1. **Cho mảng gồm các phần tử { 2, 3, 4, 10, 40 } nhập vào từ bàn phím.**

**Viết chương trình tìm kiếm giá trị 10. Sử dụng cài đặt tìm kiếm nhị phân để tìm kiếm.**

#include <stdio.h>

// Hàm tìm kiếm nhị phân

int binarySearch(int arr[], int left, int right, int x) {

while (left <= right) {

int mid = left + (right - left) / 2;

// Nếu phần tử ở giữa mảng là giá trị cần tìm

if (arr[mid] == x) {

return mid; }

// Nếu giá trị cần tìm nhỏ hơn phần tử ở giữa mảng

if (arr[mid] > x) {

right = mid - 1; }

// Nếu giá trị cần tìm lớn hơn phần tử ở giữa mảng

else {

left = mid + 1; } }

// Nếu không tìm thấy giá trị cần tìm

return -1;}

int main() {

int arr[] = {2, 3, 4, 10, 40};

int n = sizeof(arr) / sizeof(arr[0]);

int x = 10; // Giá trị cần tìm

int result = binarySearch(arr, 0, n - 1, x);

if (result == -1) {

printf("Khong tim thay gia tri %d trong mang.\n", x); } else {

printf("Gia tri %d duoc tim thay tai vi tri %d trong mang.\n", x, result); }

return 0; }

1. **Cho mảng gồm các phần tử { 2, 3, 4, 10, 40 } nhập vào từ bàn phím.**

**Viết chương trình tìm kiếm giá trị 10. Sử dụng cài đặt tìm kiếm tuyến tính để tìm kiếm.**

**#include <stdio.h>**

// Hàm tìm kiếm tuyến tính

int linearSearch(int arr[], int n, int x) {

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

if (arr[i] == x) {

return i; }}

// Nếu không tìm thấy giá trị cần tìm

return -1;}

int main() {

int arr[] = {2, 3, 4, 10, 40};

int n = sizeof(arr) / sizeof(arr[0]);

int x = 10; // Giá trị cần tìm

int result = linearSearch(arr, n, x);

if (result == -1) {

printf("Khong tim thay gia tri %d trong mang.\n", x); } else {

printf("Gia tri %d duoc tim thay tai vi tri %d trong mang.\n", x, result); }

return 0;}

1. **Cho mảng gồm các phần tử { 41, 23, 4, 14, 56, 1 } nhập vào từ bàn phím.**

**Viết chương trình để sắp xếp. Sử dụng phương pháp sắp xếp đổi chỗ trực tiếp để sắp xếp**

#include <stdio.h>

// Hàm sắp xếp mảng bằng phương pháp bubble sort

void bubbleSort(int arr[], int n) {

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

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Đổi chỗ hai phần tử nếu thứ tự không đúng

int temp = arr[j];

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

arr[j + 1] = temp; } } }}

int main() {

int arr[] = {41, 23, 4, 14, 56, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Mang truoc khi sap xep: ");

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

printf("%d ", arr[i]); }

printf("\n");

bubbleSort(arr, n);

printf("Mang sau khi sap xep: ");

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

printf("%d ", arr[i]);}

printf("\n");

return 0;}

1. **Cho mảng gồm các phần tử { 41, 23, 4, 14, 56, 1 } nhập vào từ bàn phím.**

**Viết chương trình để sắp xếp. Sử dụng phương pháp sắp xếp chọn trực tiếp để sắp xếp**

#include <stdio.h>

// Hàm sắp xếp mảng bằng phương pháp selection sort

void selectionSort(int arr[], int n) {

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

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

// Tìm phần tử nhỏ nhất trong đoạn chưa được sắp xếp

minIndex = j; } }

// Đổi chỗ phần tử nhỏ nhất tìm được với phần tử đầu đoạn chưa được sắp xếp

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp; }}

int main() {

int arr[] = {41, 23, 4, 14, 56, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Mang truoc khi sap xep: ");

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

printf("%d ", arr[i]); }

printf("\n");

selectionSort(arr, n);

printf("Mang sau khi sap xep: ");

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

printf("%d ", arr[i]); }

printf("\n");

return 0;}

1. **Cho mảng gồm các phần tử { 41, 23, 4, 14, 56, 1 } nhập vào từ bàn phím.**

**Viết chương trình để sắp xếp. Sử dụng phương pháp sắp xếp chèn trực tiếp để sắp xếp**

**#include <stdio.h>**

// Hàm sắp xếp mảng bằng phương pháp insertion sort

void insertionSort(int arr[], int n) {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

// Di chuyển các phần tử lớn hơn key sang phải một vị trí

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

j--;

}

// Đặt key vào đúng vị trí

arr[j + 1] = key;

}

}

int main() {

int arr[] = {41, 23, 4, 14, 56, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Mang truoc khi sap xep: ");

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

printf("%d ", arr[i]);

}

printf("\n");

insertionSort(arr, n);

printf("Mang sau khi sap xep: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

1. **Cho mảng gồm các phần tử { 41, 23, 4, 14, 56, 1 } nhập vào từ bàn phím. Viết chương trình để sắp xếp. Sử dụng phương pháp sắp xếp nhanh để sắp xếp.**

#include <stdio.h>

// Hàm đổi chỗ giá trị của hai phần tử

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Hàm chia mảng thành 2 phần, trả về chỉ số chốt (pivot)

int partition(int arr[], int low, int high) {

int pivot = arr[high]; // Chọn phần tử cuối làm chốt

int i = low - 1; // Chỉ số của phần tử nhỏ hơn chốt

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]); // Đưa chốt về vị trí đúng

return (i + 1);

}

// Hàm sắp xếp mảng theo phương pháp Quick Sort

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pivot = partition(arr, low, high); // Chia mảng thành 2 phần

quickSort(arr, low, pivot - 1); // Đệ quy sắp xếp nửa trái của mảng

quickSort(arr, pivot + 1, high); // Đệ quy sắp xếp nửa phải của mảng

}

}

int main() {

int arr[] = {41, 23, 4, 14, 56, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Mang ban dau: ");

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

printf("%d ", arr[i]);

}

quickSort(arr, 0, n - 1); // Gọi hàm sắp xếp Quick Sort

printf("\nMang sau khi sap xep: ");

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

printf("%d ", arr[i]);

}

return 0;

}

1. **Cho mảng gồm các phần tử { 41, 23, 4, 14, 56, 1 } nhập vào từ bàn phím.**

**Viết chương trình để sắp xếp. Sử dụng phương pháp sắp xếp cây để sắp xếp**

#include <stdio.h>

// Hàm hoán đổi giá trị của hai biến

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Hàm tạo một cây nhị phân lớn nhất từ một nút gốc cho trước

void heapify(int arr[], int n, int i) {

int largest = i; // Khởi tạo nút lớn nhất là nút gốc

int left = 2 \* i + 1; // Chỉ số của nút con trái

int right = 2 \* i + 2; // Chỉ số của nút con phải

// So sánh nút gốc với nút con trái và nút con phải

// Nếu nút con lớn hơn nút gốc, thì đổi chỗ nút con đó với nút gốc

if (left < n && arr[left] > arr[largest]) {

largest = left;

}

if (right < n && arr[right] > arr[largest]) {

largest = right;

}

// Nếu nút gốc không phải là nút lớn nhất, thực hiện hoán đổi

if (largest != i) {

swap(&arr[i], &arr[largest]);

// Tiếp tục duyệt xuống cây con để đảm bảo tính chất heap

heapify(arr, n, largest);

}

}

// Hàm sắp xếp mảng bằng phương pháp heap sort

void heapSort(int arr[], int n) {

// Xây dựng heap từ mảng đầu vào

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

// Tách từng phần tử lớn nhất từ heap và đưa về cuối mảng

for (int i = n - 1; i > 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

int main() {

int arr[] = {41, 23, 4, 14, 56, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Mang truoc khi sap xep: ");

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

printf("%d ", arr[i]);

}

printf("\n");

heapSort(arr, n);

printf("Mang sau khi sap xep: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

1. **Viết chương trình cài đặt stack bằng mảng. Nhập vào stack**

**6 phần tử { 41, 23, 4, 14, 56, 1 }.**

**Sau đó xuất giá trị top ra khỏi stack và in các phần tử trong stack còn lại.**

#include <stdio.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

int is\_empty() {

return top == -1;

}

int is\_full() {

return top == MAX\_SIZE - 1;

}

void push(int item) {

if (!is\_full()) {

stack[++top] = item;

} else {

printf("Stack is full. Cannot push item %d.\n", item);

}

}

int pop() {

if (!is\_empty()) {

return stack[top--];

} else {

printf("Stack is empty. Cannot pop item.\n");

return -1;

}

}

int top\_value() {

if (!is\_empty()) {

return stack[top];

} else {

printf("Stack is empty. No top value.\n");

return -1;

}

}

int size() {

return top + 1;

}

int main() {

// Nhập vào stack 6 phần tử

int elements[] = {41, 23, 4, 14, 56, 1};

int num\_elements = sizeof(elements) / sizeof(elements[0]);

for (int i = 0; i < num\_elements; i++) {

push(elements[i]);

}

// Xuất giá trị top ra khỏi stack

int top\_value = pop();

printf("Giá trị top đã được xuất ra khỏi stack: %d\n", top\_value);

// In các phần tử trong stack còn lại

printf("Các phần tử còn lại trong stack: ");

while (!is\_empty()) {

printf("%d ", pop());

}

printf("\n");

return 0;

}

1. **Viết chương trình cài đặt hàng đợi bằng mảng. Nhập vào queue**

**6 phần tử { 41, 23, 4, 14, 56, 1 }. Sau đó thêm vào phần tử {55} và loại bỏ phần {23}.**

**In hàng đợi sau khi thêm và bỏ.**

#include <stdio.h>

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = 0;

int rear = -1;

int rong() {

return front > rear;

}

int day() {

return rear == MAX\_SIZE - 1;

}

void themVaoHangDoi(int phanTu) {

if (!day()) {

queue[++rear] = phanTu;

} else {

printf("Hang doi da day. Khong the them phan tu %d.\n", phanTu); }}

int layRaKhoiHangDoi() {

if (!rong()) {

return queue[front++];

} else {

printf("Hang doi rong. Khong the lay phan tu.\n");

return -1;}}

int giaTriPhiaTruoc() {

if (!rong()) {

return queue[front];

} else {

printf("Hang doi rong. Khong co gia tri phia truoc.\n");

return -1; }}

int kichThuoc() {

return rear - front + 1;}

int main() {

// Nhap vao hang doi 6 phan tu

int cacPhanTu[] = {41, 23, 4, 14, 56, 1};

int soPhanTu = sizeof(cacPhanTu) / sizeof(cacPhanTu[0]);

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

themVaoHangDoi(cacPhanTu[i]); }

// Them phan tu 55 vao hang doi

themVaoHangDoi(55);

printf("Hang doi sau khi them phan tu 55: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]); }

printf("\n");

// Loai bo phan tu 23 khoi hang doi

layRaKhoiHangDoi(); // Bo qua gia tri tra ve, khong can in ra

printf("Hang doi sau khi loai bo phan tu 23: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]); }

printf("\n");

return 0;

}

1. **Cho mảng 1 chiều các số nguyên. Viết hàm tính tổng các số chẵn**

**trong mảng bằng phương pháp đệ quy.**

#include <stdio.h>

int tinhTongChanDeQuy(int mang[], int kichThuoc) {

// Điều kiện dừng của đệ quy: nếu kích thước mảng là 0, trả về 0

if (kichThuoc == 0) {

return 0;

}

// Nếu phần tử cuối cùng trong mảng là số chẵn, thì trả về tổng của phần tử đó

// và tổng của các phần tử còn lại trong mảng (gọi đệ quy)

if (mang[kichThuoc - 1] % 2 == 0) {

return mang[kichThuoc - 1] + tinhTongChanDeQuy(mang, kichThuoc - 1);

}

// Nếu phần tử cuối cùng không phải số chẵn, thì chỉ gọi đệ quy để tính tổng

// của các phần tử còn lại trong mảng

else {

return tinhTongChanDeQuy(mang, kichThuoc - 1);

}

}

int main() {

int mang[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int kichThuoc = sizeof(mang) / sizeof(mang[0]);

int tongChan = tinhTongChanDeQuy(mang, kichThuoc);

printf("Tong cac so chan trong mang la: %d\n", tongChan);

return 0;}

1. **Cho mảng 1 chiều các số thực. Viết hàm đếm số lượng giá trị**

**dương trong mảng bằng phương pháp đệ quy.**

#include <stdio.h>

int demSoDuongDeQuy(float mang[], int kichThuoc) {

// Điều kiện dừng của đệ quy: nếu kích thước mảng là 0, trả về 0

if (kichThuoc == 0) {

return 0;

}

// Nếu phần tử cuối cùng trong mảng là số dương, thì trả về 1

// (để đếm phần tử này) cộng với số lượng các phần tử dương trong

// mảng con (mảng không bao gồm phần tử cuối cùng) (gọi đệ quy)

if (mang[kichThuoc - 1] > 0) {

return 1 + demSoDuongDeQuy(mang, kichThuoc - 1);

}

// Nếu phần tử cuối cùng không phải số dương, thì chỉ gọi đệ quy

// để đếm số lượng phần tử dương trong mảng con (mảng không bao

// gồm phần tử cuối cùng)

else {

return demSoDuongDeQuy(mang, kichThuoc - 1);

}

}

int main() {

float mang[] = {1.2, -2.3, 4.5, -6.7, 8.9, 10.1};

int kichThuoc = sizeof(mang) / sizeof(mang[0]);

int soLuongDuong = demSoDuongDeQuy(mang, kichThuoc);

printf("So luong gia tri duong trong mang la: %d\n", soLuongDuong);

return 0;

}

1. **Cho mảng 1 chiều các số nguyên. Viết hàm đệ quy xuất mảng.**

#include <stdio.h>

void xuatMangDeQuy(int mang[], int kichThuoc, int viTri) {

// Điều kiện dừng của đệ quy: nếu vị trí là cuối cùng của mảng, không làm gì cả

if (viTri == kichThuoc) {

return;

}

// In giá trị của phần tử tại vị trí hiện tại

printf("%d ", mang[viTri]);

// Gọi đệ quy để in giá trị của phần tử tiếp theo

xuatMangDeQuy(mang, kichThuoc, viTri + 1);

}

int main() {

int mang[] = {1, 2, 3, 4, 5};

int kichThuoc = sizeof(mang) / sizeof(mang[0]);

printf("Mang: ");

xuatMangDeQuy(mang, kichThuoc, 0); // Gọi hàm xuất mảng đệ quy, vị trí ban đầu là 0

printf("\n");

return 0;

}

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy đếm số lượng giá trị dương có trong mảng**

#include <stdio.h>

int demSoDuongDeQuy(float mang[], int kichThuoc, int viTri) {

// Điều kiện dừng của đệ quy: nếu vị trí là cuối cùng của mảng, không làm gì cả

if (viTri == kichThuoc) {

return 0; // Trả về 0 vì không còn phần tử nào để xét

}

// Nếu giá trị tại vị trí hiện tại là dương, thì cộng 1 vào kết quả

if (mang[viTri] > 0) {

return 1 + demSoDuongDeQuy(mang, kichThuoc, viTri + 1); // Gọi đệ quy và cộng 1 vào kết quả

} else {

return demSoDuongDeQuy(mang, kichThuoc, viTri + 1); // Nếu không, gọi đệ quy tiếp tục

}

}

int main() {

float mang[] = {1.5, -2.3, 3.7, -4.2, 5.0};

int kichThuoc = sizeof(mang) / sizeof(mang[0]);

int soLuongDuong = demSoDuongDeQuy(mang, kichThuoc, 0); // Gọi hàm đệ quy đếm số lượng giá trị dương trong mảng

printf("So luong gia tri duong trong mang: %d\n", soLuongDuong);

return 0;

}

1. **Hãy khai báo cấu trúc dữ liệu của cây nhị phân các số nguyên**

// Khai báo cấu trúc dữ liệu cho nút của cây nhị phân

struct Node {

int data; // Giá trị của nút

struct Node\* left; // Con trỏ sang nút con bên trái

struct Node\* right; // Con trỏ sang nút con bên phải

};

1. **Hãy khai báo cấu trúc dữ liệu của danh sách liên kết kép các số nguyên**

// Khai báo cấu trúc dữ liệu cho một nút của danh sách liên kết kép

struct Node {

int data; // Giá trị của nút

Node\* prev; // Con trỏ sang nút trước đó trong danh sách

Node\* next; // Con trỏ sang nút tiếp theo trong danh sách

};

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy tính tổng các giá trị dương có trong mảng**

#include <stdio.h>

// Hàm đệ quy tính tổng các giá trị dương trong mảng

double sumPositiveRecursive(double arr[], int n) {

// Trường hợp cơ bản: nếu chỉ còn 1 phần tử

if (n == 1) {

// Nếu giá trị là dương thì trả về giá trị đó, ngược lại trả về 0

return (arr[0] > 0) ? arr[0] : 0;

}

// Nếu giá trị hiện tại là dương thì cộng vào tổng và đệ quy tính tổng phần tử tiếp theo

if (arr[n - 1] > 0) {

return arr[n - 1] + sumPositiveRecursive(arr, n - 1);

} else {

// Ngược lại, chỉ đệ quy tính tổng phần tử tiếp theo

return sumPositiveRecursive(arr, n - 1);

}

}

int main() {

int n;

printf("Nhap so phan tu cua mang: ");

scanf("%d", &n);

// Nhập giá trị cho từng phần tử trong mảng

double arr[n];

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

printf("Nhap phan tu thu %d: ", i + 1);

scanf("%lf", &arr[i]);

}

// Gọi hàm đệ quy tính tổng các giá trị dương trong mảng

double sum = sumPositiveRecursive(arr, n);

printf("Tong cac gia tri duong trong mang: %.2lf\n", sum);

return 0;

}

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy tính tổng các giá trị có trong mảng.**

#include <stdio.h>

// Hàm đệ quy tính tổng các giá trị trong mảng

double sumRecursive(double arr[], int n) {

// Trường hợp cơ bản: nếu chỉ còn 1 phần tử

if (n == 1) {

return arr[0]; // Trả về giá trị của phần tử đầu tiên

}

// Tính tổng phần tử cuối cùng và đệ quy tính tổng các phần tử còn lại

return arr[n - 1] + sumRecursive(arr, n - 1);

}

int main() {

int n;

printf("Nhap so phan tu cua mang: ");

scanf("%d", &n);

// Nhập giá trị cho từng phần tử trong mảng

double arr[n];

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

printf("Nhap phan tu thu %d: ", i + 1);

scanf("%lf", &arr[i]);

}

// Gọi hàm đệ quy tính tổng các giá trị trong mảng

double sum = sumRecursive(arr, n);

printf("Tong cac gia tri trong mang: %.2lf\n", sum);

return 0;

}

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy kiểm tra mảng có thỏa mảng tính chất “toàn giá trị âm”**

#include <stdio.h>

// Hàm đệ quy kiểm tra mảng có toàn giá trị âm

int isAllNegative(double arr[], int n) {

// Trường hợp cơ bản: nếu chỉ còn 1 phần tử

if (n == 1) {

return arr[0] < 0; // Trả về 1 nếu phần tử đầu tiên là số âm, ngược lại trả về 0

}

// Kiểm tra giá trị phần tử cuối cùng và đệ quy kiểm tra các phần tử còn lại

return (arr[n - 1] < 0) && isAllNegative(arr, n - 1);

}

int main() {

int n;

printf("Nhap so phan tu cua mang: ");

scanf("%d", &n);

// Nhập giá trị cho từng phần tử trong mảng

double arr[n];

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

printf("Nhap phan tu thu %d: ", i + 1);

scanf("%lf", &arr[i]);

}

// Gọi hàm đệ quy kiểm tra mảng có thỏa mãn tính chất "toàn giá trị âm"

if (isAllNegative(arr, n)) {

printf("Mang thoa man tinh chat 'toan gia tri am'\n");

} else {

printf("Mang khong thoa man tinh chat 'toan gia tri am'\n");

}

return 0;

}

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy tìm giá trị lớn nhất có trong mảng**

#include <stdio.h>

// Hàm đệ quy tìm giá trị lớn nhất trong mảng

double findMax(double arr[], int n) {

// Trường hợp cơ bản: nếu chỉ còn 1 phần tử

if (n == 1) {

return arr[0]; // Trả về giá trị của phần tử đầu tiên

}

// Tìm giá trị lớn nhất giữa phần tử cuối cùng và giá trị lớn nhất trong mảng con trước đó

double max = findMax(arr, n - 1);

if (arr[n - 1] > max) {

return arr[n - 1];

} else {

return max;

}

}

int main() {

int n;

printf("Nhap so phan tu cua mang: ");

scanf("%d", &n);

// Nhập giá trị cho từng phần tử trong mảng

double arr[n];

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

printf("Nhap phan tu thu %d: ", i + 1);

scanf("%lf", &arr[i]);

}

// Gọi hàm đệ quy tìm giá trị lớn nhất trong mảng

double max = findMax(arr, n);

printf("Gia tri lon nhat trong mang la: %lf\n", max);

return 0;

}

1. **Cho mảng 1 chiều các số thực. Hãy viết hàm đệ quy sắp xếp các giá trị trong mảng tăng dần**

#include <stdio.h>

// Hàm đổi chỗ giá trị của hai biến

void swap(double \*a, double \*b) {

double temp = \*a;

\*a = \*b;

\*b = temp;

}

// Hàm đệ quy sắp xếp mảng tăng dần

void selectionSortRecursive(double arr[], int n) {

// Trường hợp cơ bản: nếu chỉ còn 1 phần tử

if (n == 1) {

return;

}

// Tìm giá trị nhỏ nhất trong mảng con trước đó

int minIndex = 0;

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

if (arr[i] < arr[minIndex]) {

minIndex = i;

}

}

// Nếu giá trị nhỏ nhất không nằm ở vị trí đầu tiên, hoán đổi giá trị đầu tiên với giá trị nhỏ nhất

if (minIndex != 0) {

swap(&arr[0], &arr[minIndex]);

}

// Đệ quy sắp xếp mảng con từ phần tử thứ 2 đến phần tử cuối cùng

selectionSortRecursive(&arr[1], n - 1);

}

int main() {

int n;

printf("Nhap so phan tu cua mang: ");

scanf("%d", &n);

// Nhập giá trị cho từng phần tử trong mảng

double arr[n];

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

printf("Nhap phan tu thu %d: ", i + 1);

scanf("%lf", &arr[i]);

}

// Gọi hàm đệ quy sắp xếp mảng tăng dần

selectionSortRecursive(arr, n);

// In mảng sau khi sắp xếp

printf("Mang sau khi sap xep tang dan: ");

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

printf("%lf ", arr[i]);

}

printf("\n");

return 0;

}

1. **Tính T(n) = 1 x 2 x 3 x … x n**

#include <stdio.h>

// Hàm tính giai thừa đệ quy

int giaiThua(int n) {

if (n == 0 || n == 1) {

return 1; // Trường hợp cơ sở: 0! = 1, 1! = 1

} else {

return n \* giaiThua(n - 1); // Tính giai thừa của n từ đệ quy

}

}

int main() {

int n;

printf("Nhap gia tri n: ");

scanf("%d", &n);

// Gọi hàm tính giai thừa và in kết quả

int result = giaiThua(n);

printf("Giai thua cua %d la: %d\n", n, result);

return 0;

}