**CÂU HỎI CHƯƠNG 3**

**Câu 1:** Trong các phương pháp xếp thứ tự đã học, phương pháp nào tối ưu nhất, và kém tối ưu nhất? Tại sao?

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Trường hợp** | **Số lần so sánh** | **Số lần hoán vị** |
| Selection Sort | Tốt nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps1.jpg | 0 |
| Xấu nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps2.jpg | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps3.jpg |
| Insertion Sort | Tốt nhất | n-1 | 2(n-1) |
| Xấu nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps4.jpg | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps5.jpg |
| Bubble Sort | Tốt nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps6.jpg | 0 |
| Xấu nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps7.jpg | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps8.jpg |
| Interchange Sort | Tốt nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps9.jpg | 0 |
| Xấu nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps10.jpg | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps11.jpg |
| Quick Sort | Tốt nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps12.jpg | |
| Xấu nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps13.jpg | |
| Heap Sort | Tốt nhất | C:\Users\USER\AppData\Local\Temp\ksohtml11804\wps14.jpg | |
| Xấu nhất |

Theo bảng độ phức tạp trên:

Quick Sort là tối ưu nhất

SelectionSort là kém tối ưu nhất

**Câu 2:** Trong 2 phương pháp tìm kiếm đã học, trường hợp nào thì cả 2 phương pháp đều như nhau? Giải thích tại sao?

Trong trường hợp duyệt danh sách có thứ tự, thì cả hai phương pháp đều như nhau. Vì trong trường hợp này, số lần so sánh của 2 phương pháp là như nhau.

**Câu 3:** Ngoài các phương pháp xếp thứ tự đã học, hãy tìm hiểu thêm một phương pháp xếp thứ tự khác, giới thiệu sơ và giải thích.

Phương pháp khác là Merge Sort, là thuật toán có độ phức tạp trung bình và cũng sử dụng phương pháp chia để trị như Quick Sort. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp.

Độ phức tạp thuật toán

Trường hợp tốt: O(nlog(n))

Trung bình: O(nlog(n))

Trường hợp xấu: O(nlog(n))

Không gian bộ nhớ sử dụng: O(n)

Code:

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2]; // tạo mảng tạm.

/\* 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);

}

}