Bài tập câu hỏi chương 3

**Câu 1:**

Trong các phương pháp sắp xếp thứ tự đã học, thực ra không có phương pháp nào là tối ưu hoặc kém tối ưu nhất. Trong mỗi trường hợp khác nhau, dựa vào yêu cầu bài toán, dữ liệu đầu vào và bộ nhớ cần sử dụng… để lựa chọn phương pháp thích hợp. Ví dụ như những bài toán sắp xếp chỉ gồm 10 – 20 phần tử, selection sort sẽ thực hiện nhanh hơn quick sort, những bài toán hơn 1000 phân tử sử dụng quick sort sẽ tối ưu hơn. Vậy nên mỗi một bài toán sẽ só 1 phương pháp tối ưu riêng, cần phải biết lựa chọn phương pháp phù hợp để tối ưu hóa bài toán.

**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:**

**merge sort**

* *Ý tưởng của thuật toán merge sort:*

Giống như Quick 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:

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)

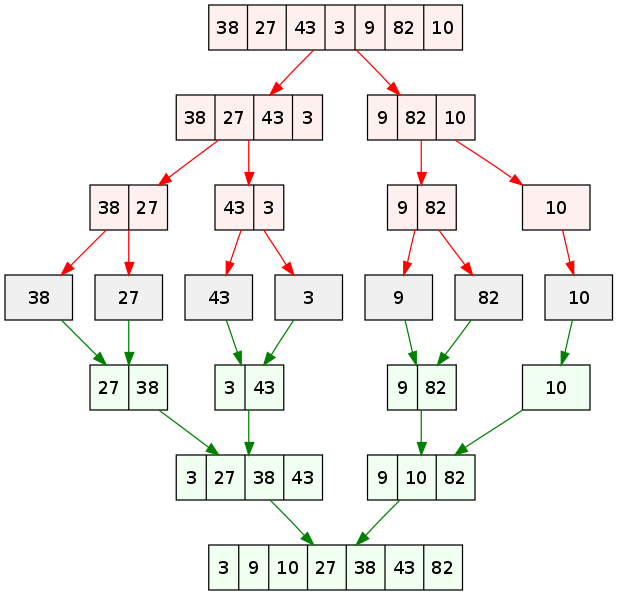
     4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):

             merge(arr, l, m, r)

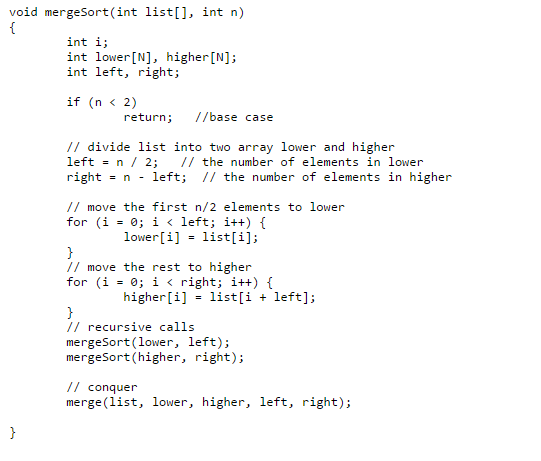
* *Cách hàm merge hoạt động khi gộp hai mảng con :*

Với trường hợp khi 2 mảng con chỉ có 1 phần tử, ta chỉ việc xem phần tử nào nhỏ hơn và đẩy lên đầu, phần tử còn lại đặt phía sau. Do vậy, các mảng con cần gộp lại có tính chất luôn được sắp tăng dần.

* *Ví dụ:*



* *Chương trình:*



* *Độ  phức tạp thuật toán:*
* Trường hợp tốt: O(nlog(n))
* Trung bình: O(nlog(n))
* Trường hợp xấu: O(nlog(n))