a[n - 1] = S_{n-2} + a[n - 1]$
  - Nếu  $n \leq 0 \Rightarrow S = 0$

| Dòng | Mô tả                                            |    |                                             |
|------|--------------------------------------------------|----|---------------------------------------------|
| 1    | <code>long sum(long a[], int n){</code>          | 6  | <code>void main(){</code>                   |
| 2    | <code>if(n &lt;= 0) return 0;</code>             | 7  | <code>long b[] = {1,2,3,4,5}, r;</code>     |
| 3    | <code>long kq = a[n - 1] + sum(a, n - 1);</code> | 8  | <code>r = sum(b, 5);</code>                 |
| 4    | <code>return kq;</code>                          | 9  | <code>cout &lt;&lt; r &lt;&lt; endl;</code> |
| 5    | <code>}</code>                                   | 10 | <code>}</code>                              |



# KỸ THUẬT ĐỆ QUY

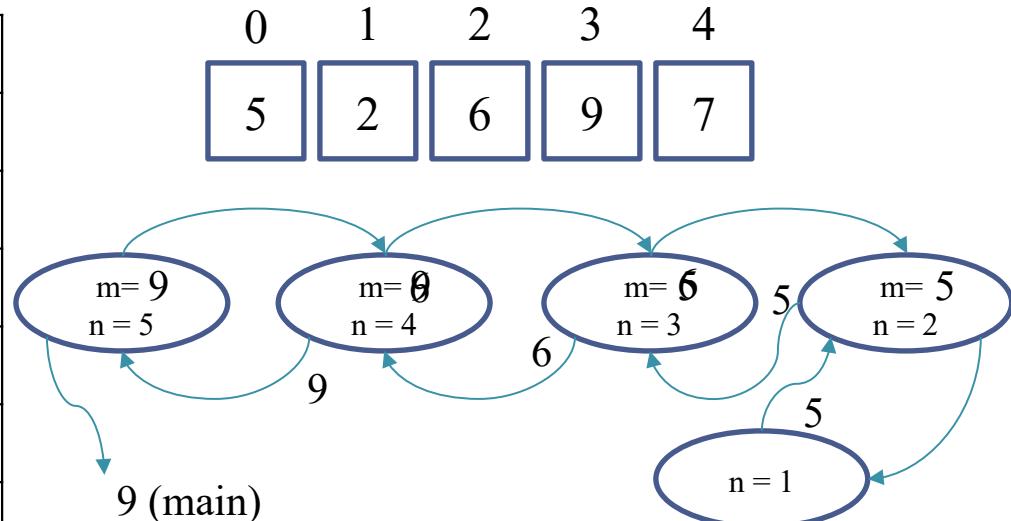
- Ví dụ 3: tính tông sau với  $n \in \mathbb{Z}^*$ 
  - $E_n = 1/1^k + 1/2^k + \dots + 1/n^k.$
  - ↔  $E_n = 1/1^k + \dots + 1/(n-1)^k + 1/n^k = E_{n-1} + 1/n^k$

| Dòng | Mô tả                         |
|------|-------------------------------|
| 1    | long eulerFunc(int n, int k){ |
| 2    | if(n <= 0    k < 1) return 0; |
| 3    | int a = 1/pow(n, k);          |
| 4    | int b = eulerFunc(n - 1, k);  |
| 5    | return a + b;                 |
| 6    | }                             |

# KỸ THUẬT ĐỆ QUY

- Ví dụ 4: tìm phần tử lớn nhất trong mảng
- Ý tưởng
  - Nếu mảng có 1 phần tử thì đó là phần tử max
  - Nếu mảng có nhiều phần tử, ta tìm phần tử lớn nhất trong  $n - 2$  phần tử, sau đó so sánh với phần tử cuối ( $a[n - 1]$ ) để tìm ra phần tử max

| Dòng | Mô tả                              |
|------|------------------------------------|
| 1    | double Max(double a[], int n){     |
| 2    | if( $n \leq 1$ ) return a[0];      |
| 3    | double m = Max(a, n - 1);          |
| 4    | if( $m < a[n - 1]$ ) m = a[n - 1]; |
| 5    | return m;                          |
| 6    | }                                  |



# KỸ THUẬT ĐỆ QUY

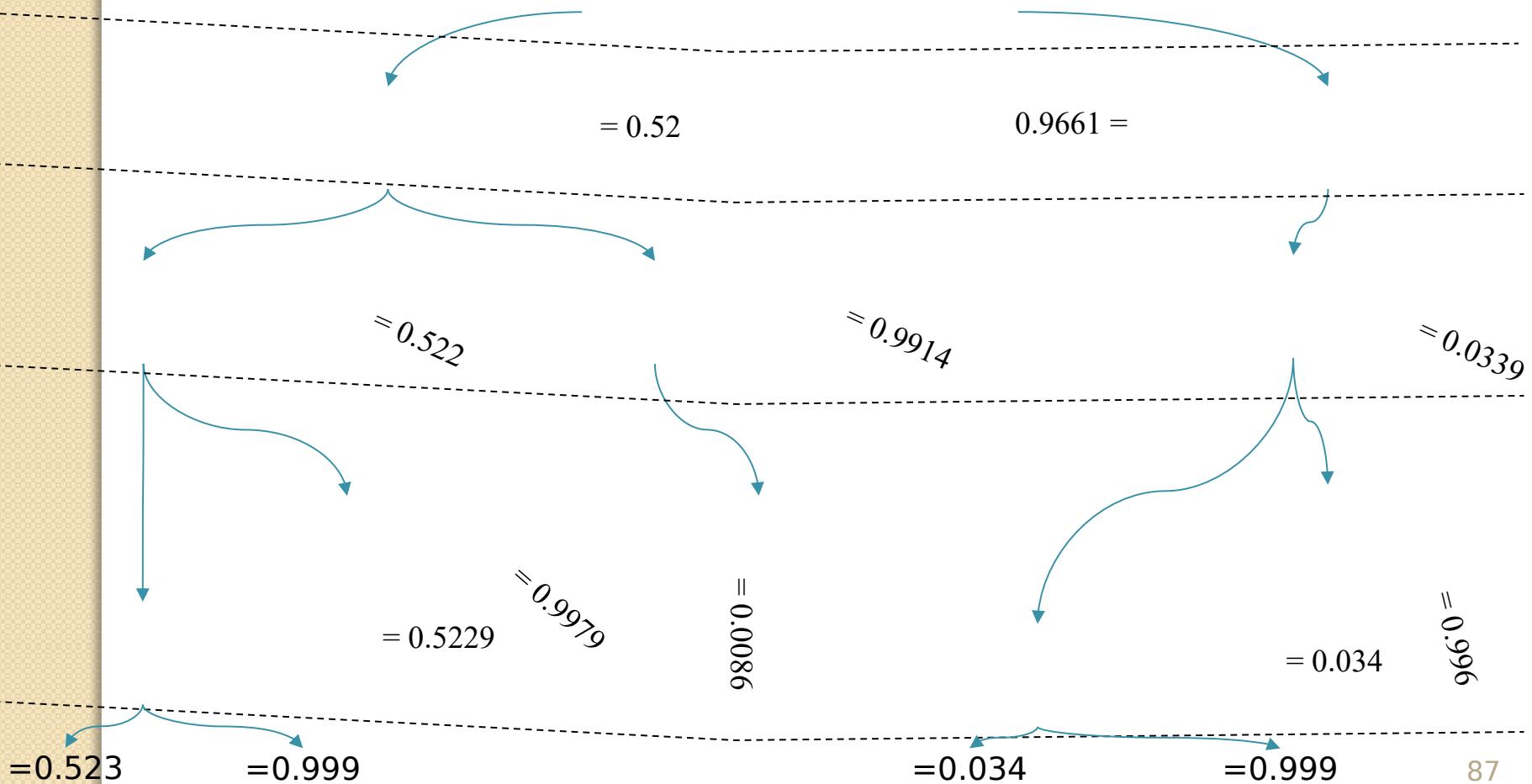
- Ví dụ 5: bài toán tính sin & cos
  - $\sin(x) = 2\sin(x/2)\cos(x/2)$
  - $\cos(x) = 1 - 2\sin^2(x/2)$
  - Khi  $|x| < \varepsilon$  thì
    - $\sin(x) = x - x^3/6$
    - $\cos(x) = 1 - x^2/2$

|   |                                 |
|---|---------------------------------|
| 1 | #define epsilon 0.000001        |
| 2 | double Sin(double x){           |
| 3 | if(-epsilon<=x && x <= epsilon) |
| 4 | return x - x*x*x/6;             |
| 5 | return 2*Sin(x/2)*Cos(x/2);     |
| 6 | }                               |

|   |                                 |
|---|---------------------------------|
| 1 |                                 |
| 2 | double Cos(double x){           |
| 3 | if(-epsilon<=x && x <= epsilon) |
| 4 | return 1 - x*x/2;               |
| 5 | return 1 - 2*Sin(x/2)*Sin(x/2); |
| 6 | }                               |

# KỸ THUẬT ĐỆ QUY

- Ví dụ 5: giả sử tính



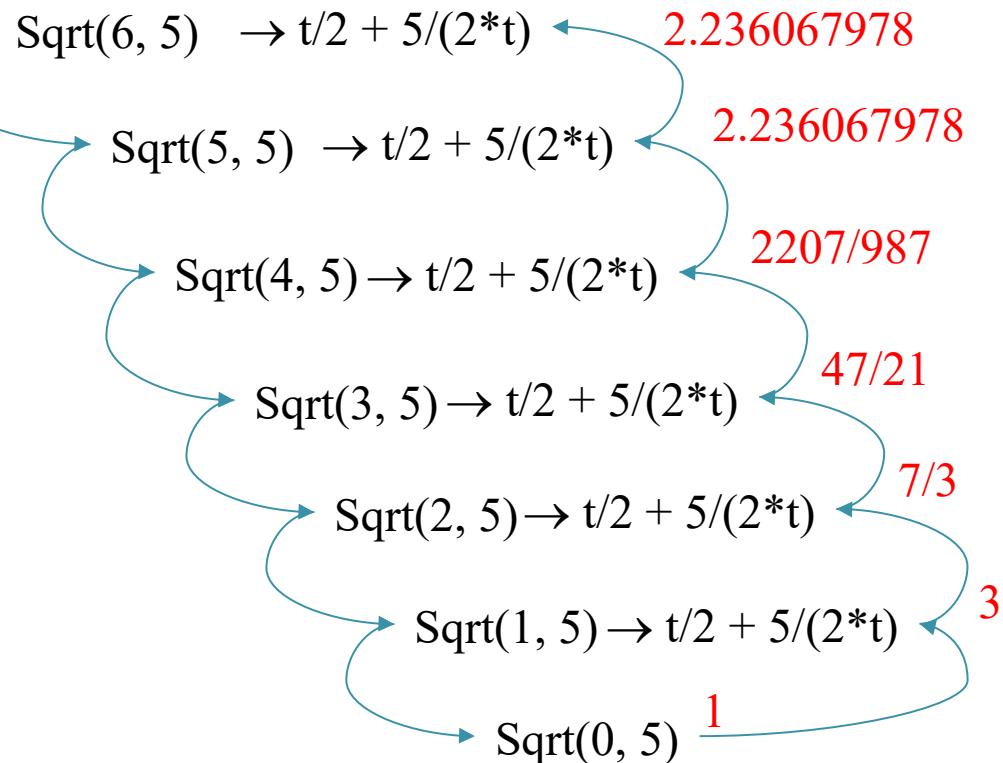
# KỸ THUẬT ĐỆ QUY

- Ví dụ 6: bài toán tính xấp xỉ căn bậc hai

- $x_0 = 1$  nếu  $n = 0$

- $x_n = x_{n-1}/2 + a/(2 \times x_{n-1})$  ( $n \geq 1$  &  $a \in \mathbb{R}^*$ )

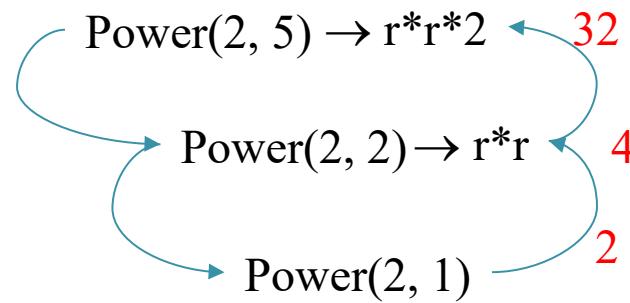
```
1 double Sqrt(int n, double a){  
2     double t;  
3     if(a <= 0) return 0;  
4     if(n <= 0) return 1;  
5     t = Sqrt(n - 1, a);  
6     return t/2 + a/(2*t);  
7 }
```



# KỸ THUẬT ĐỆ QUY

- Ví dụ 7: bài toán tính lũy thừa nhanh
  - $x^n = (x^{n/2})^2$  nếu  $n$  chẵn
  - $x^n = (x^{(n-1)/2})^2$  nếu  $n$  lẻ

```
1 double Power(double x, int n){  
2     double r;  
3     if(n < 0) return 1/(Power(x, -n));  
4     if(n == 0) return 1;  
5     if(n == 1) return x;  
6     r = Power(x, n/2);  
7     if(n % 2 == 0) return r*r;  
8     return r*r*x;  
9 }
```



# BÀI TẬP

- Áp dụng kĩ thuật đệ quy xây dựng các hàm với yêu cầu sau
  - Hàm **đếm số lượng** số chẵn trong mảng các số nguyên
  - Hàm **in ra các vị trí** mà tại đó giá trị phần tử **âm** trong mảng các số
  - Hàm **kiểm tra** xem mảng có phải **tăng dần** hay không