**Chương 3:**

**XẾP THỨ TỰ VÀ TÌM KIẾM**

🟑**CÂU HỎI**

Câu 1:

|  |  |
| --- | --- |
| **Thuật toán sắp xếp** | **Độ phức tạp** |
| Selection Sort |  |
| Intsertion Sort |  |
| Interchange Sort |  |
| Bubble Sort |  |
| Quick Sort |  |
| Heap Sort |  |

Trong các phương pháp sắp xếp đã học, phương pháp *Quick Sort* là tối ưu nhất vì mặc dù *Heap Sort* và *Quick Sort* đều có độ phức tạp là nhưng thức tế thời gian thực hiện và thuật toán của Quick Sort tối ưu hơn so với *Heap Sort*. Còn phương pháp kém tối ưu nhất là phương pháp *Selection Sort* vì với phương pháp này, ta phải thực hiện lặp lại việc xét từng phần tử trong danh sách để chọn phần tử bé nhất đưa lên đầu danh sách. Việc lặp lại này làm cho thời gian thực hiện kéo dài và lâu hơn so với những phương pháp khác có độ phức tạp là .

Câu 2:

Trong hai phương pháp tìm kiếm đã học, cả hai phương pháp đều như nhau trong trường hợp danh sách chưa được sắp xếp thứ tự. Vì khi đó, nếu ta sử dụng phương pháp tìm kiếm nhị phân thì ta phải dùng thuật toán sắp xếp để sắp xếp lại danh sách rồi mới thực hiện tìm kiếm nhị phân. Do vậy, thời gian thực hiện đối với tìm kiếm nhị phân sẽ kéo dài ra và có thể tương đương với thời gian thực hiện tìm kiếm tuần tự.

Câu 3:

Ngoài các phương pháp xếp thứ tự đã học, ta còn có những phương pháp sắp xếp thứ tự khác trong đó có phương pháp *Merge Sort*:

**🟋**Thuật toán sắp xếp *Merge Sort* là một trong những thuật toán có độ phức tạp ở mức trung bình và cùng sử dùng phương pháp chia để trị giống thuật toán sắp xếp nhanh quick sort. Thuật toán này không chỉ áp dụng trong sắp xếp mà còn ở nhiều bài toán khác.

🟋Ý tưởng của thuật toán *Merge Sort* *(Sắp xếp trộn)*

Ý tưởng chúng ta sẽ chia mảng lớn thành những mảng con nhỏ hơn bằng cách chia đôi mảng lớn và chúng ta tiếp tục chia đôi các mảng con cho tới khi mảng con nhỏ nhất chỉ còn 1 phần tử. Sau đó chúng ta sẽ tiếng hành so sánh 2 mảng con có cùng mảng cơ sở (khi chúng ta chia đôi mảng lớn thành 2 mảng con thì mảng lớn đó chúng ta gọi là mảng cơ sở của 2 mảng con đó) khi so sánh chúng sẽ vừa sắp xếp vừa ghép 2 mảng con đó lại thành mảng cơ sở, chúng ta tiếp tục so sánh và ghép các mảng con lại đến khi còn lại mảng duy nhất thì đó là mảng đã được sắp xếp.

🟋Thuật toán:

MergeSort(arr[], l,  r)

If r > l

     1. Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:

             middle m = (l+r)/2

     2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:

             mergeSort(arr, l, m)

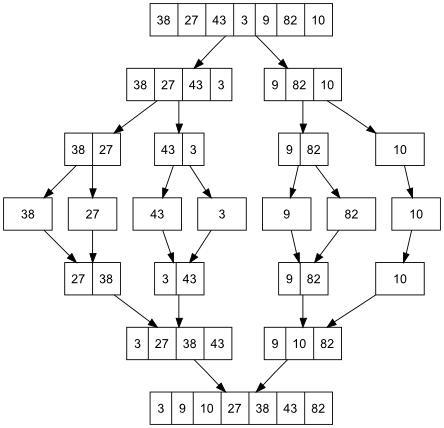
     3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:

             mergeSort(arr, m+1, r)

     4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):

             merge(arr, l, m, r)

🟋Ví dụ minh họa:



🟋Cài đặt giải thuật *Merge Sort*:

#include<stdlib.h>

#include<stdio.h>

// Gộp hai mảng con arr[l...m] và arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 =  r - m;

/\* Tạo các mảng tạm \*/

int L[n1], R[n2];

/\* Copy dữ liệu sang các mảng tạm \*/

for (i = 0; i < n1; i++)

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

for (j = 0; j < n2; j++)

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

/\* Gộp hai mảng tạm vừa rồi vào mảng arr\*/

i = 0; // Khởi tạo chỉ số bắt đầu của mảng con đầu tiên

j = 0; // Khởi tạo chỉ số bắt đầu của mảng con thứ hai

k = l; // IKhởi tạo chỉ số bắt đầu của mảng lưu kết quả

while (i < n1 && j < n2)

{

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

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy các phần tử còn lại của mảng L vào arr nếu có \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy các phần tử còn lại của mảng R vào arr nếu có \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

/\* l là chỉ số trái và r là chỉ số phải của mảng cần được sắp xếp \*

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Tương tự (l+r)/2, nhưng cách này tránh tràn số khi l và r lớn

int m = l+(r-l)/2;

// Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

🟋Độ phức tạp của thuật toán:

|  |  |
| --- | --- |
| Trường hợp | Độ phức tạp |
| Trường hợp tốt nhất |  |
| Trường hợp trung bình |  |
| Trường hợp xấu nhất |  |

Không gian bộ nhớ sử dụng: