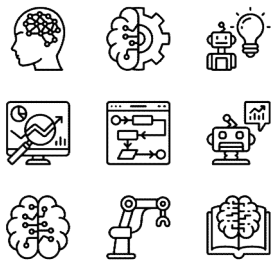
Computer Science for Practicing Engineers

Phương pháp chia để trị



TS. Huỳnh Bá Diệu
Email: dieuhb@gmail.com
Phone: 0914146868

Phương pháp Chia để trị (Divide and Conquer)

Các phương pháp thiết kế thuật toán

- Phương pháp Vét cạn (Brute-force or exhaustive search)
- Phương pháp quay lui (Backtracking)
- Phương pháp Chia để trị (Divide and Conquer)
- Phương pháp Qui hoạch động (Dynamic Programming)
- Phương pháp tham lam (Greedy Algorithms)
- *Phương pháp ngẫu nhiên (Randomized Algorithm)*

Phương pháp Chia để trị (Divide and Conquer)

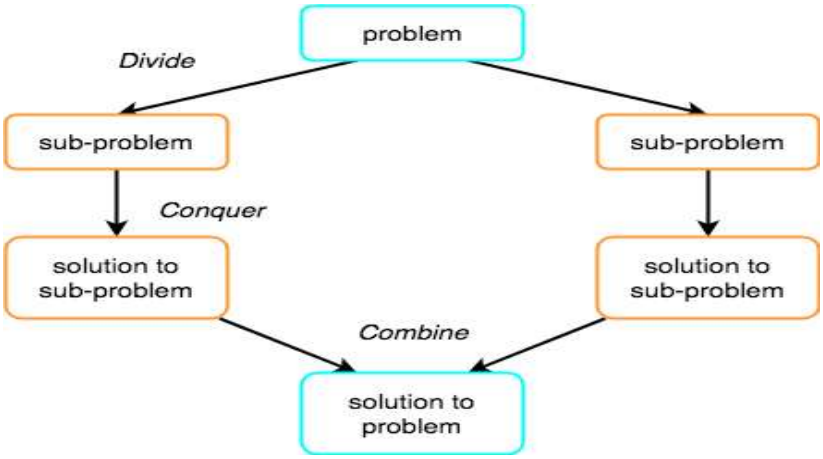
Nội dung:

1. Giới thiệu kỹ thuật chia để trị
2. Thuật toán sắp xếp trộn
3. Thuật toán tính X^n
4. Thuật toán nhân Ấn Độ
5. Thuật toán tìm kiếm Ternary
6. Thuật toán nhân Karatsuba

Phương pháp chia để trị (divide and conquer)

Divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
The solutions to the sub-problems are then combined to give a solution to the original problem.

Phương pháp chia để trị (divide and conquer)



Phương pháp chia để trị (divide and conquer)

There are mainly three steps in a divide and conquer algorithm

Divide — The problem is broken down or divided into a number of subproblems that are smaller instances of the same problem.

Conquer — The subproblems are solved recursively. If the subproblem sizes are small enough , however, just solve them in a straightforward manner.

Combine — The solution to the subproblems are combined to get the solution for the main problem.

Phương pháp chia để trị (divide and conquer)

Advantages 1

Solving difficult problems

Divide and conquer is a powerful tool for solving conceptually difficult problems: all it requires is a way of breaking the problem into sub-problems, of solving the trivial cases and of combining sub-problems to the original problem

Phương pháp chia để trị (divide and conquer)

Advantages 1

Solving difficult problems

Similarly, decrease and conquer only requires reducing the problem to a single smaller problem, such as the classic Tower of Hanoi puzzle, which reduces moving a tower of height n to moving a tower of height n − 1.

Phương pháp chia để trị (divide and conquer)

Advantages 2

Algorithm efficiency

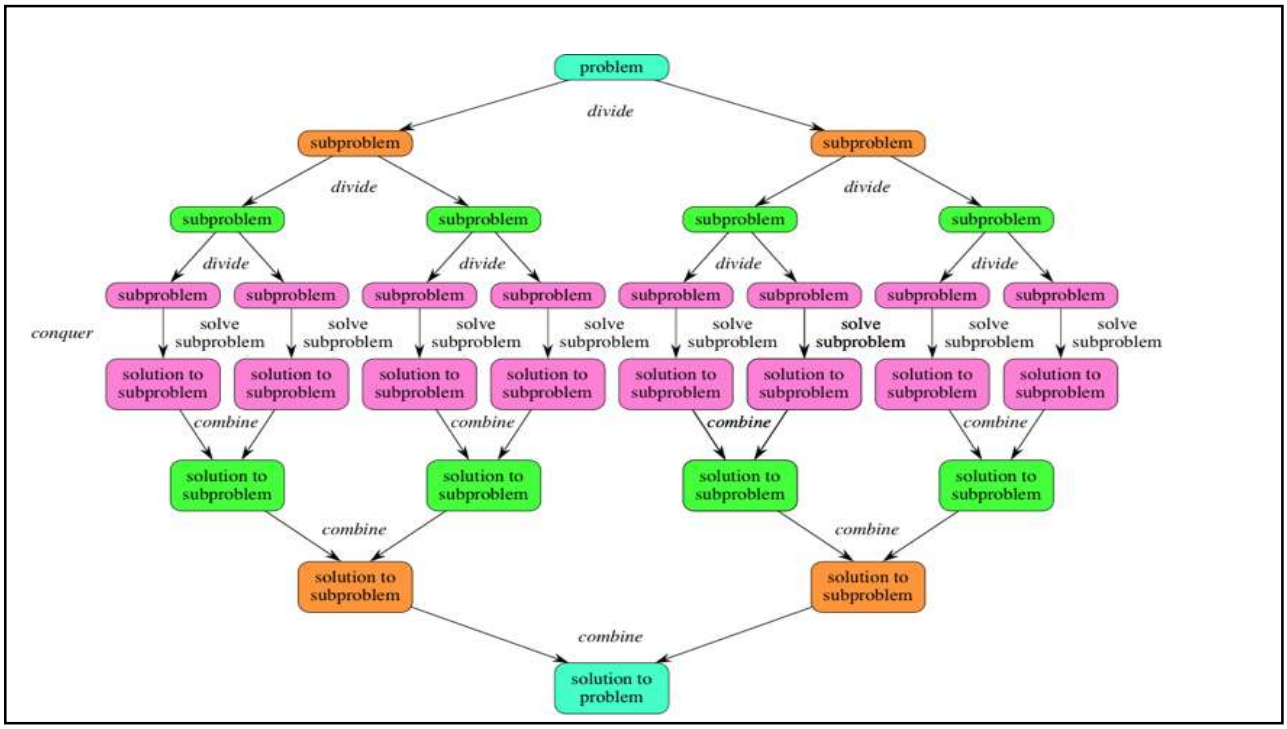
The divide-and-conquer paradigm often helps in the discovery of efficient algorithms. It was the key, for example, to Karatsuba's fast multiplication method, the quicksort and mergesort algorithms, the Strassen algorithm for matrix multiplication, and fast Fourier transforms.

Phương pháp chia để trị (divide and conquer)

Advantages 3

Parallelism

Divide-and-conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance, because distinct sub-problems can be executed on different processors.



Phương pháp chia để trị (divide and conquer)

- **Chia bài toán lớn thành các bài toán con**
- **Giải các bài toán con**
- **Kết hợp lời giải các bài toán con để có lời giải bài toán lớn**

Ví dụ: Sắp xếp dãy a gồm n phần tử

+ Chia thành 2 dãy con L và R

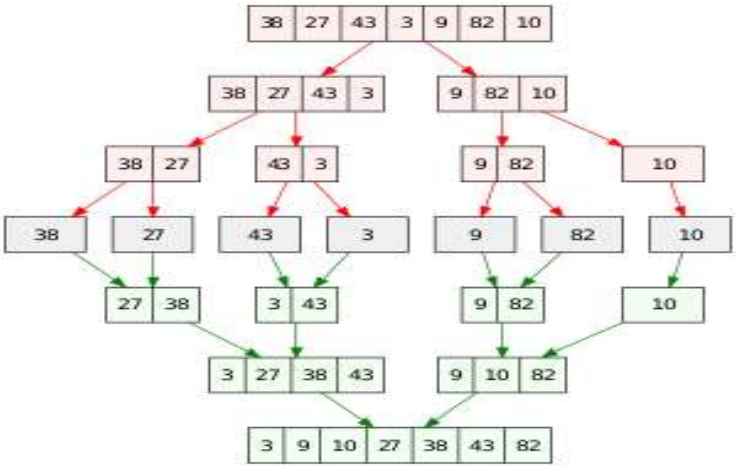
+ Sắp xếp dãy L

+ Sắp xếp dãy R

+ Trộn hai dãy có thứ tự thành 1 dãy có thứ tự

Phương pháp chia để trị (divide and conquer)

Ví dụ: Sắp xếp dãy a gồm n phần tử



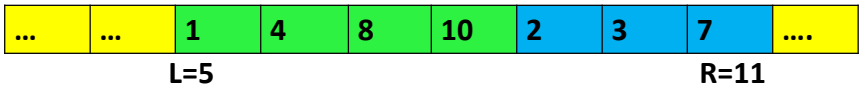
Phương pháp chia để trị: **Merge Sort**

Thuật toán MergeSort sắp xếp dãy từ left (L) đến right (R)

```
MergeSort(L, R)
If L<R
        m= (L+R)/2;
        MergeSort(L, m);
        MergeSort(m+1, R);
        MERGE(L, R);
End if
```

Thuật toán Merge (trộn hai dãy có thứ tự)

```
Merge(L, R){
 if (L< R){
      + m= (L+R)/2; v=0;  i=L; j= m+1;   // i đầu dãy TRAI và j đầu dãy PHAI
      + cấp phát mảng phụ kq gồm [R-L +1] phần tử
       + while (i<=m && j<=R)
          nếu a[i]<= a[j] kq[v++]= a[i++]; // nếu đầu dãy TRAI nhỏ hơn thì cho vào T
          ngược lại kq[v++]= a[j++];          // ngược lại cho đầu dãy PHAi vào T
        + while (i<=m ) kq[v++]= a[i++];
        + while (j<=R) kq[v++]= a[j++];
        + for (int i=0; i<=R-L; i++) a[L + i] = kq[i];
   }
}
```



Sắp xếp trộn

```
#include<bits/stdc++.h>
using namespace std;
void sinh(int *&a, int &n){
        //cout<<"\n so phan tu mang can sinh:"; cin>>n;
        a= new int[n];
        for(int i=0; i<n; i++)  a[i] =rand()%10001;
}
void in(int *&a, int &n){
        cout<<"\n noi dung mang:\n";
        for(int i=0; i<n; i++) cout<<a[i] <<" ";
}
```

```
int main()
{
    int *a, n=10;
    sinh(a,n);   in(a,n);
}
```

Sắp xếp trộn

```
void Mergesort(int*&a, int L, int R) {
        if(L<R){
                int m=(L+R)/2;
                Mergesort(a,  L, m); Mergesort(a,  m+1, R); Merge(a, L, R);
        }
}
int main(){
        int *a, n=10;   sinh(a,n);  in(a,n);
        Mergesort(a, 0, n-1);  cout<<"\n day sau khi sap xep:\n";  in(a,n);
}
```

Phương pháp chia để trị: **Merge Sort**

```
void Merge(int*&a, int L, int R){
        int m= (L+R)/2;
        int i=L, j=m+1, v=0;
        int *kq= new int[R-L+1]; // mang ket qua tam chua day tang dan
         while (i<=m && j<=R)
           if(a[i]<=a[j]) { kq[v]=a[i]; i++; v++;}  else { kq[v] =a[j]; v++; j++;}
         while (i<=m) { kq[v]=a[i]; i++; v++;}
         while (j<=R ) { kq[v] =a[j]; v++; j++;}
         i=L;  for(j=0; j<v; j++)  a[i++] = kq[j];
        if(kq!=NULL) delete []kq;
}
```

Phương pháp chia để trị: **Merge Sort**

Yêu cầu viết chương trình:

- **Sinh ngẫu nhiên dãy n số nguyên: sinh(int n)**
- **In dãy n số :inday()**
- **Sắp xếp theo thuật toán MergeSort**

So sánh thời gian chạy chương trình khi sắp xếp dãy theo thuật toán merge sort và bubble sort *(n=10000, n= 1000000)*???

Phương pháp chia để trị: **Merge Sort**

```
int main(){
        int *a, n;
        long t1, t2;
        sinh(a,n);   //in(a,n);
        t1= clock();        Mergesort(a, 0, n-1);   t2= clock();
        //cout<<"\n day sau khi sap xep:\n"; in(a,n);
        cout<<"\n thoi gian chay thuat toan=" <<t2-t1;
}
```

```
Mang A:----------
 5564 2633 5682 6504 6560 258 2800 1259 9205 1382
 ----------------

Mang A:----------
258 1259 1382 2633 2800 5564 5682 6504 6560 9205
----------------
```

Phương pháp chia để trị: Tìm max min

Ví dụ 2: Tìm max min trong dãy A

Cho dãy A. Hãy tìm số max và số min trong dãy.

Cách 1: Duyệt tuần tự, tìm max, tìm min

Cách 2: Tiếp cận theo phương pháp chia để trị

+ Chia làm hai đoạn từ L→mid và từ mid +1 → R
+ Tìm **max1, min1** trong đoạn L → mid
+ Tìm **max2, min2** trong đoạn mid +1 → R
+ So sánh min1, min2 để tìm **min**, so sánh max1, max2 để tìm **max**

Phương pháp chia để trị: Tìm max min

```
void MAXMIN0(int*&a, int n)
{
        int ma,mi; ma=mi=a[0];
        for(int i=1; i<n; i++) {
                if(a[i]>ma) ma=a[i];
                if(a[i]<mi) mi=a[i];
        }
        cout<<"\n gia tri lon nhat la "<<ma<<"\t gia tri nho nhat la "<<mi;
}
```

Phương pháp chia để trị: Tìm max min

```
void MaxMin(int*&a, int L, int R, int &ma, int &mi){
        if(L>R);
        else if(L==R) ma= mi= a[L];
        else   {
                int m= (L+R)/2;     int max1, min1, max2, min2;
                MaxMin(a, L, m, max1, min1); MaxMin(a, m+1, R, max2, min2);
                if(max1>max2) ma=max1; esle ma= max2;
                if(min1<min2) mi=min1; else mi= min2;
        }
}
```

Phương pháp chia để trị: Tìm max min

```
void MAXMIN(int*&a, int n)
{
        int ma,mi; ma=mi=a[0];
        MaxMin(a,0,n-1, ma, mi);
        cout<<"\n gia tri lon nhat la "<<ma<<"\t gia tri nho nhat la "<<mi;
}
```

Phương pháp chia để trị: Tìm max min

Yêu cầu xây dựng chương trình

- **Sinh dãy số nguyên**
- **Tìm max_min theo phương pháp duyệt tuần tự**
- **Tìm max_min theo phương pháp chia để trị**
- **So sánh thời gian**

```
int main()
{
    int *a, n;
    long t1, t2;
    sinh(a,n);  //in(a,n);
    t1= clock();
    MAXMIN(a, n);
    t2= clock();
    cout<<"\n thoi gian chay thuat toan 1=" <<t2-t1;
}
```

Phương pháp chia để trị: Tính X^n

Ví dụ: Tính X^n

Thuật toán lặp

```
Kq= 1;
for (int i=1; i<=n; i++ ) Kq= Kq*X;
return Kq;
```

tinh(5,7);

Phương pháp chia để trị: Tính X^n

```
public class CSPPE_MyP07_Tinh_X_MU_N {
   long xmun(int x, int n)    {
      long kq=1;
      for (int i=1; i<=n; i++)  kq=kq*x;
      return kq;
   }
   public static void main(String[] args) {
      CSPPE_MyP07_Tinh_X_MU_N m= new CSPPE_MyP07_Tinh_X_MU_N();
      int n=5;
      System.out.println(" 20^" + n+ "= "+ m.xmun(20, n));
  }
}
```

Phương pháp chia để trị: Tính X^n

```
public class CSPPE_MyP07_Tinh_X_MU_N {
   long xmun(int x, int n)    {
      long kq=1;
      for (int i=1; i<=n; i++) kq=kq*x;
      return kq;
   }
   public static void main(String[] args) {
      CSPPE_MyP07_Tinh_X_MU_N m= new CSPPE_MyP07_Tinh_X_MU_N();
      int n=50;
      System.out.println(" 20^" + n+ "= "+ m.xmun(20, n));
  }
}
```

Nếu thay n=50, kết quả như thế nào?

Phương pháp chia để trị: Tính X^n

Ví dụ: Tính X^n

```
BigInteger X_MU_N(BigInteger x, int n)
  {
     if(n==0) return BigInteger.ONE;
     else return x.multiply(X_MU_N(x, n-1));
  }
```

Phương pháp chia để trị: Tính X^n

```
import java.math.BigInteger;
public static void main(String[] args) {
    CSPPE_MyP07_Tinh_X_MU_N m= new
CSPPE_MyP07_Tinh_X_MU_N();
      int n=50;
      BigInteger x = new BigInteger("20");
      System.out.println(x+ "^" + n+ "= "+ m.X_MU_N(x, n));
  }
}
```

Phương pháp chia để trị: Tính X^n

Phân tích X^n = X*X* X * X* X*……..* X

X^{20} = [X * X * X* X* X* X* X* X* X* X]* [X* X* X* X* X* X* X* X* X* X]

X^{10000} = [X * X * X* * * X* X* X]*[X* X* X* ** *X*X* X]

X^{11} = X * X^{10}

Nhận xét:

Nếu n lẻ thì ta tính $X^n = X * X^{n-1}$

Nếu n chẵn: $X^n = X^{n/2} * X^{n/2}$

Phương pháp chia để trị: Tính X^n

Nhận xét:

Nếu n lẻ thì ta tính $X^n = X * X^{n-1}$

Nếu n chẵn: $X^n = X^{n/2} * X^{n/2}$

Do $X^{n/2}$ giống nhau nên ta chỉ tính 1 lần rồi thực hiện thêm 1 phép nhân, do đó giảm được (n/2) -1 phép tính nhân.

Độ phức tạp khi tính toán theo cách này là logn

Ví dụ n=9 thì cần X^9 = X* X^8 $X^8 = X^4 * X^4$

$X^4 = X^2 * X^2$; $X^2 = X*X$

Số phép nhân = 1 (khi tính X^2) + 1 (khi tính X^4) + 1 (khi tính X^8) + 1 (khi tính X^9) = 4

Phương pháp chia để trị: Tính X^n

Ví dụ: Tính X^n

Phân tích X^n = X*X* X * X* X*…….* X

X^{20} = X * X * X* X* X* X* X* X* X* X* X* X* X* X* X* X* X* X* X* X

X^{10000} = X * X * X* X* * * X* X* X* X* X* X* X* X* X* X* X* X * ** X* X

Cần bao nhiêu phép nhân khi tính X^{100} ???
Cần bao nhiêu phép nhân khi tính X^{10000} ???

Phương pháp chia để trị: Tính X^n

```
BigInteger X_MU_N1(BigInteger x, int n)   {
      if(n==0) return BigInteger.ONE;
      else
         if(n%2!=0) return x.multiply(X_MU_N1(x, n-1));
         else
         {
            BigInteger kq= X_MU_N1( x, n/2);
            return kq.multiply(kq);
         }
   }
```

```
long long xmun(int x, int n)
{
    if(n==0) return 1;
    else if(n==1) return x;
    else
        if (n&1)
            return x* xmun(x, n-1);
        else {
          long long kq= xmun(x, n/2);
          return kq*kq;
        }
}
```

Phương pháp chia để trị

```
public static void main(String[] args) {
      CSPPE_MyP07_Tinh_X_MU_N m= new CSPPE_MyP07_Tinh_X_MU_N();
      int n=5200; long t1,t2;
     BigInteger x = new BigInteger("2000");      BigInteger y= new BigInteger("0");
    t1=System.currentTimeMillis();
      for (int i=1; i<100; i++)       y=m.X_MU_N1(x, n);
    t2=System.currentTimeMillis();
   System.out.println("\n Thoi gian chay la: " + (t2-t1));
}
```

Phương pháp chia để trị (divide and conquer)

```
public static void main(String[] args) {
      CSPPE_MyP07_Tinh_X_MU_N m= new CSPPE_MyP07_Tinh_X_MU_N();
      int n=9200;
      //System.out.println(" 2^" + n+ "= "+ m.xmun(2, n));
      BigInteger x = new BigInteger("2");
      BigInteger y= new BigInteger("0");
       for (int i=1; i<1000; i++)
          y=m.X_MU_N1(x, n);
      //System.out.println(x+ "^" + n+ "= "+ m.X_MU_N(x, n));
   }
```

Cho n= 9200 và x=2, so sánh thời gian 2 cách tính???

Phương pháp chia để trị (divide and conquer)

```
public static void main(String[] args) {
        CSPPE_MyP07_Tinh_X_MU_N m= new CSPPE_MyP07_Tinh_X_MU_N();
        int n=9200;
        //System.out.println(" 2^" + n+ "= "+ m.xmun(2, n));
        BigInteger x = new BigInteger("2");
        BigInteger y= new BigInteger("0");
         for (int i=1; i<1000; i++)
            y=m.X_MU_N1(x, n);
        //System.out.println(x+ "^" + n+ "= "+ m.X_MU_N(x, n));
    }
```

Cho n= 9200 và x=2, so sánh thời gian 2 cách tính

Phương pháp chia để trị: Tính X^n

Ví dụ kết quả chạy:

```
 X= 4000
 X ^ n =
41495155688809929585124078636911611510124462322424368999956573296906528114129081463997070489471
03794288197886611300789182395151075411775307886874834113963687061181803401509523685376000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000Process returned 0 (0x0)   execution time : 2.661 s
Press any key to continue.
```

Tính X^n theo phương pháp lặp

Tính X^n theo pp lặp

```
s=1;
while n>0 {
  nếu n lẻ thì s=s*x;
  n=n/2;
  if(n>0) x=x*x;
}
return ;
```

Thử với 2^{10}

Chạy	S	n	x
0	1	10	2
1	1	5	4
2	1*4	2	16
3	4	1	256
4	1024	0	
5	Dừng vì n=0		

Tính X^n theo phương pháp lặp

Tính X^n theo pp lặp

```
long long xmun1(int x, int n) {
    long long s=1;
    while (n>0) {
       if(n&1) s=s*x;
       n=n>>1;  if(n>0) x=x*x;
      }
  return s;
}
```

Chạy	S	n	x

Thử với 3^{13}???

Tính X^n theo phương pháp lặp

```
int main(){
        int n=30, x=3;
        long long s,t1,t2;
        t1=clock();
        for (int i=1; i<=100000; i++) s= xmun1(x, n);
        t2=clock();
        cout<<"\n "<<x<< "^"<<n<<" = "<<s;
        cout<<"\n thoi gian tinh toan la = "<<(t2-t1);
}
```

Thuật toán nhân Ấn Độ

Thử S=7*23

Chỉ thực hiện các phép nhân 2 và chia 2 khi tính S=a*b

```
S=0;
while b>0 {
  nếu b lẻ thì S=S+ a;
  b=b/2;
  if(b>0) a=a+a;
}
return S;
```

Chạy	S	a	b
0	0	7	23
1	7	14	11
2			
3			

Hãy điền bảng kết quả???

Phương pháp chia để trị: Ternary Search

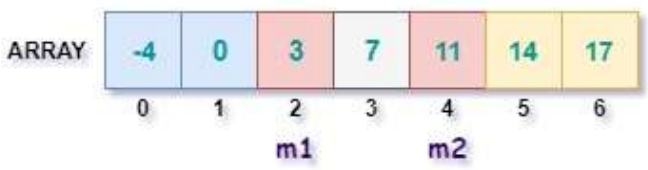
Ternary search is a divide and conquer algorithm that can be used to find an element in an array. It is similar to binary search. In this algorithm, we divide the given array into three parts and determine which has the key. Initially, l and r will be equal to 0 and n-1 respectively, where n is the length of the array.

mid1 = l + (r-l)/3
mid2 = r – (r-l)/3

Phương pháp chia để trị: Ternary Search

Ternary search is a divide and conquer algorithm that can be used to find an element in an array. It is similar to binary search. In this algorithm, we divide the given array into three parts and determine which has the key. Initially, l and r will be equal to 0 and n-1 respectively, where n is the length of the array.

mid1 = l + (r-l)/3
mid2 = r – (r-l)/3



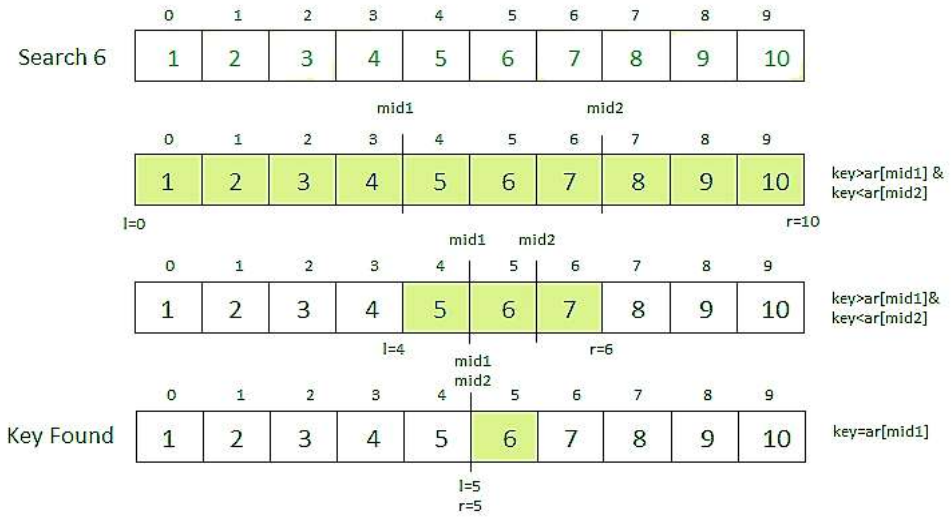
Phương pháp chia để trị: Ternary Search algorithm

```
int Ternary_search(int *a, int x, int L, int R){
        if(L>R) return -1;
        else    {
                int m1= L+ (R-L)/3; int m2= R- (R-L)/3;
                if(x==a[m1]return m1; else if(x==a[m2] return m2;
                else if (x<a[m1]) return Ternaty_search(a,x, L, m1-1);
                else if(x>a[m2]) return Ternaty_search(a,x, m2+1, R);
                else return Ternaty_search(a,x, m1+1, m2-1);
                }
}
```

Phương pháp chia để trị: Ternary Search algorithm

```
int Ternary_search(int *a, int x, int L, int R){
        if(L>R) return -1;
        else    {
                int m1= L+ (R-L)/3; int m2= R- (R-L)/3;
                if(x==a[m1]return m1; else if(x==a[m2] return m2;
                else if (x<a[m1]) return Ternaty_search(a,x, L, m1-1);
                else if(x>a[m2]) return Ternaty_search(a,x, m2+1, R);
                else return Ternaty_search(a,x, m1+1, m2-1);
                }
}
```

Phương pháp chia để trị: Ternary Search



Thuật toán nhân Karatsuba

The Karatsuba algorithm is a fast multiplication algorithm. It was discovered by Anatoly Karatsuba in 1960 and published in 1962.

It reduces the multiplication of two n-digit numbers to at most $n^{(\log_2 3)}$. It is therefore faster than the classical algorithm, which n^2 single-digit products.

Anatoly Alexeyevich Karatsuba
(31/1/1937 - 28/9/2008)

1.584963

Thuật toán nhân Karatsuba

Ví dụ:

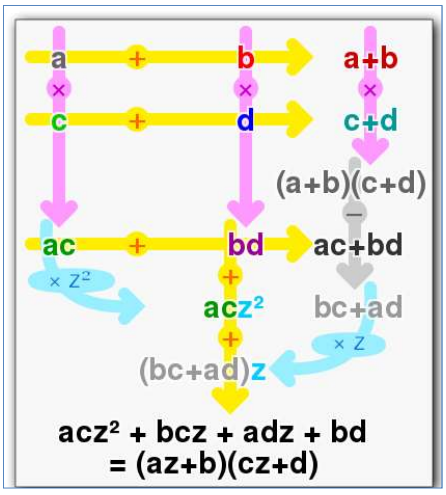
Để nhân hai số có 1024 chữ số, thuật toán nhân Karatsuba cần 3^{10} = 59,049 phép nhân các số đơn (single-digit multiplications) trong khi đó phép nhân truyền thống cần $(2^{10})^2$ = **1,048,576** phép nhân.

Như vậy nó nhanh hơn 17.75 lần.

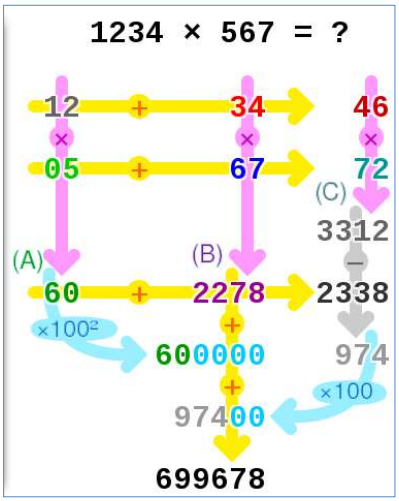
Anatoly Alexeyevich Karatsuba
(31/1/1937 - 28/9/2008)

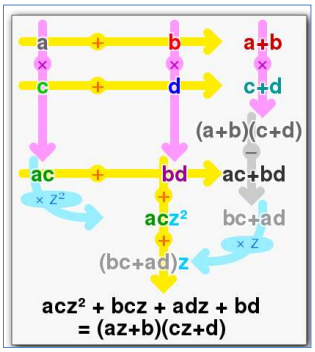
Thuật toán nhân 2 số lớn có độ dài n

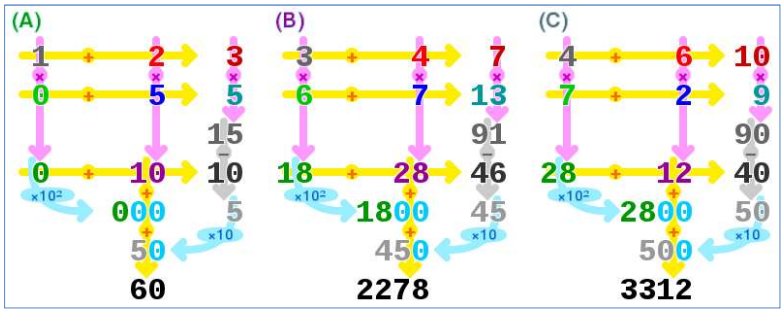
The basic step of Karatsuba's algorithm is a formula that allows one to compute the product of two large x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts.



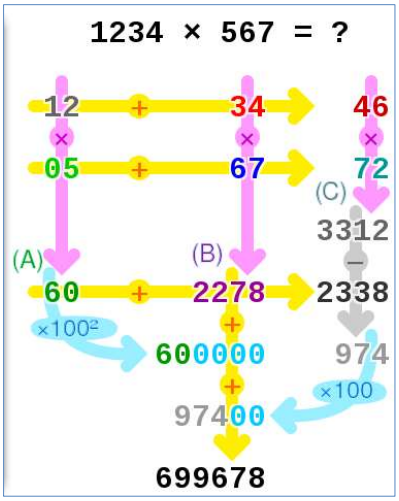
Thuật toán nhân 2 số lớn có độ dài n







Thuật toán nhân 2 số lớn có độ dài n



Thử tính 2345*9876???

Thuật toán nhân 2 số lớn có độ dài n

$$x = x_1 B^m + x_0,$$
$$y = y_1 B^m + y_0,$$

$$xy = (x_1 B^m + x_0)(y_1 B^m + y_0)$$
$$= z_2 B^{2m} + z_1 B^m + z_0,$$

x_0 and y_0 are less than B^m

$$z_2 = x_1 y_1,$$
$$z_1 = x_1 y_0 + x_0 y_1,$$
$$z_0 = x_0 y_0.$$

Charles Babbage.

$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0.$$

$$z_1 = (x_0 - x_1)(y_1 - y_0) + z_2 + z_0.$$

Karatsuba

produce a result in the range $0 \leq \text{result} < 2B^m$),

Thuật toán nhân 2 số lớn có độ dài n

```
procedure karatsuba(num1, num2)
    if (num1 < 10) or (num2 < 10)        return num1 × num2
    m = min(size_base10(num1), size_base10(num2)); /* Calculates the size */
    m2 = floor(m / 2)     /*m2 = ceil(m / 2) will also work */
     /* Split the digit sequences in the middle. */
    high1, low1 = split_at(num1, m2);    high2, low2 = split_at(num2, m2)
     /* 3 calls made to numbers approximately half the size. */
    z0 = karatsuba(low1, low2);    z1 = karatsuba((low1 + high1), (low2 + high2))
    z2 = karatsuba(high1, high2)
    return (z2 × 10 ^ (m2 × 2)) + ((z1 - z2 - z0) × 10 ^ m2) + z0
```

Thảo luận nhóm: Problem 1. (50 points)

Consider two sums, $X = x_1 + x_2 + \ldots + x_n$ and $Y = y_1 + y_2 + \ldots + y_m$.
Give an algorithm that finds indices i and j such that swapping x_i with y_j makes the two sums equal, that is, $X - x_i + y_j = Y - y_j + x_i$, if they exist.

X={ 1, 2,7, 9} Y= { 1, 2} ➔ i=4, j=1

X= { 20, 10} Y= { 1,2, 3} → Không có i và j thoả điều kiện

Thảo luận nhóm về thuật toán của nhóm để giải bài toán 1!!!

Thảo luận nhóm: Problem 2. (50 points)

A game-board has n columns, each consisting of a top number, the cost of visiting the column, and a bottom number, the maximum number of columns you are allowed to jump to the right. The top number can be any positive integer, while the bottom number is either 1, 2, or 3. The objective is to travel from the first column off the board, to the right of the nth column.

Thảo luận nhóm: Problem 2. (50 points) (tt)

The cost of a game is the sum of the costs of the visited columns. Assuming the board is represented in a twodimensional array, B[2, n], the following recursive procedure computes the cost of the cheapest game:

```
int CHEAPEST(int i)
  if i > n then return 0 endif;
  x = B[1,i] + CHEAPEST(i+1);
  y = B[1,i] + CHEAPEST(i+2);
  z = B[1,i] + CHEAPEST(i+3);
  case B[2,i] = 1: return x;
       B[2,i] = 2: return min{x,y};
       B[2,i] = 3: return min{x,y,z}
  endcase.
```

Thảo luận nhóm trò chơi ở bài 2!!! Hãy Tìm giải pháp tốt hơn

Tài liệu đọc thêm

Bernstein, D. J., "Multidigit multiplication for mathematicians"

Link YouTube

https://www.youtube.com/watch?v=OtzDFLnIREc

https://www.youtube.com/watch?v=rg517uhXV1o