Giải thích hàm `quickSort` và `operator<`

Hàm `operator<`

Hàm `operator<` là một toán tử so sánh được nạp chồng trong lớp `sanPham`. Hàm này cho phép so sánh dựa trên giá sản phẩm (`gia`). Hàm sẽ trả về `true` nếu giá của đối tượng hiện tại nhỏ hơn giá của sản phẩm khác (`sp`), và `false` trong các trường hợp còn lại. Việc nạp chồng toán tử này giúp cho việc sắp xếp dễ dàng và trực quan hơn khi so sánh các đối tượng `sanPham` theo giá.

Mã nguồn:

```
Copy code
bool operator<(const sanPham &sp)
{
    return gia < sp.gia;
}
```

Giải thích:

- `bool operator<(const sanPham &sp)`: Hàm này được gọi khi sử dụng toán tử `<` để so sánh giữa hai đối tượng `sanPham`.
- `return gia < sp.gia;`: So sánh giá (`gia`) của đối tượng hiện tại với `sp`. Trả về `true` nếu giá của đối tượng hiện tại nhỏ hơn; ngược lại trả về `false`.

Hàm `quickSort`

Hàm `quickSort`, cùng với hàm phụ trợ `partition`, được sử dụng để sắp xếp các sản phẩm trong vector `danhSach` theo thứ tự tăng dần của giá. Thuật toán quicksort rất hiệu quả cho các tập dữ liệu lớn, và hoạt động bằng cách chọn một phần tử làm `pivot`, sau đó phân chia danh sách sao cho các phần tử nhỏ hơn `pivot` nằm bên trái và các phần tử lớn hơn `pivot` nằm bên phải. Quá trình này được thực hiện đệ quy trên các danh sách con cho đến khi toàn bộ danh sách được sắp xếp.

Hàm `partition`

Hàm `partition` có nhiệm vụ chọn `pivot` và sắp xếp các phần tử trong phạm vi từ `low` đến `high`. Sau khi phân chia, tất cả các phần tử nhỏ hơn `pivot` sẽ nằm bên trái và các phần tử lớn hơn sẽ nằm bên phải.

Mã nguồn:

```
int partition(int low, int high)
{
    double pivot = danhSach[high].getGia(); // Chọn phần tử cuối làm pivot
    int i = low - 1;
```

```
    for (int j = low; j < high; j++)
    {
        if (danhSach[j].getGia() < pivot)
        {
            i++;
            swap(danhSach[i], danhSach[j]);
        }
    }
    swap(danhSach[i + 1], danhSach[high]);
    return i + 1; // Trả về vị trí của pivot
}
```

Giải thích:

- `pivot = danhSach[high].getGia();`: Chọn `pivot` là giá của phần tử cuối trong phạm vi hiện tại.
- `i = low - 1;`: Khởi tạo biên để đánh dấu các phần tử nhỏ hơn `pivot`.
- Vòng `for` kiểm tra nếu mỗi phần tử có giá nhỏ hơn `pivot`. Nếu có, đổi chỗ phần tử đó với phần tử tại chỉ mục `i + 1`.
- Hàm trả về vị trí cuối cùng của `pivot`, tách danh sách thành hai phần để tiếp tục sắp xếp.

Hàm đệ quy `quickSort`

Hàm `quickSort` gọi lại chính nó để sắp xếp các phần tử trước và sau `pivot` được trả về từ hàm `partition`.

Mã nguồn:

```
void quickSort(int low, int high)
{
    if (low < high)
    {
        int pi = partition(low, high);
        quickSort(low, pi - 1); // Sắp xếp các phần tử trước pivot
        quickSort(pi + 1, high); // Sắp xếp các phần tử sau pivot
    }
}
```

Giải thích: - `if (low < high)`: Đảm bảo rằng vẫn còn phần tử cần sắp xếp. - `partition(low, high)`: Phân chia danh sách và trả về chỉ mục của `pivot`. - Hàm sau đó đệ quy gọi `quickSort` để sắp xếp các danh sách con ở hai phía của `pivot` cho đến khi danh sách con không còn phần tử để sắp xếp.

Các hàm này giúp sắp xếp danh sách sản phẩm theo giá tăng dần, hỗ trợ quản lý sản phẩm hiệu quả trong lớp `quanLySP`.