* Mô hình thuật toán chia để trị
* Giới thiệu thuật toán chia để trị : dùng để giải các lớp bài toán có thể thực hiện được 3 bước:

1. Devide (chia). Chia bài toán lớn thành những bài toán con có cùng kiểu với bài toán lớn.

2. Conquer(Trị).Giải các bài toán con.Thông thường các bài toán con chỉ khác nhau về dữ liệu vào nên ta có thể thực hiện bằng một thủ tục quay lui.

3.Conbine (Tổng hợp). Tổng hợp lại kết quả của các bài toán con để nhận được kết quả của bài toán lớn.

* Ví dụ: Tính x^n mod 10e9+7
* Ta có x^n trong trường hợp x chẵn, x.x^n/2 trong trường hợp x lẻ.
* Kết quả chính là x^n/2 \* x^n/2 trong trường hợp x chẵn và x\*x^n/2\*x^n/2 trong trường hợp x lẻ.
* Chương trình nâng x lên lũy thừa n bằng phương pháp chia để trị
* #include<bits/stdc++.h>
* using namespace std;
* int powmod(int x, int n){
* if(n == 0) return 1;
* else if(n % 2 == 0) {
* return powmod(x,n/2) \* powmod(x,n/2);
* }
* else return x \* powmod(x,n/2) \* powmod(x,n/2);
* }
* int main(){
* int test = 1;
* // cin >> test;
* while(test--){
* int x ,n;
* cin >> x >> n;
* cout << powmod(x, n) << endl;
* }
* return 0;
* }
* Thuật toán tìm kiếm nhị phân
* Thuật toán tìm kiếm nhị phân là phương pháp định vị phần tử x trong một danh sách A[] gồm n phần tử đã được sắp xếp. Quá trình tìm kiếm bắt đầu việc chia danh sách thành 2 phần.Sau đó, so sánh x với phần tử x ở giữa. Khi đó có 3 trường hợp có thể xảy ra:
* **Trường hợp 1:** Nếu x bằng phần tử ở giữa A[mid], thì mid chính là vị trí của x trong danh sách A[].
* **Trường hợp 2:** Nếu x lớn hơn phần tử ở giữa thì nếu x có mặt trong dãy A[] thì ta chỉ cần tìm các phần tử từ mid+1 đến vị trí thứ n.
* **Trường hợp 3:** Nếu x nhỏ hơn A[mid] thì x chỉ có thể dãy con bên trái của dãy A[].

Lặp lại quá trình cho đến khi cận dưới vượt cận trên của dãy A[] mà vẫn chưa tìm thấy x thì ta kết luận x không có mặt trong dãy A[].

Thuật toán:

int binary\_search(int A[],int n, int x){

int low = 0;

int hight = n – 1;

int mid =(low + hight) / 2;

while( low <= hight ){ // lặp trong khi cận dưới vẫn nhỏ hơn cận trên

if(x > A[mid]) // nếu x lớn hơn phần tử ở giữa

low = mid + 1;// cận dưới được đặt lên vị trí mid + 1

else if(x < A[i])

hight = mid – 1; // cận lùi về vị trí mid-1

else

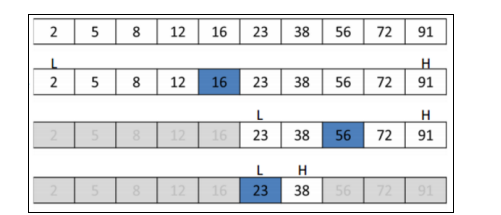
return mid; // đây là vị trí của x

mid = (low + hight) / 2; // xác định phần tử ở giữa

}

Return -1

* **Độ phức tạp thuật toán**
* Độ phức tạp thuật toán là O(log(n)), với n là số lượng phần tử của dãy A[].
* **Kiểm nghiệm thuật toán**
* Ví dụ ta cần tìm x = 23 trong dãy A[] = {2, 5, 8, 12, 16, 23, 38, 56,72,91}. Khi đó



**#include<bits/stdc++.h>**

**using namespace std;**

**int binary\_seach(int A[100], int n, int x){**

**int low = 0;**

**int hight = n - 1;**

**int mid = (low + hight ) / 2;**

**while(low <= hight){**

**if(x > A[mid]) low = mid + 1;**

**else if(x < A[mid]) hight = mid - 1;**

**else return mid;**

**mid = (low + hight) / 2;**

**}**

**return -1;**

**}**

**int main(){**

**int test = 1;**

**// cin >> test;**

**while(test--){**

**int A[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};**

**int x = 91, n = sizeof(A)/sizeof(A[0]);**

**int result = binary\_seach(A, n, x);**

**if(result >= 0 ) cout << "\nVi tri cua x:" << result;**

**else cout << "\n khong tim thay x";**

**}**

**return 0;**

**}**

**#include<iostream>**

**#include<iomanip>**

**using namespace std;**

**void merge(int arr[], int l, int m, int r){//thuật toán hợp nhất hai đoạn đã sắp xếp**

**int i, j, k;**

Thuật toán Merge-Sort:

Input :

• Dãy số : Arr[];

• Cận dưới: l;

• Cận trên m;

Output:

• Dãy số Arr[] được sắp theo thứ tự tăng dần.

Formats: Merge-Sort(Arr, l, r);

Actions:

if ( l< r ) {

m = (l + r -1) / 2; //phép chia Arr[] thành hai nửa

Merge-Sort(Arr, l, m);//trị nửa thứ nhất

Merge-Sort(Arr, m+1, r); //trị nửa thứ hai

Merge(Arr, l, m, r); //hợp nhất hai nửa đã sắp xếp

}

End.

Kiểm nghiệm thuật toán Merge-Sort: Merge-

Sort(Arr,0,7)

Input : Arr[] = {38, 27, 43, 3, 9, 82, 10}; n = 7;

int n1 = m - l + 1; //số lượng phần tử đoạn 1

int n2 = r - m; //số lượng phần tử đoạn 3

int L[n1], R[n2]; //tạo hai mảng phụ để lưu hai đoạn được sắp

for(i = 0; i < n1; i++)//lưu đoạn thứ nhất vào L[]

L[i] = arr[l + i];

for(j = 0; j < n2; j++)//lưu đoạn thứ hai vào R[]

R[j] = arr[m + 1+ j];

i = 0; j = 0; k = l; //bắt đầu hợp nhất

while (i < n1 && j < n2){ //quá trình hợp nhất

if (L[i] <= R[j]){

arr[k] = L[i]; i++;

}

else {

arr[k] = R[j]; j++;

}

k++;

}

while (i < n1) { //lấy các phần tử còn lại trong L[] vào arr[]

arr[k] = L[i];

i++; k++;

}

while (j < n2){ //lấy các phần tử còn lại trong R[] vào arr[]

arr[k] = R[j];

j++; k++;

}

}

void mergeSort(int arr[], int l, int r){ //thuật toán Merge Sort

if (l < r){ //nếu cận dưới còn bé hơn cận trên

int m = l+(r-l)/2; //tìm vị trí ở giữa đoán l, r

mergeSort(arr, l, m); //trị nửa thứ nhất

mergeSort(arr, m+1, r); //trị nửa thứ hai

merge(arr, l, m, r); //hợp nhất hai đoạn đã được sắp

}

}

void printArray(int Arr[], int size){ //in kết quả

int i;cout<<"\n Dãy được sắp:";

for (i=0; i < size; i++)

cout<<Arr[i]<<setw(3);

}

int main(){

int arr[] = {38, 27, 43, 3, 9, 82, 10};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

mergeSort(arr, 0, arr\_size - 1);

printArray(arr, arr\_size);

}