|  |
| --- |
| #include <stdio.h>    void Swap(int &a, int &b)  {   int c = a;  a = b;  b = c;  }    void SelectionSort(int a[],int N ){ //Ghi chú: tại sao không sử dụng kí hiệu & trong hàm này?    int min; //chỉ số phần tử nhỏ nhất trong dãy hiện hành     for (int  i=0; i<N-1 ; i++){ //Ghi chu: vòng lặp này dùng để làm gì?     min = i;        for(int j = i+1; j < N ; j++){ //Ghi chu: vòng lặp này dùng để làm gì?          if (a[j] < a[min]){                  min = j; //Ghi chu: thao tác này dùng để làm gì?                           }                 }          if (min != i){          Swap(a[min], a[i]); //Ghi chu: thao tác này dùng để làm gì?                 }    }  }    void main()  {    int x[10] = {12, 2, 8, 5, 1, 6, 4, 15}; // khởi tạo các giá trị trong mảng    int n = 8; // số phần tử của mảng      SelectionSort(x, n);      for (int  i=0; i<n ; i++){    printf("%d ", x[i]);    }  } |

*Yêu cầu*

1. Biên dịch đoạn chương trình nêu trên.
2. Tại sao trong hàm SelectionSort, vòng lặp thứ nhất có điều kiện là i < N-1?
   1. Sau mỗi vòng lặp, phần tử nhỏ nhất trong dãy chưa được sắp xếp sẽ được hoán đổi với phần tử tại vị trí i. Điều này đảm bảo rằng tại mỗi vòng lặp, phần tử ở vị trí i sẽ được đặt đúng vị trí của nó trong dãy đã sắp xếp.
   2. Đến lần lặp thứ N-1 (khi i = N-1), ta đã chắc chắn rằng tất cả các phần tử trước đó đều đã được sắp xếp. Phần tử cuối cùng không cần phải kiểm tra thêm, vì nó sẽ tự động ở đúng vị trí sau khi các phần tử còn lại được sắp xếp.
3. Trả lời các dòng lệnh có yêu cầu ghi chú.
   1. Ký hiệu & trong C++ dùng để truyền tham chiếu, nhưng đối với mảng, khi truyền vào hàm, nó tự động truyền bằng tham chiếu vì mảng thực chất là con trỏ. Do đó, không cần sử dụng & trong tham số của hàm SelectionSort vì bất kỳ thay đổi nào trong mảng bên trong hàm cũng sẽ ảnh hưởng trực tiếp đến mảng gốc.
   2. Vòng lặp này duyệt qua từng phần tử của mảng. Mục đích là để chọn phần tử đầu tiên của dãy chưa được sắp xếp. Vòng lặp chạy N-1 lần, vì sau khi sắp xếp N-1 phần tử, phần tử cuối cùng tự nhiên sẽ ở vị trí đúng.
   3. Điều kiện này kiểm tra xem phần tử tại vị trí j có nhỏ hơn phần tử hiện tại tại vị trí min không. Nếu đúng, nó cập nhật min thành j, có nghĩa là phần tử tại vị trí j bây giờ được coi là nhỏ nhất trong dãy chưa sắp xếp.
   4. Điều kiện này kiểm tra xem phần tử nhỏ nhất (a[min]) có khác với phần tử hiện tại (a[i]) hay không. Nếu đúng, nó sẽ hoán đổi hai phần tử này, đảm bảo rằng phần tử nhỏ nhất sẽ được đưa lên đầu của dãy chưa sắp xếp.
4. Sửa lại chương trình để nhập dãy số nguyên từ file input.txt có nội dung như sau:

5 1 2 3 8 6 23 10

Sau đó dùng thuật toán Selection Sort sắp xếp dãy số nguyên trên tăng dần.

#include <stdio.h>

#include <stdlib.h> // Dùng cho thao tác với file

// Hàm hoán đổi hai giá trị

void Swap(int &a, int &b) {

int c = a; a = b; b = c;

}

// Hàm sắp xếp tăng dần bằng thuật toán Selection Sort

void SelectionSortAsc(int a[], int N) {

int min;

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

min = i;

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

if (a[j] < a[min]) { // Tìm phần tử nhỏ nhất

min = j;

}

}

if (min != i) {

Swap(a[min], a[i]); // Đưa phần tử nhỏ nhất lên vị trí đầu

}

}

}

int main() {

FILE \*inputFile;

inputFile = fopen("input.txt", "r");

if (inputFile == NULL) {

printf("Lỗi: Không thể mở file input.txt\n");

return 1;

}

int x[100]; // Giả sử kích thước tối đa của mảng là 100

int n = 0; // Biến lưu số phần tử của mảng

// Đọc các số nguyên từ file

while (fscanf(inputFile, "%d", &x[n]) != EOF) {

n++;

}

fclose(inputFile); // Đóng file sau khi đọc

// Sắp xếp mảng theo thứ tự tăng dần

SelectionSortAsc(x, n);

// Xuất mảng đã sắp xếp

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

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

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

}

return 0;

}

1. Sửa hàm SelectionSort trên để sắp xếp dãy số nguyên ở câu 4 giảm dần.

void SelectionSortDesc(int a[], int N) {

int max;

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

max = i;

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

if (a[j] > a[max]) { // Tìm phần tử lớn nhất

max = j;

}

}

if (max != i) {

Swap(a[max], a[i]); // Đưa phần tử lớn nhất lên vị trí đầu

}

}

}

|  |
| --- |
| void Shift (int a[], int left, int right){    int  x, curr, joint;    curr = left; joint =2\*curr+1; *// a :  Phần tử liên đới joint*            x = a[curr];      while (joint <= right){                   if (joint < right){ *// Ghi chú: điều kiện này có ý nghĩa gì?*                         if (a[joint] < a[joint+1]){                               joint = joint+1;                         }                  }                   if (a[joint]<x){                        break;*// Thỏa quan hệ liên đới*                  }                    a[curr] = a[joint];                 curr = joint; *// Xét khả năng hiệu chỉnh lan truyền*                 joint = 2\*curr+1;    }       a[curr] = x;  } |

*Yêu cầu*

1. Trả lời các dòng lệnh có yêu cầu ghi chú.
   1. Điều kiện if (joint < right)

Ý nghĩa: Điều kiện này kiểm tra xem chỉ số joint có nhỏ hơn right hay không, tức là kiểm tra xem chỉ số joint có phải là phần tử cuối cùng của đoạn đang xét hay không. Nếu joint nhỏ hơn right, điều này có nghĩa là còn ít nhất một phần tử nữa để so sánh, và ta có thể so sánh giá trị tại joint với giá trị tại joint+1 để xác định phần tử lớn hơn.

* 1. if (a[joint] < a[joint+1]) joint = joint+1

Ý nghĩa: Câu lệnh này so sánh hai phần tử liên tiếp trong cây (tại vị trí joint và joint+1) và chọn phần tử lớn hơn để tiếp tục so sánh. Nếu phần tử tại joint+1 lớn hơn phần tử tại joint, chỉ số joint sẽ được tăng lên (chuyển sang joint+1), tức là chúng ta sẽ xét phần tử lớn hơn để duy trì tính chất heap.

1. Cho biết chức năng của đoạn chương trình trên.
   1. Đoạn chương trình trên là một phần của thuật toán điều chỉnh cây nhị phân để thỏa mãn tính chất **heap tối đa** (Max Heap). Hàm Shift này hiệu chỉnh một phần của cây nhị phân với nút gốc là a[left] và các con ở chỉ số từ left đến right để đảm bảo rằng nút gốc lớn hơn hoặc bằng các nút con. Nếu không, nó sẽ hoán đổi giá trị của nút gốc với nút con lớn hơn, sau đó tiếp tục hiệu chỉnh cho đến khi cây thỏa mãn tính chất Max Heap.
2. Viết hàm void CreateHeap(int a[], int N); để chuyển đổi dãy a0, a1, …, aN-1 thành heap.

Gợi ý: Sử dụng hàm **Shift** bên trên với left hiện hành là phần tử ở giữa dãy (**(N-1)/2**). Lặp lại quá trình trên với left giảm dần về đầu dãy.

void CreateHeap(int a[], int N) {

// Bắt đầu từ phần tử ở giữa (N-1)/2 và lùi về đầu dãy

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

Shift(a, i, N - 1); // Hiệu chỉnh dãy từ vị trí i

}

}

1. Viết hàm void HeapSort(int a[], int N); để sắp xếp một dãy số nguyên tăng dần.

  Gợi ý: *Giai đoạn 1*: Hiệu chỉnh dãy ban đầu thành heap *Giai đoạn 2*: Sắp xếp dãy số dựa trên heap.

* Xét dãy hiện hành là dãy nhập
* Hoán vị phần tử lớn nhất (a0) về vị trí cuối.
* Xét dãy hiện hành loại đã trừ phần tử cuối.
* Hiệu chỉnh lại dãy hiện hành thành heap   • Lặp lại quá trình trên tới hết dãy ban đầu.

void HeapSort(int a[], int N) {

// Giai đoạn 1: Chuyển đổi dãy thành Max Heap

CreateHeap(a, N);

// Giai đoạn 2: Sắp xếp dãy dựa trên Max Heap

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

// Hoán đổi phần tử lớn nhất (gốc heap) với phần tử cuối cùng của dãy

Swap(a[0], a[i]);

// Hiệu chỉnh lại heap, loại bỏ phần tử cuối đã được sắp xếp

Shift(a, 0, i - 1);

}

}

1. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để  sắp xếp dãy số nguyên sau tăng dần:

44  55  12  42  94  18  6  67

1. Viết lại thuật toán Heap Sort để sắp xếp dãy số ở câu 3 giảm dần.

void ShiftMin(int a[], int left, int right) {

int x, curr, joint;

curr = left;

joint = 2 \* curr + 1; // Phần tử liên đới joint

x = a[curr];

while (joint <= right) {

if (joint < right) { // Kiểm tra nếu có phần tử bên phải

if (a[joint] > a[joint + 1]) { // Tìm phần tử nhỏ hơn

joint = joint + 1;

}

}

if (a[joint] > x) {

break; // Nếu phần tử liên đới lớn hơn phần tử hiện tại, dừng lại

}

a[curr] = a[joint];

curr = joint;

joint = 2 \* curr + 1;

}

a[curr] = x;

}

void CreateMinHeap(int a[], int N) {

// Chuyển dãy thành Min Heap

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

ShiftMin(a, i, N - 1);

}

}

void HeapSortDesc(int a[], int N) {

// Giai đoạn 1: Chuyển dãy thành Min Heap

CreateMinHeap(a, N);

// Giai đoạn 2: Sắp xếp dựa trên Min Heap (giảm dần)

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

// Hoán đổi phần tử nhỏ nhất với phần tử cuối cùng

Swap(a[0], a[i]);

// Hiệu chỉnh lại heap, loại bỏ phần tử cuối đã sắp xếp

ShiftMin(a, 0, i - 1);

}

}

|  |
| --- |
| void QuickSort(int a[], int left, int right){    int i, j, x;      if (left >= right){               return;          }            x = a[(left+right)/2]; *// chọn phần tử giữa làm giá trị mốc*         i = left; j = right;        while(i < j) {             while(a[i] < x){                      i++;               }                 while(a[j] > x){                      j--;               }                if(i <= j) {                   Swap(a[i], a[j]);                   i++ ;                   j--;              }         }           QuickSort(a, left, j);         QuickSort(a, i, right);  } |

*Yêu cầu*

1. Bổ sung các hàm trên vào chương trình mẫu (CacThuatToanSapXep) đồng thời thay đổi hàm main và file input để  sắp xếp dãy số nguyên sau tăng dần:

42  23  74  11  65  58  94  36  99  87

#include <stdio.h>

void Swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

void QuickSort(int a[], int left, int right) {

int i, j, x;

if (left >= right) return;

x = a[(left + right) / 2]; // Chọn phần tử giữa làm giá trị mốc

i = left;

j = right;

while (i < j) {

while (a[i] < x) i++; // Di chuyển i đến khi a[i] >= x

while (a[j] > x) j--; // Di chuyển j đến khi a[j] <= x

if (i <= j) {

Swap(a[i], a[j]);

i++;

j--;

}

}

QuickSort(a, left, j); // Sắp xếp phần mảng bên trái

QuickSort(a, i, right); // Sắp xếp phần mảng bên phải

}

int main() {

int a[] = {42, 23, 74, 11, 65, 58, 94, 36, 99, 87};

int N = sizeof(a) / sizeof(a[0]);

QuickSort(a, 0, N - 1);

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

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

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

}

printf("\n");

return 0;

}

1. Sửa lại chương trình để đếm số phép gán và số phép so sánh sự dụng trong hàm QuickSort.

#include <stdio.h>

int assignmentCount = 0; // Biến đếm số phép gán

int comparisonCount = 0; // Biến đếm số phép so sánh

void Swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

assignmentCount += 3; // 3 phép gán (temp = a, a = b, b = temp)

}

void QuickSort(int a[], int left, int right) {

int i, j, x;

if (left >= right) return;

x = a[(left + right) / 2]; // Chọn phần tử giữa làm giá trị mốc

assignmentCount++; // Phép gán cho x

i = left;

j = right;

while (i < j) {

while (a[i] < x) {

comparisonCount++; // Phép so sánh a[i] < x

i++;

}

comparisonCount++; // Phép so sánh khi điều kiện dừng

while (a[j] > x) {

comparisonCount++; // Phép so sánh a[j] > x

j--;

}

comparisonCount++; // Phép so sánh khi điều kiện dừng

if (i <= j) {

comparisonCount++; // Phép so sánh i <= j

Swap(a[i], a[j]);

i++;

j--;

}

}

QuickSort(a, left, j); // Sắp xếp phần mảng bên trái

QuickSort(a, i, right); // Sắp xếp phần mảng bên phải

}

int main() {

int a[] = { 42, 23, 74, 11, 65, 58, 94, 36, 99, 87 };

int N = sizeof(a) / sizeof(a[0]);

QuickSort(a, 0, N - 1);

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

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

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

}

printf("\n");

printf("So phep gan: %d\n", assignmentCount);

printf("So phep so sanh: %d\n", comparisonCount);

return 0;

}

|  |
| --- |
| int b[MAX], c[MAX], nb, nc; *//* Ghi chú: 2 mảng này dùng để làm gì?    void Distribute(int  a[], int N,  int &nb, int &nc, int k){    int i, pa, pb, pc; //Ghi chú: các biến này có ý nghĩa gì?    pa = pb = pc = 0;   while (pa < N){    for (i=0; (pa<N) && (i<k); i++, pa++, pb++){ //Ghi chú: vòng lặp này có ý nghĩa gì?     b[pb] = a[pa];               }      for (i=0; (pa<N) && (i<k); i++, pa++, pc++){ //Ghi chú: vòng lặp này có ý nghĩa gì?    c[pc] = a[pa];               }    }    nb = pb; nc = pc;  } |

|  |
| --- |
| void Merge(int a[], int nb, int nc, int k){    int pa, pb, pc;  pa = pb = pc = 0;      while ((pb < nb) && (pc < nc)){      MergeSubarr(a, nb, nc, pa, pb, pc, k);           }      while (pb < nb){    a[pa ++] = b[pb ++]; //Ghi chú: câu lệnh này có ý nghĩa gì?           }      while (pc < nc){    a[pa ++] = c[pc ++]; //Ghi chú: câu lệnh này có ý nghĩa gì?           }  } |

Trong đó hàm MergeSubarr được cài đặt như sau:

|  |
| --- |
| void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k){    int rb, rc;  rb = min(nb, pb+k);         rc = min(nc, pb+k);      while ((pb < rb) && (pc < rc)){      if (b[pb] < c[pc])      a[pa ++] = b[pb ++];        else  a[pa ++] = c[pc ++];          }      while (pb < rb){      a[pa ++] = b[pb ++];          }      while (pc < rc){        a[pa ++] = c[pc ++];          }  } |

*Yêu cầu*

1. Trả lời các dòng lệnh có yêu cầu ghi chú.
   1. int b[MAX], c[MAX], nb, nc;:
      1. b[] và c[]: Đây là hai mảng phụ dùng để chứa các phần tử của mảng a[] trong quá trình phân phối (distribute) và hợp nhất (merge) của thuật toán Merge Sort.
      2. nb và nc: Lần lượt là số lượng phần tử được phân phối vào mảng b[] và c[].
   2. int i, pa, pb, pc;:
      1. pa: Chỉ số hiện tại của mảng gốc a[] mà ta đang xét trong hàm Distribute.
      2. pb và pc: Lần lượt là chỉ số hiện tại của hai mảng phụ b[] và c[] mà ta sẽ ghi phần tử từ mảng a[] vào.
   3. Vòng lặp for (i=0; (pa<N) && (i<k); i++, pa++, pb++):
      1. Vòng lặp này dùng để phân phối tối đa k phần tử từ mảng a[] vào mảng b[] tại vị trí pb. Điều kiện kiểm tra pa<N đảm bảo rằng không vượt quá số phần tử của mảng.
   4. Vòng lặp for (i=0; (pa<N) && (i<k); i++, pa++, pc++):
      1. Tương tự như trên nhưng để phân phối các phần tử còn lại từ mảng a[] vào mảng c[] tại vị trí pc.
   5. a[pa++] = b[pb++];:
      1. Lệnh này dùng để chép phần tử từ mảng b[] về lại mảng a[] sau khi so sánh hoặc khi mảng b[] còn phần tử chưa được xử lý.
   6. a[pa++] = c[pc++];:
      1. Tương tự, lệnh này dùng để chép phần tử từ mảng c[] về lại mảng a[] sau khi so sánh hoặc khi mảng c[] còn phần tử chưa được xử lý.
2. Cho biết chức năng của từng hàm trên.

 **Hàm Distribute**:

* Phân chia mảng a[] thành hai mảng phụ b[] và c[], mỗi mảng phụ chứa tối đa k phần tử từ mảng a[]. Số phần tử của hai mảng phụ này được lưu trong nb và nc.

 **Hàm MergeSubarr**:

* Thực hiện việc trộn hai mảng phụ b[] và c[] có độ dài tối đa k phần tử từ mảng b[] và c[]. Các phần tử được so sánh và chép trở lại vào mảng a[] theo thứ tự tăng dần.

 **Hàm Merge**:

* Gọi MergeSubarr để thực hiện việc trộn toàn bộ hai mảng b[] và c[] trở lại mảng a[] sau khi đã chia chúng thành các đoạn con có kích thước tối đa k phần tử.

1. Bổ sung các hàm cần thiết vào chương trình mẫu (CacThuatToanSapXep) và viết hàm void MergeSort(int a[], int N); để sắp xếp dãy số nguyên sau tăng dần.

5  2  9  3  7  2  4 11 Gợi ý: Xem lại 2 thao tác đã nêu bên trên.

#include <stdio.h>

#include <algorithm> // C++ cần thư viện này cho hàm min

#define MAX 100

int b[MAX], c[MAX], nb, nc;

void Swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

// Hàm phân chia mảng a[] thành b[] và c[]

void Distribute(int a[], int N, int& nb, int& nc, int k) {

int i, pa = 0, pb = 0, pc = 0;

while (pa < N) {

for (i = 0; (pa < N) && (i < k); i++, pa++, pb++) {

b[pb] = a[pa]; // Gán phần tử từ a sang b

}

for (i = 0; (pa < N) && (i < k); i++, pa++, pc++) {

c[pc] = a[pa]; // Gán phần tử từ a sang c

}

}

nb = pb;

nc = pc;

}

// Hàm trộn các mảng con b[] và c[] vào lại mảng a[]

void MergeSubarr(int a[], int nb, int nc, int& pa, int& pb, int& pc, int k) {

int rb = std::min(nb, pb + k); // Xác định ranh giới của b[]

int rc = std::min(nc, pc + k); // Xác định ranh giới của c[]

while (pb < rb && pc < rc) { // Trộn b[] và c[] theo thứ tự tăng dần

if (b[pb] < c[pc]) {

a[pa++] = b[pb++];

}

else {

a[pa++] = c[pc++];

}

}

while (pb < rb) { // Chép phần còn lại của b[]

a[pa++] = b[pb++];

}

while (pc < rc) { // Chép phần còn lại của c[]

a[pa++] = c[pc++];

}

}

// Hàm trộn hoàn toàn hai mảng b[] và c[] về lại a[]

void Merge(int a[], int nb, int nc, int k) {

int pa = 0, pb = 0, pc = 0;

while (pb < nb && pc < nc) {

MergeSubarr(a, nb, nc, pa, pb, pc, k);

}

while (pb < nb) {

a[pa++] = b[pb++];

}

while (pc < nc) {

a[pa++] = c[pc++];

}

}

// Hàm MergeSort chính

void MergeSort(int a[], int N) {

int k = 1;

while (k < N) {

Distribute(a, N, nb, nc, k); // Phân phối các phần tử vào b[] và c[]

Merge(a, nb, nc, k); // Trộn b[] và c[] về lại mảng a[]

k \*= 2; // Tăng kích thước khối phần tử

}

}

int main() {

int a[] = { 5, 2, 9, 3, 7, 2, 4, 11 };

int N = sizeof(a) / sizeof(a[0]);

MergeSort(a, N);

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

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

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

}

printf("\n");

return 0;

}

1. Sửa lại chương trình để sắp xếp dãy số trên giảm dần.

void MergeSubarr(int a[], int nb, int nc, int& pa, int& pb, int& pc, int k) {

int rb = std::min(nb, pb + k);

int rc = std::min(nc, pc + k);

while (pb < rb && pc < rc) {

if (b[pb] > c[pc]) { // Đổi dấu so sánh để sắp xếp giảm dần

a[pa++] = b[pb++];

}

else {

a[pa++] = c[pc++];

}

}

while (pb < rb) {

a[pa++] = b[pb++];

}

while (pc < rc) {

a[pa++] = c[pc++];

}

}