1. **Sắp xếp chọn:**

void SelectionSort(int \*&a){

int count = 0;

while (1){

int LonggestDistance = 0;// độ chênh lệch bao giờ cũng là 0 ở thời điểm bân đầu

//vì ta cần so sánh a[đang xét] với một con a[x] nào đó nhỏ hơn nó và a[x] là min của dãy ở sau.

int indexCurrent = count;// biến vị trí hiện tại. đc khởi tạo là tại ví trí đang xét

for (int i = count + 1; i < MAX; i++){

if (a[count]>a[i] && a[count] - a[i]>LonggestDistance){// chốt: a[đang xét]>a[i] và độ chênh lệch của hai thằng phải lớn hơn độ chênh lệch ban đầu khởi tạo

LonggestDistance = a[count] - a[i];// biến đọ dài chênh lệch

indexCurrent = i;// lưu vị trí a[i] khi thỏa điều kiện ban đầu.

}

}

{//1

int temp = a[count];

a[count] = a[indexCurrent];

a[indexCurrent] = temp;

}//2

// từ chổ 1 và 2 là hàm swap á, tại tao lười nên viết ngay vô luôn.

count++;// tăng đếm lên 1 đơn vị

if (count > MAX - 1)break;// điều kiện để thóat lặp

}

}

1. **Sắp xếp nổi bọt**

void BubbleSort(int\* &a){

int count = MAX;

while (count--){

for (int i = 0; i < count; i++){

if (a[i]>a[i + 1]){

int temp = a[i+1];

a[i+1] = a[i];

a[i] = temp;

}

}

if (count == -1) return;

}

}

1. **Sắp xếp chèn**

void InsertAnElement(int\* &a,int index){

for (int i = 0; i < index; i++){

if (a[i] > a[index]){

int temp = a[index];

for (int j = index; j>i; j--){

a[j] = a[j - 1];

}

a[i] = temp;

return;

}

}

}

void InsertionSort(int\* &a){

int count = 1;

while (1){

InsertAnElement(a, count);

count++;

if (count == MAX)break;

}

}

1. **Sắp xếp vung đống:**

void Push\_Down(int \*&a,int index,int limit){

int check = 0;

while (index <= limit / 2 && check != 1){

int son;

if (index == limit / 2 && index \* 2 == limit) son = 2 \* index;

else{

(a[2 \* index] < a[2 \* index + 1]) ? son = 2 \* index : son = 2 \* index + 1;

}

if (a[index]>a[son]){

swap(a[index], a[son]);

index = son;

}

else{

check = 1;

}

}

}

void HeapSort(int\* &a){

int n = MAX;

for (int height = MAX / 2; height > 0; height--){

Push\_Down(a, height,n);

}

for (int k = n; k > 2; k--){

for (int l = k / 2; l > 0; l--){

Push\_Down(a, l, k);

}

}

}

1. **Sắp xếp nhanh:**

void QuickSort(int\* &a, int left, int right){

int i = left + 1;// cho left qua 1 đơn vị ý là chọn a[left] làm chốt

int j = right;

do{

while (a[i] < a[left] && i < right)i++;//tìm ptu lớn hơn hay bằng với chốt

while (a[j]>a[left])j--;// tìm ptu nhỏ hơn hay bằng với chốt

if (i < j){

swap(a[i], a[j]);// sau khi tìm dc hai ptu thì hoán vị

}

} while (i<j);

swap(a[left], a[j]);// chạy mộtt hơi tới khi mà j<i thì hoán đổi chốt với a[j]

if (left<j-1){

QuickSort(a, left, j - 1); //gọi đệ quy bên trái

}

if(j+1<right){

QuickSort(a, j + 1, right);// gội đệ quy bên phải

}

}

1. **Sắp xếp Theo cơ số Radix Sort**

struct node{//khai báo struct node

int info;

node\* pNext;

};

struct Queue{

node\* Head;

node\* Tail;

}Array[10];// tạo mảng hàng đợi và khai báo mảng các hàng đợi gồm 10 hàng đợi

node\* getNode(int x){// hàm tạo node

node\* p = (node\*)malloc(sizeof(node));

if (p == NULL)return NULL;

p->info = x;

p->pNext = NULL;

return p;

}

void getQueue(Queue &q){// khởi tạo một hàng đợi

q.Head = NULL;

q.Tail = NULL;

}

void InsLast(Queue &q, int x){// chèn vào cuối

node\*p = getNode(x);

if (q.Head == NULL){

q.Head = q.Tail = p;

}

else{

q.Tail->pNext = p;

q.Tail = p;

}

}

int DelFirst(Queue &q){// hàm này gồm hai mục đích

if (q.Head == NULL){

q.Tail = NULL;

return 0;

}

else{

node\* p = q.Head;// 1. xóa node đầu của hàng đợi

q.Head = q.Head->pNext;

int temp = p->info;

free(p);

return temp;//2. lưu lại giá trị của node đầu để sau này cho vô mảng lại

}

}

void printArray(int\* a){// hàm in thôi!

int count = 0;

for (int i = 0; i < MAX; i++){

count++;

printf("%d%c", a[i], (count % 10 == 0) ? '\n' : '\t');

}

printf("\n--------------------------------------------------------------------------------------------------\n");

}

void Random(int\* &a){// hàm tạo ngẫu nhiên số

for (int i = 0; i < MAX; i++){

a[i] = rand() % 9000 + 10;// 10<=a[i]<=9009

}

}

bool CheckStoping(){// hàm này quan trọng nè: kiểm tra xem điều kiện dừng việc gửi các giá trị node vào hàng đợi tại vì ta không có cơ sở để xác định liệu mảng đã dc sắp xếp xong chưa

for (int i = 1; i < 10; i++){

if (Array[i].Head != NULL) return false;// khi cập nhật vô hàng đợi thì đến một lúc nào đó các giá

}//trị sẽ dồn hết về Array[0] nên ta cần kiểm tra khúc sau xem có ai còn phần tử không?

return true;

}

void RadixSort(int\* &a){

for (int i = 0; i < 10; i++){

getQueue(Array[i]);

}// khởi tạo mảng 10 hàng đợi

int CoSo = 1;// cơ số đầu tiên là hàng đơn vị.

while (1){

for (int i = 0; i < MAX; i++){

int temp = (a[i] % (int)pow(float(10), float(CoSo)))/ pow(float(10), float(CoSo - 1));// công thức tính cơ số ở vị trí bất kì

InsLast(Array[temp], a[i]);// chèn giá trị a[i] vô hàng đợi tại cơ số đag xét, không đá động gì tới mảng a[]

}

int count = 0;// biến đếm để cập nhật lại giá trị mới sau các sắp xếp vô mảng hàng đợi

for (int i = 0; i < 10; i++){

if (CheckStoping()==true){// kiểm tra điều kiện nếu điều kiện đúng thì chép toàn bộ node của hàng đợi đầu vào mảng và thoát ra ngay lập tức

while (Array[i].Head!=NULL){

a[count]=DelFirst(Array[i]);

count++;

}

return;// sau vòng lặp là thoát ra chỗ này nè!

}

else{// nếu không vô điều kiện thì vẫn chép vô mảng rồi cứ tiếp tục quá trình với Cơ số tăng thêm 1 đơn vị

while (Array[i].Head!=NULL){

a[count] = DelFirst(Array[i]);

count++;

}

}

}

CoSo++;// cơ số tăng thêm 1 đơn vị

}

}

1. **Sắp xếp dựa trên giá trị của chính Key (Counting Sort)**

void CreatingZeroForEmptyElement(int\* &a,int from,int to){

for (int i = from; i < to+1;i++){

a[i] = 0;

}

}

int\* CountingSort(int\* a){

int MaxNum;

int \*arrCounter = (int\*)calloc((a[0]+1), sizeof(int));

int count = a[0] + 1;

arrCounter[count-1]++;

for (int i = 1; i < MAX; i++){

if (a[i]<count - 1){

arrCounter[a[i]]++;

}

else{

arrCounter = (int\*)realloc(arrCounter, (a[i] + 1)\*sizeof(int));

CreatingZeroForEmptyElement(arrCounter, count, a[i] + 1);

count = a[i] + 1;

arrCounter[count - 1]++;

}

}

int\* arrDesk = (int\*)malloc(MAX\*sizeof(int));

int index = 0;

for (int i = 0; i < count+1; i++){

//printf("%d ", arrCounter[i]);

if (arrCounter[i] != 0){

while (arrCounter[i] != 0){

arrDesk[index] = i;

arrCounter[i]--;

index++;

}

}

}

//printArray(arrDesk);

if (arrCounter == NULL) free(arrCounter);

return arrDesk;

}

1. **Sắp xếp Shell**

void InsertAnElement(int\* &a, int index,int par){

for (int i = 0; i < index; i+=par){

if (a[i] > a[index]){

int temp = a[index];

for (int j = index; j>i; j-=par){

a[j] = a[j - par];

}

a[i] = temp;

return;

}

}

}

void ShellSort(int\* &a,int par){

if (par == 0)return;

int count = 0;

while (1){

InsertAnElement(a, count,par);

count+=par;

if (count >= MAX)break;

}

ShellSort(a,par/2);

}