BÀI TẬP TUẦN 4

Môn học: Cấu Trúc Dữ Liệu và Giải Thuật.
GV hướng dẫn thực hành: Nguyễn Khánh Toàn (ktoan271199@gmail.com).
Nội dung chính:

- Các thuật toán sắp xếp với thời gian chạy trung bình là $O(n^2)$ với n là kích thước của mảng.

Thời gian thực hiện: **1.5** tuần.

Bài tập được biên soạn lại và tham khảo từ tài liệu thực hành môn Cấu Trúc Dữ Liệu và Giải Thuật của bộ môn Công Nghệ Tri Thức, trường Đại Học Khoa Học Tự Nhiên TPHCM. Trân trọng cảm ơn quý Thầy Cô của bộ môn.

1 Cài đặt và so sánh các thuật toán sắp xếp

1.1 Sắp xếp đổi chỗ trực tiếp (interchange sort)

Ý tưởng: Để sắp xếp một mảng tăng dần, thuật toán sắp xếp đổi chỗ trực tiếp xét từng cặp phần tử trong mảng, nếu cặp nào bị sai thứ tự thì chúng ta hoán đổi giá trị của chúng với nhau. Thuật toán được mô tả bằng mã giả như sau:

```
Với mỗi vị trí i = 0 -> n-2:
    Với mỗi vị trí j = i+1 -> n-1:
        Nếu arr[i] > arr[j] thì:
            Hoán đổi giá trị arr[i] và arr[j]]
```

Hình 1: Sắp xếp đổi chỗ trực tiếp (interchange sort).

Trong đó *arr* là mảng số có n phần tử. Với ý tưởng đó, các bạn thực hiện cài đặt lại thuật toán sắp xếp đổi chỗ trực tiếp.

1.2 Sắp xếp chèn (insertion sort)

Ý tưởng: Để sắp xếp một mảng tăng dần, thuật toán sắp xếp chèn từng bước lấy một phần tử chèn vào phần đã được sắp xếp tăng dần trước đó. Thuật toán này có thể được mô tả bằng mã giả như sau:

Trong đó *arr* là mảng số có n phần tử. Với ý tưởng đó, các bạn thực hiện cài đặt lại thuật toán sắp xếp chèn. Một số thuật toán cải tiến:

```
Với mỗi vị trí i = 1 -> n-1:
    x = arr[i]
    pos = i - 1
    Trong khi pos >= 0 và x < arr[pos] thì:
        arr[pos + 1] = arr[pos]
        pos = pos - 1
    arr[pos + 1] = x
```

Hình 2: Sắp xếp chèn (insertion sort).

- **Sắp xếp chèn với tìm kiếm nhị phân (binary insertion sort,** tham khảo wikipedia): Để tìm vị trí thích hợp khi chèn phần tử hiện tại vào mảng trước đó đã được sắp xếp, chúng ta hoàn toàn có thể sử dụng thuật toán tìm kiếm nhị phân để giảm bớt số lượng thao tác so sánh. Các bạn thay đổi mã nguồn của thuật toán sắp xếp chèn và tích hợp kĩ thuật tìm kiếm nhị phân như mô tả.
- Với phiên bản gốc của thuật toán sắp xếp chèn, chúng ta đưa tuần tự một phần tử về đúng vị trí của nó ở trong mảng đã sắp xếp (bước nhảy bằng 1). Tuy nhiên trong trường hợp vị trí đúng của nó ở rất xa so với vị trí hiện tại (chẳng hạn mảng đang được sắp ngược) thì sao? Chính vì thế Shell Sort ra đời để giải quyết vấn đề này. Thuật toán Shell Sort chia mảng thành các mảng con mà các phần tử trong mỗi mảng cách nhau một khoảng bằng h, sau đó sắp xếp mỗi mảng con bằng Insertion Sort, rồi tiếp tục giảm h để sắp xếp cho đến khi h giảm còn 1. Lưu ý rằng với thuật toán Shell Sort, bước nhảy thay vì bằng 1 như phiên bản thuật toán gốc đã tăng lên thành h ở mỗi lần duyệt.

```
void ShellSort(int arr[], int n) {
    for (int gap = n / 2; gap > 0; ...) { // khoảng cách có dạng n/2, n/4,... 1
        for (int i = gap; ...; i++) { // sắp xếp các mảng con bằng Insertion Sort
            int x = arr[i];
            int j = ... - gap;
            while (j >= 0 && x ... arr[j]) {
                arr[j ... gap] = arr[j];
                j -= ...;
            }
            arr[j ... gap] = x;
        }
    }
}
```

Hình 3: Sắp xếp shell (shell sort).

Các bạn hoàn thành mã nguồn của thuật toán Shell Sort như hình ảnh và ý tưởng đã gợi ý ở trên.

1.3 Sắp xếp chọn (selection sort)

Ý tưởng: Để sắp xếp một mảng tăng dần, thuật toán sắp xếp chọn liên tục tìm kiếm phần tử nhỏ nhất trong phần còn lại của mảng rồi hoán đổi nó với phần tử đầu tiên. Thuật toán này có thể được mô tả bằng mã giả như sau:

```
Với mỗi vị trí i = 0 -> n-2:
    min_index = i
    Với mỗi vị trí j = i+1 -> n-1:
        Nếu arr[j] < arr[min_index] thì:
            min_index = j
    Hoán đổi arr[i] và arr[min_index]
```

Hình 4: Sắp xếp chọn (selection sort).

Trong đó *arr* là mảng số có n phần tử. Với ý tưởng đó, các bạn thực hiện cài đặt lại thuật toán sắp xếp chọn.

1.4 Sắp xếp nổi bọt (bubble sort)

Ý tưởng: Để sắp xếp một mảng tăng dần, thuật toán sắp xếp chọn liên tục hoán đổi giá trị các cặp phần tử **kề nhau** nếu chúng sai thứ tự cho đến khi mảng được sắp xếp hoàn toàn. Ở mỗi lượt duyệt phần tử nhỏ nhất (hoặc lớn nhất) sẽ được 'nổi bọt' lên đầu hoặc cuối mảng. Thuật toán này có thể được mô tả bằng mã giả như sau:

```
Với mỗi vị trí i = 0 -> n-2:
    Với mỗi vị trí j = n-1 -> i+1:
        Nếu arr[j - 1] > arr[j] thì:
            Hoán đổi giá trị arr[j] và arr[j - 1]
```

Hình 5: Sắp xếp nổi bọt (bubble sort).

Trong đó *arr* là mảng số có n phần tử. Với ý tưởng đó, các bạn thực hiện cài đặt lại thuật toán sắp xếp nổi bọt.

Thay vì lúc nào cũng chỉ đẩy phần tử lớn nhất hoặc nhỏ nhất lên như Bubble Sort, thì Shaker Sort luân phiên đẩy phần tử nhỏ lên trên và phần tử lớn xuống dưới. Không những thế, Shaker Sort còn phát hiện được khi mảng đã được sắp xếp hoàn toàn thì dừng ngay thuật toán, giúp cho độ phức tạp trong trường hợp tốt nhất chỉ còn $O(n)$ thay vì $O(n^2)$ của Bubble Sort. Dưới đây là mã giả của thuật toán Shaker Sort, các bạn tham khảo và cài đặt hoàn thiện:

```
left = 0
right = n - 1
stop = 0
Trong khi left < right:
    // đầu tiên là đẩy phần tử lớn về cuối mảng
    Với mỗi vị trí i = left -> right-1:
        Nếu arr[i] > arr[i + 1] thì:
            Hoán đổi giá trị arr[i] và arr[i + 1]
            stop = i
    right = stop

    // sau đó đẩy phần tử nhỏ về đầu mảng
    Với mỗi vị trí i = right -> left+1:
        Nếu arr[i] < arr[i - 1] thì:
            Hoán đổi giá trị arr[i] và arr[i - 1]
            stop = i
    left = stop
```

Hình 6: Sắp xếp shaker (shaker sort).

1.5 Tính ổn định của thuật toán sắp xếp

Bạn được cho danh sách của sinh viên, mỗi sinh viên bạn biết được MSSV và điểm trung bình (trên thang điểm 10). Bạn hãy thực hiện sắp xếp lại danh sách sinh viên này theo điểm số tăng dần, và nếu có nhiều sinh viên cùng bằng điểm nhau thì người nào có mặt trong danh sách trước được đứng trước.

Ví dụ về mẫu **đầu vào** của bài toán:

5

1512431 10

1613234 9.0

1514920 9.0

1712000 7.0

1812000 4.0

Đầu ra: 1812000, 1712000, 1613234, 1514920, 1512431.

Lưu ý: Đối với mảng trong các bài tập trên các bạn có thể sử dụng *vector*, tuy nhiên ở các bài tập không được sử dụng các thư viện sắp xếp có sẵn.

Sau bài tập này, các bạn nên làm bảng thống kê về các thuật toán sắp xếp đã học với các thông tin như: độ phức tạp tệ nhất, trung bình và tốt nhất (tự đưa ra được ví dụ cho mỗi trường hợp); có thỏa mãn tính ổn định và tính in-place không? Tính in-place tạm dịch là tính tại chỗ, tức là không dùng thêm bộ nhớ quá nhiều để thực hiện sắp xếp (thường là quy ước nhỏ hơn $O(N)$ về mặt bộ nhớ với N là kích thước mảng).

Hết