**BT Chương 3**

**Câu 1:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp | Số lần so sánh | Số lần hoán vị |
| Selection Sort | Tốt nhất |  | 0 |
| Xấu nhất |  |  |
| Insertion Sort | Tốt nhất | n-1 | 2(n-1) |
| Xấu nhất |  |  |
| Bubble Sort | Tốt nhất |  | 0 |
| Xấu nhất |  |  |
| Interchange Sort | Tốt nhất |  | 0 |
| Xấu nhất |  |  |
| Quick Sort | Tốt nhất |  | |
| Xấu nhất |  | |
| Heap Sort | Tốt nhất |  | |
| Xấu nhất |

Theo bảng độ phức tạp trên:

* Quick Sort là tối ưu nhất
* SelectionSort là kém tối ưu nhất

# Câu 2:

# \*Cả hai phương pháp tìm kiếm như nhau khi:

# -Tìm kiếm tuần tự ở trường hợp trung bình ((n+1)/2 ; n là độ lớn đầu vào)

# -Tìm kiếm nhị phân ở trường hợp xấu nhất (log n) với n là độ lớn đầu vào (Đối với dãy đã sắp xếp)

# \*Giải thích

# Vì tìm kiếm tuần tự chạy từ đầu đến khi tìm được sẽ dừng, còn tìm kiếm nhị phân đã được sắp xếp dãy, và mỗi lần tìm kiếm lại phân thành các dãy nên trường hợp trung bình của tìm kiếm tuần tự và xấu nhất của tìm kiếm nhị phân là chạy đều như nhau.

# Câu 3:

# Ví dụ: Merge sort

# Merge sort là một thuật toán chia để trị. 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. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất 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);

    }

}