Cần sắp xếp theo GPA giảm dần. Nhiệm vụ:

1. Viết hàm sắp xếp theo GPA.
2. Prompt AI giải thích logic thuật toán (bubble sort/selection sort).
3. Sinh test case: GPA bằng nhau, GPA tăng dần sẵn, GPA ngẫu nhiên.

Output:

```
Danh sách sau khi sắp xếp:
3 Chi 19 9.0
1 An 20 8.0
2 Bình 21 7.5
```

Prompt :

"Hãy đóng vai một chuyên gia lập trình C, hãy viết một chương trình quản lý sinh viên chuyên nghiệp đáp ứng chính xác các yêu cầu sau:

1. Cấu trúc dữ liệu & Khởi tạo:

- Định nghĩa `struct Student` gồm: `id` (int), `name` (chuỗi), `age` (int), và `gpa` (float).
- Khởi tạo sẵn một mảng chứa 3 sinh viên mẫu.
- Viết hàm hiển thị danh sách sinh viên dưới dạng bảng có tiêu đề cột đẹp mắt.

2. Quản lý & Nhập liệu:

- Viết chức năng cho phép người dùng nhập thêm tối đa 5 sinh viên từ bàn phím.
- Viết hàm `calculateAverageGPA` để tính và trả về điểm trung bình của toàn bộ danh sách.

3. Xử lý Tệp tin (File I/O):

- Viết hàm `saveToFile` lưu toàn bộ danh sách hiện tại vào tệp `students.txt`.
- Viết hàm `readFromFile` để đọc dữ liệu từ tệp `students.txt`, lưu vào mảng và hiển thị ra màn hình.
- Yêu cầu Output: Định dạng in ra phải giống hệt trong ảnh mẫu (ví dụ: `1 An 20 8.0`), trong đó điểm GPA luôn hiển thị chính xác 1 chữ số thập phân.

4. Tìm kiếm & Sắp xếp (Logic nâng cao):

- Tìm kiếm: Viết hàm `findStudentByID` kèm theo Docstring (mô tả chức năng, tham số, giá trị trả về).
- Sắp xếp: Viết hàm `sortByGPA` để sắp xếp sinh viên theo điểm giảm dần.
- Giải thích thuật toán: Hãy giải thích ngắn gọn logic của thuật toán sắp xếp bạn đã sử dụng (Bubble Sort hoặc Selection Sort).

5. Dữ liệu giả lập & Kiểm thử (Testing):

- Sinh dữ liệu: Hãy tạo sẵn 10 đối tượng sinh viên giả lập (dummy data) với tên và GPA khác nhau để phục vụ kiểm thử.
- Test cases tìm kiếm: Thực hiện kiểm thử hàm tìm kiếm với các trường hợp: ID tồn tại, ID không tồn tại, và ID là số âm.
- Test cases sắp xếp: Thực hiện kiểm thử với các trường hợp: Danh sách có GPA bằng nhau, danh sách đã được sắp xếp sẵn, và danh sách có GPA ngẫu nhiên.
- Trường hợp đặc biệt: Viết mã xử lý và in thông báo lỗi nếu danh sách sinh viên đang rỗng.

6. Yêu cầu về mã nguồn:

- Sử dụng `#define` để quản lý hằng số (kích thước mảng, tên file).
- Xử lý lỗi đầy mảng hoặc lỗi không mở được file.
- Mã nguồn phải sạch, có chú thích rõ ràng bằng tiếng Việt."

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// ======= ĐỊNH NGHĨA HẰNG SỐ =======
#define MAX_STUDENTS 18                 // 3 mẫu + 5 nhập tay + 10 giả lập
#define MAX_INPUT 5                     // Tối đa 5 sinh viên nhập
#define FILE_NAME "students.txt"

// ====== ĐỊNH NGHĨA STRUCT =======
typedef struct {
    int id;
    char name[50];
    int age;
    float gpa;
} Student;

// ====== HÀM IN ĐẸP ======
void printHeader() {
    printf("\n%-10s %-30s %-10s %-10s\n", "ID", "Ten", "Tuoi", "GPA");
    printf("-------------------------------------------------------------------------\n");
}
void printStudent(const Student s) {
    printf("%-10d %-30s %-10d %-10.1f\n", s.id, s.name, s.age, s.gpa);
}
void printStudentList(Student students[], int count) {
    if (count <= 0) {
        printf("(Danh sach sinh vien rong!)\n");
        return;
    }
    printHeader();
    for (int i = 0; i < count; i++)
        printStudent(students[i]);
    printf("\n");
}

// ====== KHỞI TẠO MẢNG SINH VIÊN ======
void initSampleStudents(Student arr[], int* n) {
    arr[0].id = 1; strcpy(arr[0].name, "An");    arr[0].age = 20; arr[0].gpa = 8.0;
    arr[1].id = 2; strcpy(arr[1].name, "Binh");  arr[1].age = 21; arr[1].gpa = 7.5;
    arr[2].id = 3; strcpy(arr[2].name, "Cuong"); arr[2].age = 19; arr[2].gpa = 8.5;
    *n = 3;
}
```

```c
// Sinh dữ liệu giả lập đa dạng
void generateDummyStudents(Student arr[], int startIdx, int* n) {
    const char* names[10] = {"Dung","Hoa", "Khanh", "Lam", "Mai", "Nhan","Oanh","Phuc","Quang","Son"};
    int ages[10] = {20,22,18,21,20,23,19,21,20,20};
    float gpas[10] = {6.7,7.3,5.9,9.1,8.3,5.8,7.0,9.6,6.0,8.9};
    for (int i = 0; i < 10; i++) {
        arr[startIdx+i].id = startIdx + i + 1;
        strcpy(arr[startIdx+i].name, names[i]);
        arr[startIdx+i].age = ages[i];
        arr[startIdx+i].gpa = gpas[i];
    }
    *n = startIdx + 10;
}

// ====== HÀM NHẬP THÊM SINH VIÊN ======
int inputStudents(Student students[], int currCount, int maxCount) {
    int nAdd = 0;
    printf("\nNhap so luong sinh vien muon them (toi da %d): ", maxCount-currCount);
    scanf("%d", &nAdd);
    if (nAdd < 1 || currCount >= maxCount) {
        printf("Khong them sinh vien moi.\n");
        return 0;
    }
    if (nAdd > maxCount - currCount) nAdd = maxCount - currCount;
    for (int i = 0; i < nAdd; i++) {
        printf("\nNhap thong tin sinh vien thu %d:\n", currCount+i+1);
        printf("ID: "); scanf("%d", &students[currCount+i].id);
        printf("Ten: "); scanf("%49s", students[currCount+i].name);
        printf("Tuoi: "); scanf("%d", &students[currCount+i].age);
        printf("GPA: "); scanf("%f", &students[currCount+i].gpa);
    }
    return nAdd;
}

// ====== TÍNH GPA TRUNG BÌNH ======
float calculateAverageGPA(Student students[], int count) {
    if (count <= 0) return -1.0f;
    float sum = 0.0f;
    for (int i = 0; i < count; i++)
        sum += students[i].gpa;
    return sum / count;
}
```

```c
// ====== XỬ LÝ TỆP TIN ======
void saveToFile(Student students[], int count, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (!file) {
        printf("Loi: Khong mo duoc file de ghi!\n");
        return;
    }
    for (int i = 0; i < count; i++)
        fprintf(file, "%d %s %d %.1f\n", students[i].id, students[i].name, students[i].age, students[i].gpa);
    fclose(file);
    printf("Da luu %d sinh vien vao file %s\n", count, filename);
}
// Đọc danh sách từ file, trả về số lượng sinh viên đọc được
int readFromFile(Student students[], int maxCount, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        printf("Loi: Khong mo duoc file de doc!\n");
        return 0;
    }
    int count = 0;
    while (count < maxCount &&
           fscanf(file, "%d %49s %d %f", &students[count].id, students[count].name,
                  &students[count].age, &students[count].gpa) == 4) {
        count++;
    }
    fclose(file);
    return count;
}
// In danh sách dạng thô từ file
void printListFromFile(Student students[], int count) {
    printf("Danh sach doc tu file:\n");
    if (count <= 0) {
        printf("(Khong co du lieu)\n");
        return;
    }
    for (int i = 0; i < count; i++)
        printf("%d %s %d %.1f\n", students[i].id, students[i].name, students[i].age, students[i].gpa);
}
```

```c
// ====== TÌM KIẾM & SẮP XẾP ======
/**
 * Tìm kiếm sinh viên theo ID trong mảng.
 * @param students Mảng các sinh viên.
 * @param count Số lượng sinh viên trong mảng.
 * @param id ID của sinh viên cần tìm.
 * @return Con trỏ đến sinh viên nếu tìm thấy, NULL nếu không tìm thấy hoặc ID không hợp lệ.
 */
Student* findStudentByID(Student students[], int count, int id) {
    if (id < 0 || count <= 0) return NULL;
    for (int i = 0; i < count; i++)
        if (students[i].id == id) return &students[i];
    return NULL;
}

// Sắp xếp sinh viên theo GPA giảm dần (Selection Sort).
void sortByGPA(Student students[], int count) {
    // Thuật toán Selection Sort:
    // Lặp qua từng vị trí, chọn phần tử lớn nhất dời lên đầu (vị trí đang duyệt).
    for (int i = 0; i < count-1; i++) {
        int max_idx = i;
        for (int j = i+1; j < count; j++)
            if (students[j].gpa > students[max_idx].gpa)
                max_idx = j;
        if (max_idx != i) {
            // Hoán đổi
            Student tmp = students[i];
            students[i] = students[max_idx];
            students[max_idx] = tmp;
        }
    }
    // Selection Sort là thuật toán đơn giản, dễ triển khai khi số lượng vừa phải.
    // Chọn điểm GPA cao nhất ở mỗi vòng và đưa về đầu dãy.
}
```

```c
// ====== MAIN + KIỂM THỬ ======
int main() {
    Student students[MAX_STUDENTS];
    int count = 0;

    // 1. Khởi tạo 3 sinh viên mẫu
    initSampleStudents(students, &count);

    // 2. Sinh thêm 10 dữ liệu giả lập:
    generateDummyStudents(students, count, &count);

    printf("\n=== DANH SACH BAN DAU (3 sinh vien + 10 dummy) ===\n");
    printStudentList(students, count);

    // 3. Nhập thêm tối đa 5 sinh viên từ bàn phím
    int nAdded = inputStudents(students, count, MAX_STUDENTS);
    count += nAdded;

    printf("\n=== DANH SACH SINH VIEN SAU KHI THEM ===\n");
    printStudentList(students, count);

    // 4. Tính GPA trung bình
    float avg = calculateAverageGPA(students, count);
    if (avg < 0)
        printf("(Khong the tinh GPA trung binh: danh sach rong)\n");
    else
        printf("GPA trung binh cua %d sinh vien: %.2f\n", count, avg);

    // 5. Lưu file
    saveToFile(students, count, FILE_NAME);

    // 6. Đọc và in lại file
    Student loadArr[MAX_STUDENTS];
    int nFromFile = readFromFile(loadArr, MAX_STUDENTS, FILE_NAME);
    printf("\n=== DANH SACH DOC TU FILE ===\n");
    printListFromFile(loadArr, nFromFile);
```

```c
// ====== KIỂM THỬ CHỨC NĂNG TÌM KIẾM ======
printf("\n=== TEST CASES: TIM KIEM SINH VIEN THEO ID ===\n");
// a. ID tồn tại
int existID = students[2].id; // chắc chắn có
Student* found = findStudentByID(students, count, existID);
printf("Test case 1: Tim ID = %d (ton tai):\n", existID);
if (found)
    printf("Tim thay: ID=%d, Ten=%s, Tuoi=%d, GPA=%.1f\n", found->id, found->name, found->age, found->gpa);
else
    printf("Khong tim thay sinh vien co ID = %d\n", existID);
// b. ID không tồn tại
int notExistID = 9999;
found = findStudentByID(students, count, notExistID);
printf("Test case 2: Tim ID = %d (khong ton tai):\n", notExistID);
if (found)
    printf("Tim thay: ID=%d, Ten=%s, Tuoi=%d, GPA=%.1f\n", found->id, found->name, found->age, found->gpa);
else
    printf("Khong tim thay sinh vien co ID = %d\n", notExistID);
// c. ID âm
int negID = -5;
found = findStudentByID(students, count, negID);
printf("Test case 3: Tim ID = %d (so am):\n", negID);
if (found)
    printf("Tim thay: ID=%d, Ten=%s, Tuoi=%d, GPA=%.1f\n", found->id, found->name, found->age, found->gpa);
else
    printf("ID = %d khong hop le (so am hoac khong ton tai)\n", negID);
```

```c
// ====== KIỂM THỬ SẮP XẾP THEO GPA ======
printf("\n=== TEST CASES: SAP XEP SINH VIEN THEO GPA GIAM DAN ===\n");
// a. GPA đã bằng nhau (test mọi phần tử = 7.0)
Student test1[5] = { {1,"A",19,7.0}, {2,"B",20,7.0},{3,"C",21,7.0},{4,"D",22,7.0},{5,"E",22,7.0}};
printf("\nTest case 1: Tat ca GPA = 7.0\n");
printStudentList(test1,5);
sortByGPA(test1,5);
printf("Sau khi sap xep:\n");
printStudentList(test1,5);

// b. GPA đã giảm dần (không thay đổi gì sau sort)
Student test2[4] = { {1,"AA",21,9.0}, {2,"BB",20,8.0}, {3,"CC",22,7.0}, {4,"DD",20,5.0}};
printf("\nTest case 2: Danh sach da duoc sap xep giam dan san:\n");
printStudentList(test2,4);
sortByGPA(test2,4);
printf("Sau khi sap xep:\n");
printStudentList(test2,4);

// c. GPA ngẫu nhiên
Student test3[6] = { {7,"Y",20,5.5}, {8,"Z",21,8.2}, {9,"X",18,6.7}, {10,"U",20,9.0}, {11,"Q",22,7.8}, {12,"R",18,7.8}};
printf("\nTest case 3: Danh sach GPA ngau nhien:\n");
printStudentList(test3,6);
sortByGPA(test3,6);
printf("Sau khi sap xep:\n");
printStudentList(test3,6);

// ====== XỬ LÝ ĐẶC BIỆT: DANH SÁCH RỖNG ======
printf("\n=== TEST CASE: DANH SACH SINH VIEN RONG ===\n");
Student emptyArr[5];
int emptyCount = 0;
if (emptyCount <= 0)
    printf("- Danh sach hien tai dang rong. Moi chuc nang lien quan hien thi hoac xu ly duoc xu ly an toan.\n");
float avgEmpty = calculateAverageGPA(emptyArr, emptyCount);
if (avgEmpty < 0)
    printf("- GPA trung binh: Khong the tinh vi danh sach rong.\n");
else
    printf("- GPA: %.2f\n", avgEmpty);
Student* foundEmpty = findStudentByID(emptyArr, emptyCount, 1);
if (foundEmpty == NULL)
    printf("- Tim kiem: Khong tim thay sinh vien tren mang rong.\n");
else
    printf("- Tim thay bat ky (KHONG THE XAY RA)\n");
sortByGPA(emptyArr, emptyCount);
printf("- Sap xep: Da thu hien tren mang rong (ok).\n");
```

```c
    // ---- KẾT THÚC ----
    printf("\n--- CHUONG TRINH KET THUC! ---\n");
    return 0;
}
```