# Đọc file và ghi file

Chương trình sẽ đọc file Books\_rating.csv vào ghi dữ liệu vào 1 mảng lines: lines[n]

Với lines có cấu trúc Book: string ID, string text.

ID dùng để chứa các ID (cột đầu) để dùng cho việc sắp xếp.

text dùng để chứa các ký tự được đọc nguyên 1 dòng trong file csv.

1 dòng file file csv được tổ chức như sau:các dòng được ngăn cách bới dấu cách, các cột được ngăn cách bới dấu phấy( , ). Với các đoạn (1 cột trong 1 dòng) mà trong đó có dấu phẩy thì ở đầu và cuối đoạn ấy được cho thêm dấu ngoặc kép ( “” ), nếu trong đoạn đó vừa có dấu phẩy và dấu ngoặc kép thì ở đầu và cuối dấu ngoặc kép cho thêm dấu ngoặc kép ( “” ) nữa. Khi hết 1 dòng, file sẽ xuống hàng.

vd: trong file csv: 1 cột (trong 1 dòng) có ghi abc thì khi đọc sẽ là …,abc, …

1 cột có ghi zx,cv thì khi đọc sẽ là …,“zx,cv”, …

1 cột có ghi 2,3 “sd” thì khi đọc sẽ là …,“2,3 “”sd”””, …

Khi đọc, chương trình sẽ đọc nguyên 1 dòng bằng hàm getline(), các dòng được tách bởi xuống dòng (\n), mỗi dòng sẽ gán vào biến Lines[i].text tương ứng trong mảng Lines. Vì ID ở cột đầu tiên và dữ liệu ID trong file csv không có các trường đặc biệt như có dấu phẩy… nên chỉ đọc đơn giản là đọc từng ký tự từ trái sang của line cho đến khi gặp dấu phẩy thì dừng, sau gán đoạn ấy vào biến ID.

Trong quá trình nạp dữ liệu vào mảng lines, khi hoàn thành đọc 1 dòng, chương trình sẽ chuyển đến dòng tiếp theo và tiếp tục cho đến khi gặp điểm dừng: đến giới hạn của mảng hoặc đến cuối file csv.

Do line chứa hết các ký tự trong 1 dòng nên không cần điều chỉnh gì cả chỉ cần xuất ra, mỗi dòng thì thêm dấu cách.

Sau khi chạy thử có phát hiện ra vấn đề: sau mỗi dòng thêm dấu cách khi xuất file đó là vô tình tạo ra một dòng rỗng hay dòng NULL ở cuối file. Nếu chỉ xuất ra và đọc nội dung thì không có vấn đề, nhưng nếu dùng cho các tác vụ khác như sắp xếp (hàm quicksort) thì dòng rỗng được xem như là một dòng bình thường, tức là tổng số dòng sẽ tăng lên 1 và kết quả nếu sắp xếp tăng dần thì rỗng đó sẽ xuất hiện ở đầu gây ảnh hưởng tới kết quả. Để khắc phục vấn đề này, mỗi lần đọc một dòng cần kiểm tra thử dòng đó có phải dòng rỗng hay không, nếu có thì bỏ qua. Ngoài ra còn có cách đó là không tạo dòng rỗng ấy nhưng sẽ làm phức tạp hơn và khó đảm bảo rằng mọi file xuất ra không có dòng rỗng.

# Đặt giới hạn của mảng

Kết quả chạy trên máy với cấu hình RAM 16GB, CPU Ryzen 5 4600H.

Khi nạp dữ liệu vào mảng lines[RowLimit] với RowLimit là số lượng dòng trong 1 lần nạp, có cấu trúc string ID, string line đã được định nghĩa ở trên:

Sau đây là 3 hình khi lần lượt 100 ngàn, 200 ngàn, 500 ngàn dòng đầu tiên nạp vào mảng Lines đọc từ file Books\_rating.csv có 3 triệu dòng:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

Ta có thể thấy 100 ngàn dòng cần 165 MB và chạy trong 12 giây

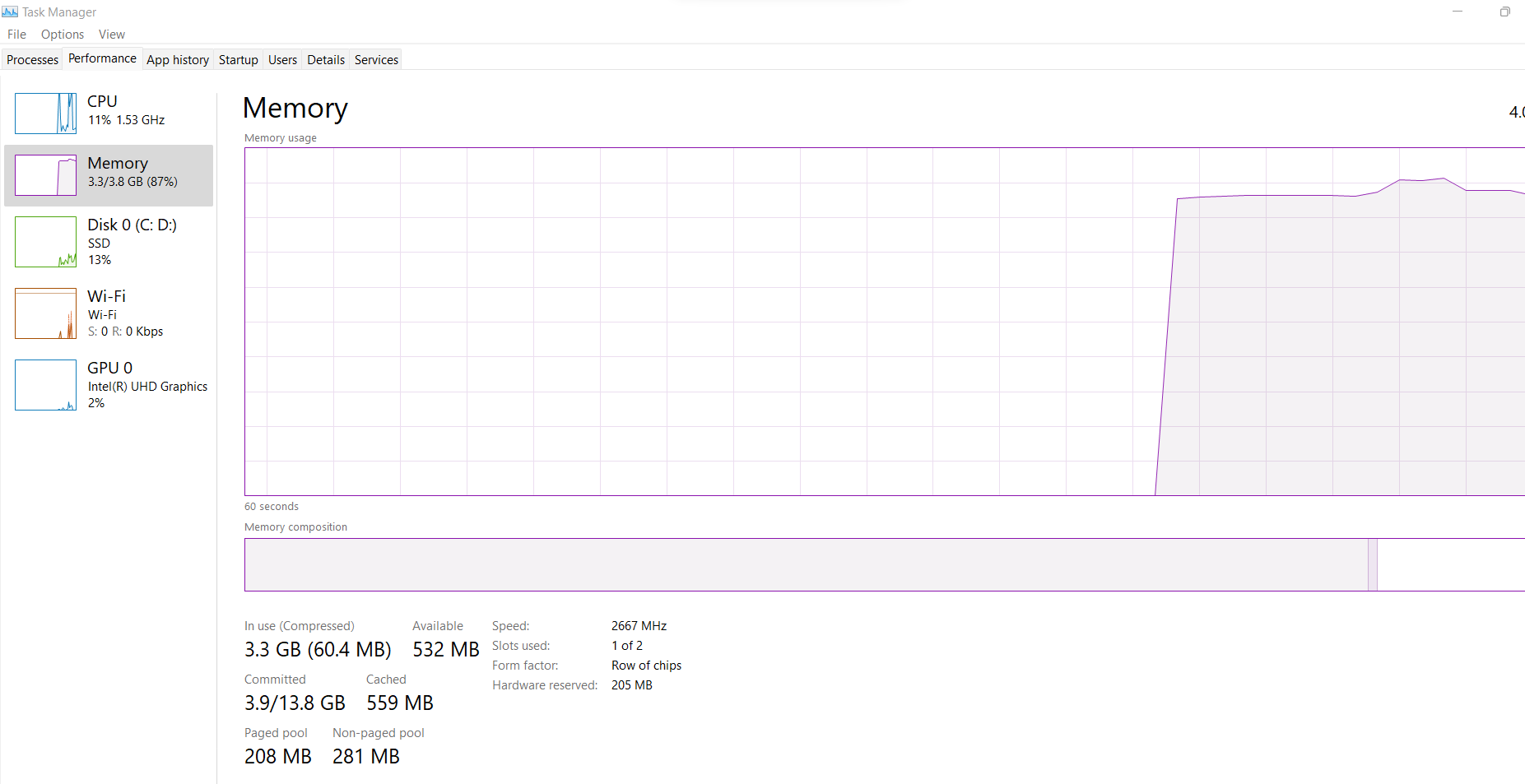
200 ngàn dòng cần 301 MB và chạy trong 23 giây

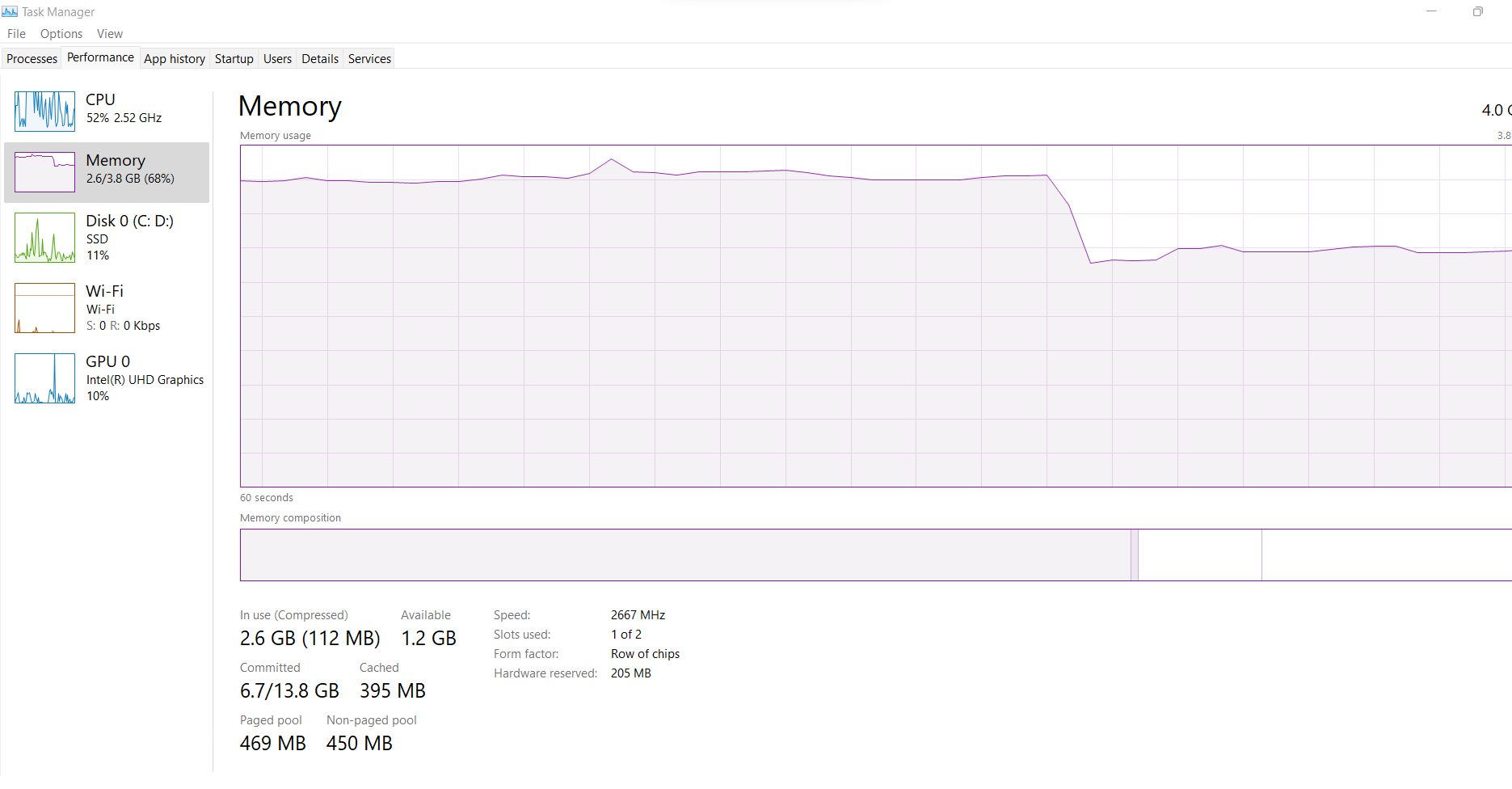
500 ngàn dòng cần 796 MB và chạy trong 58 giây

Tính ra trung bình 1 ngàn dòng cần cấp khoảng 1.5775 MB

1 GB chứa 1024 MB, suy ra 1 GB chứa được khoảng 649 ngàn dòng trong file Books\_rating.csv cho 1 lần nạp.

Sau đây là ảnh chụp từ máy của Đào Duy Anh với 4 GB RAM đang dùng Windows 11: ảnh đầu là khi vừa khởi động máy, ảnh sau là khi dùng các ứng dụng sau 1 thời gian và tắt các ứng dụng chiếm nhiều RAM đi.





Với máy 4 GB RAM chạy Windows 11 thì lý tưởng trống khoảng 1 GB nhưng thông thường là thấp hơn. Vậy nên 1 lần đọc file nạp vào mảng tối đa là 1 GB nhưng nên nạp tối đa 512 MB hoặc thấp hơn sẽ hợp lý vì còn có những tác vụ hay chương trình khác đang chạy (quicksort có đệ qui chiếm tương đối nhiều bộ nhớ). 512 MB có thể chứa khoảng 324.5 ngàn dòng cho 1 lần nạp làm tròn xuống là 300 ngàn dòng.

Với máy 8 GB RAM chạy Windows 11 cho là trống 4 GB RAM 1 lần nạp là 2596 ngàn dòng thì nên giảm xuống 2500 ngàn dòng (chiếm 3943 MB) hoặc 2000 ngàn dòng (chiếm 3155 MB).

Với máy 16 GB RAM chạy Windows 11 cho là trống 8 GB RAM thì hoàn toàn có thể chứa hết 3 triệu dòng cho 1 lần nạp, chiếm 4732.5 MB RAM.

Chú ý đây chỉ là ước lượng RAM chiếm trong việc nạp vào biến, chương trình còn nhiều tác vụ khác cũng có thể chiếm nhiều bộ như các thuật toán đệ qui (quick sort),… Vậy nên với máy càng ít RAM trống thì càng giảm số dòng hơn nữa để tránh bị đầy RAM.

Mục đích của đặt giới hạn của mảng này nhằm tránh RAM bị đầy khi chạy; ngoài ra giảm số lần tạo ra nhiều file con nhất có thể giúp giảm số lần merge các file lại với nhau từ đó cũng cải thiện thời gian chạy chương trình;...

# Tổ chức các file tạo ra

Mỗi lần dữ liệu được nạp vào mảng, sau khi nạp xong sẽ được sắp xếp. Khi sắp xếp xong sẽ xuất tạo ra 1 file mới. Quá trình lặp lại cho tới khi đọc hết file csv.

Các tên file tạo ra được lưu vào 1 mảng riêng có n phần tử cho tên là mảng Sorted, sau đó tạo thêm mảng khác có kích thước n – 1 để lưu tên file merge phục vụ cho việc merge file cho tên là mảng Merge, cách merge giống như merge sort. Cách đặt tên của file một cách tự động hoá bằng cách: name + to\_string(count) + “.csv” trong đó name là 1 string cho sẵn; count là 1 số int tăng lên 1 đơn vị mỗi lần tạo file, sau đó chuyển sang dạng string bằng hàm to\_string(); còn “.csv” để định dạng file tạo ra là csv.

\*tên file ám chỉ chính file đó trong mảng Sorted và mảng Merge, xoá tên file cũng có nghĩa là xoá file đó.

Cách chọn file của quá trình merge file: trước hết, phần tử đầu tiên và phần tử thứ hai của mảng Sorted tiến hành Merge ghi dữ liệu vào phần tử đầu tiên của mảng Merge, sau đó xoá 2 file đã được dùng để merge trong mảng Sorted. Sau đó tiến hành vòng lặp: Phần tử tiếp theo của mảng Sorted (lúc này là phần tử thứ ba) tiến hành merge với phần tử vừa được tạo (lúc này là phần tử đầu tiên) của mảng Merge, tạo ra phần tử mới (lúc này là phần tử thứ hai) của mảng Merge, sau đó xoá các file đã được dùng cho merge. Vòng lặp chạy cho đến khi mảng Sorted chạy đến phần tử cuối cùng (file phần tử cùng đã được xoá), cũng có nghĩa là file cửa phần tử cuối cùng trong mảng Merge là kết quả được sort từ file csv.

Vd:

Cho mảng Sorted: 0 1 2 3 4

Cho mảng Merge: 0 1 2 3

Đầu tiên Sorted[0] và Sorted[1] merge, ghi vào Merge[0], sau đó xoá Sorted[0] và Sorted[1]

Còn lại: Sorted: 2 3 4 và Merge: 0 1 2 3

Chạy vòng lặp: Sorted [2] và Merge[0] merge, ghi vào Merge[1], sau đó xoá Sorted[2] và Merge[0]

Sorted[3] và Merge[1] merge, ghi vào Merge[2], sau đó xoá Sorted[3] và Merge[1]

Sorted[4] và Merge[2] merge, ghi vào Merge[3], sau đó xoá Sorted[4] và Merge[2]

Đến file merge cuối cùng, dừng vòng lặp. Merge[3] là file kết quả

# Thuật toán sắp xếp

Có 2 thuật toán sắp xếp được sử dụng trong chương trình này: quick sort và merge sort, trong chương trình này sắp xếp tăng dần.

Quick sort được dùng trong việc sắp xếp dựa vào string ID của các phần tử trong mảng Lines[]. Quá trình này nằm trong giai đoạn sau khi nạp vào mảng Lines đọc từ file csv và trước khi ghi vào file trong mảng Sorted.

Đối với merge sort được sử dụng vì merge sort có khả năng sắp xếp các mảng rời lại với nhau được, mảng lúc này tương đương với các file được tạo ra trong mảng Sorted. Điều kiện của việc merge này là các mảng ấy phải được sắp xếp vậy nên cần sắp xếp dữ liệu các file trong mảng Sorted trước khi merge lại.

Lí do không dùng luôn merge sort vì merge sort có dùng mảng phụ và dung lượng tạo ra có thể chiếm nhiều hơn 2 - 3 lần so với việc nạp vào mảng lines (1024 MB - 1536 MB so với 512 MB khi nạp 300000 dòng). Các thuật toán khác có dùng mảng phụ cũng không phù hợp, vì vậy mới dùng thuật toán chỉ sắp xếp trên chính mảng đó thì có các thuật toán như: selection sort, bubble sort,… trong đó quick sort phù hợp hơn vì có độ phức tạp là O(nlog(n)) so với O(n^2). (Để giảm thiểu gặp trường hợp xấu nhất, lấy 3 phẩn tử đầu, ở giữa, cuối để chọn pivot cho quick sort.)

# Các chi tiết nhỏ khác

* Dòng đầu của file csv chứa định nghĩa của các cột vậy nên khi đọc và nạp, bỏ qua nó; nhưng khi file kết quả cuối cùng được tạo thì chèn thêm dòng đó ở đầu file, có cách dòng ở cuối dòng.
* Khi chạy cần đảm bảo ổ cứng cần trống ít nhất 2 lần file cần sắp xếp (tổng dung lượng của 3 file khi chưa thực hiện việc xoá: phần tử cuối cùng của mảng sorted, phần tử kế cuối và phần tử cuối cùng của mảng merge)
* Cẩn thận khi dùng mảng lines được nạp từ file, đảm bảo khi dùng xong cần xoá/làm trống mảng đó trước khi nạp tiếp. Nếu không sẽ bị tràn bộ nhớ.
* Giữa các máy và giữa các lần chạy, kết quả dữ liệu:“1 ngàn dòng được nạp vào mảng lines chiếm 1.5775 MB RAM” và thời gian chạy có thể khác nhau. Vì vậy nên khi tính ra kết quả RowLimit, nên giảm thêm nữa phòng trường hợp 1 lần đọc 1 ngàn dòng chiếm nhiều RAM hơn.
* Chương trình chưa có tự động hoá trong việc xác định phần cứng, lượng RAM trống, cũng như tự tính trung bình 1 ngàn dòng chiếm bao nhiêu RAM để tính toán số dòng trong 1 lần nạp mảng lines. Tuỳ vào dòng được nạp vào mà dung lượng RAM sử dụng cũng sẽ khác nhau.

# Các hàm và cấu trúc

Struct Book: string ID; string text;

Int partition(Book\*& arr, int low, int high): thuộc hàm phụ vụ cho quicksort.

void quickSort(Book\*& arr, int low, int high): Hàm sắp xếp quicksort.

void writeSmallFile(Book\*& Lines, int& end, vector<string>& Sorted, int& count): Hàm viết các file sau khi sắp xếp.

void writeSmallMergeFile(string output, string input1, string input2, bool lastFile): Hàm viết các file sau khi merge lại với nhau.

void ReadAndCreateFile(string fileName, vector<string>& Sorted, vector<string>& Merge): Hàm đọc và tạo các file nhỏ nếu RowLimit nhỏ hơn tổng số dòng, các file trước khi ghi được sắp xếp bằng quicksort.

void MergeFile(vector<string> Sorted, vector<string> Merge): Hàm thao tác merge các file

void Run(): Hàm chạy sắp xếp file

void runResultFile(): Hàm chạy xuất file Result.csv từ file đã sắp xếp với số dòng được giới hạn

int main(): Hàm chạy chính

Ngoài ra có các lênh cout và hàm phục vụ cho việc xác định giai đoạn chương trình và cho biết thời gian chạy.

# Kết quả

Graphical user interface, text, application

Description automatically generated

So sánh kích thước, đến kích cỡ bytes là như nhau cho thấy nội dung ít nhất giống nhau, chưa xác nhận được số dòng có thêm hay mất không. Nội dung thì trong file Result.csv là kết quả 1000 dòng đầu tiên của file Books\_rating\_Sorted.csv.

Lần chạy 1500 ngàn dòng cho 1 lần nạp:

Quá trình được thực hiện trên máy 16 GB RAM, CPU Ryzen 5 4600H, có 1 số tác vụ chạy ngầm nhưng khi chạy CPU chưa dùng quá 25%, RAM trước khi chạy trống hơn 6GB. Thời gian chạy là 1097 giây tức 18 phút 17 giây.

Quan sát thấy: khi đặt giới hạn là 1500 ngàn dòng thì 1 lần nạp chiếm tới khoảng 2GB và cũng là giai đoạn tốn thời gian nhất khi qua quá trình sort và ghi file nhanh hơn nhiều. Trong quá trình sort, tuy hàm có đệ quy nhiều lần nhưng không chiếm RAM đáng kể so với quá trình nạp. Mỗi lần ghi file xong, trong hàm sẽ giải phóng mảng động đi và lượng RAM sử dụng cũng giảm theo. Merge thì vừa nạp vừa ghi file xen kẽ nhưng dựa vào suy diễn trước, việc đọc và nạp vào biến chiếm thời gian là chủ yếu. Khi chạy các file được tách ra có tổng kích thước gần bằng file ban đầu và khi merge tổng kích ngày càng tăng gấp 3 lần khi càng về giai đoạn merge cuối.

Vậy nên khi sort file csv này cần ít nhất 6 GB ổ đĩa trống, nếu không muốn tách file cần ít nhất 4-4.5 GB RAM trống nhưng nhờ đó bỏ qua giai đoạn merge giúp cải thiện thời gian chạy nhiều hơn.

Còn nhiều chỗ có thể thay đổi như quá trình merge file áp dụng buffer nạp 1 lần rồi dùng đến khi hết rồi lại nạp tiếp (nhanh hay chậm hơn thì không rõ), hay thay đổi cách nạp file nhanh hơn nhưng không gây ảnh hưởng nhiều các giai đoạn sau khiến thời gian chạy lâu hơn.