Các bạn luôn phải nhớ rằng thường với mấy dạng bài kiểu này lúc nào cũng không muốn ta nhìn ra được hướng giải quyết thực sự của nó, mà cố tình đánh lạc hướng chúng ta khỏi hướng làm tốt nhất để giải quyết nó. Thậm chí cả những test ví dụ còn cố tình đưa ra những trường hợp mà làm chúng ta nhìn vào đó mà lầm tưởng đi xử lý theo 1 hướng khác với hướng giải quyết tối ưu thực sự, nên các bạn phải tỉnh táo mà nhìn kỹ bản chất vấn đề nhé ☺️

Ví dụ với bài này theo như đề cố tình lái chúng ta theo hướng làm đó là vét cạn xét tất cả các trường hợp hoán vị các số để sau cùng tìm ra được số lớn nhất. Thì với N số ta sẽ có N! hoán vị. Ví dụ như trong đề cho 3 số thì có 3! = 6 hoán vị. Nhưng rõ ràng ta nhìn ra ngay đây sẽ không thể là cách làm tốt vì phép tính giai thừa sẽ mang đến kết quả rất lớn. Ví dụ đề cho N tối đa có thể lên đến 100 thì các bạn phải biết 100! lớn lắm độ dài đến 158 chữ số thì quá kinh khủng sẽ không thể nào xét hết nổi. Thực ra cỡ 12! = 479,001,600 hoán vị thì may ra còn xét được hết trong phạm vi giới hạn 1 giây (ngưỡng (3 đến 5)\*10^7 hay tối đa 10^8 ) nhưng lên cỡ 13! là ra 6 tỷ hơn là lúc này nếu xét hết là vượt quá giới hạn rồi. Vậy nên đề mới cho thêm dữ kiện sẽ có 50% test case có N <= 10 để dành cho bạn nào làm theo cách này vớt vát được 50% điểm. Còn nếu bạn nào muốn giải quyết trọn vẹn 100% điểm thì chắc chắn không thể chọn cách này được.

Thực ra kể cả nếu bài này mà có sửa lại ngưỡng giới hạn N tối đa chỉ là 10 chẳng hạn thì thực ra cách vét cạn cũng vẫn không nên làm vì nó thực sự phức tạp hơn cách làm tối ưu mà chịu khó nhìn thêm các bạn sẽ thấy được ngay. Cách làm vét cạn sẽ dẫn các bạn đến bài toán phải xét in ra lần lượt các hoán vị số từ 1 đến N và đây cũng là bài toán kinh điển mà nếu các bạn không khéo lại xử lý rất phức tạp vấn đề. Nói chung là với dạng bài này nếu chọn giải quyết theo cách vét cạn là các bạn chỉ có từ chết đến bị thương thôi 😂các bạn phải cố gắng động não tìm một hướng xử lý tốt hơn, triệt để hơn cho nó. Thì dưới đây mình sẽ phân tích để các bạn thấy rõ.

Hãy thử suy nghĩ: Nếu giả sử các giá trị a[i] đề cho không phải nằm trong đoạn từ 1 —> 10^5 mà chỉ nằm trong đoạn từ 1 —> 9 tức là nó chỉ có 1 chữ số thôi. Nếu như thế bài này giải quyết có phải rất đơn giản không? Chỉ là đi sắp xếp mảng giảm dần theo đúng giá trị của nó, giả sử ban đầu cho mảng là: {5, 3, 1, 9, 8, 5, 3} thì sau khi sắp xếp giảm dần mảng sẽ là: {9, 8, 5, 5, 3, 3, 1} và đáp án chính là 9855331 là số lớn nhất ghép được.

Nếu áp dụng quy tắc trên với ví dụ mẫu đề cho 3 số là: {12, 907, 91} thì sắp xếp giảm dần xong sẽ có mảng là: {907, 91, 12} trong khi đó đáp án đúng mà đề đưa ra phải là: {91, 907, 12} nghĩa là số 91 sẽ là số lớn nhất lớn hơn 2 số còn lại, rồi 907 sẽ là số lớn nhì lớn hơn 12 còn 12 là số nhỏ nhất. Vậy nếu muốn sắp xếp ra đúng được như vậy ta phải nhìn ra được quy tắc tụi nó so sánh với nhau làm sao để biết được thằng nào lớn hơn thằng nào. Chỉ cần nhìn ra được đúng quy luật tụi nó so sánh với nhau để quyết định thằng nào lớn hơn thằng nào thì đó chính là mấu chốt giúp giải quyết được bài này.

Ở đây mình sẽ phân tích cho các bạn nhìn ra cách mà tụi nó so sánh quyết định thằng nào lớn hơn thằng nào để biết thằng nào sẽ đứng trước, thằng nào sẽ đứng sau để tạo thành được số lớn nhất. Nó đơn giản làm theo đúng quy luật của bài thôi, nếu ta có 2 số a, b cần so sánh với nhau (hiện tại a đang đứng trước b) thì ta sẽ có 2 trường hợp hoặc kết quả ab hoặc kết quả ba ta xem trong 2 trường hợp đó trường hợp nào cho kết quả lớn hơn. Nếu trường hợp ab cho kết quả lớn hơn ba thì hiện tại tụi nó đã đứng đúng vị trí (a đứng trước b) nên ta không cần đổi chỗ, nhưng nếu trường hợp ba cho kết quả lớn hơn ab thì ta phải đổi chỗ tụi nó lại vì hiện tại a đang đứng trước b, ta cần đổi chỗ cho b ra trước a. Đơn giản là vậy thôi.

Thử áp dụng quy tắc so sánh trên để đi sắp xếp mảng ví dụ đề bài cho là: {12, 907, 91} giả sử theo thuật toán sắp xếp cơ bản Interchange Sort dùng 2 vòng lặp lồng nhau mà các bạn hay làm hồi học nhập môn lập trình.

Thì đầu tiên so sánh a[0] là 12 sẽ so sánh với a[1] là 907, ta có 2 số là 12907 và 90712 thì ta thấy 90712 sẽ lớn hơn 12907 nên ta sẽ đổi chỗ cho 907 lên, 12 xuống, lúc này mảng được: {907, 12, 91}.

Tiếp tục so sánh a[0] là 907 với a[2] là 91, ta có 2 số là 90791 và 91907 thì ta thấy 91907 sẽ lớn hơn 90791 nên ta sẽ đổi chỗ tụi nó, lúc này mảng được: {91, 12, 907}.

Tiếp tục so sánh a[1] là 12 với a[2] là 907, ta có 2 số là 12907 và 90712 thì ta thấy 90712 sẽ lớn hơn 12907 nên ta sẽ đổi chỗ tụi nó, lúc này mảng được: {91, 907, 12}.

Đến đây đã hết cặp để so sánh. Ta kết luận số lớn nhất được tạo ra sẽ là: 9190712.

Đó giờ đây các bạn có thấy đơn giản chưa? Nó chỉ là bài toán đi sắp xếp mảng giảm dần thôi có điều quy tắc so sánh 2 số với nhau nó sẽ không phải so sánh theo kiểu giá trị như cách bình thường hay làm mà nó sẽ so sánh theo kiểu cứ đi ghép 2 trường hợp thằng này đứng trước thằng kia đứng sau rồi xem trường hợp nào số lớn nhất? Nếu đang đứng sai thì đổi chỗ lại thôi.

Thì nếu các bạn đang đi xử lý trên mảng số chỗ ghép 2 số lại với nhau ví dụ 123 cần so sánh với 45 lúc này để tạo ra được số 12345 thì ta hiểu nó sẽ là 123 \* 10^2 + 45 cũng như để tạo ra số 45123 thì ta hiểu nó sẽ là 45 \* 10^3 + 123. Nghĩa là số hiện tại nhân cho 10 mũ (số lượng chữ số của số kia) rồi cộng với số kia. Chỗ này phải xử lý có một hàm đếm số lượng chữ số của 1 số rồi mới làm được, rồi cũng cần lưu ý mỗi số tối đa 10^5 thì khi ghép với nhau có thể lên đến tối đa 12 chữ số thì phải chọn kiểu dữ liệu long long lưu trữ kết quả cũng như chỗ nhân rồi cộng lại với nhau nếu giả sử mảng đang là kiểu int thì các vùng nhớ trung gian được tạo ra cũng sẽ căn cứ theo kiểu int lúc đó có thể bị tràn, nên ta phải nhớ mà ép kiểu hoặc ngay từ đầu chọn kiểu long long cho mảng dữ liệu thì sẽ không sợ tràn.

Nói chung nếu mảng dữ liệu đang là kiểu số nguyên thì quy trình xử lý sẽ rườm rà thêm, trong khi đó sẽ rất gọn nếu chúng ta chọn kiểu chuỗi, bởi chuỗi trong C++ có thể ngay lập tức nối chuỗi này với chuỗi kia bằng phép tính + . Ví dụ: string a = “123”, b = “45”; thì string x = a + b lúc này x = “12345” cũng như string y = b + a lúc này y = “45123” rồi từ đây khi so sánh x với y thì lúc này x, y đang có cùng độ dài, nguyên tắc so sánh của chuỗi trong C++ lúc này cũng tương tự như cách so sánh 2 số cùng độ dài đó là so sánh từng ký tự từ trái qua phải và tại chỗ nào ký tự có sự khác biệt nhau thì ngay lập tức kết luận ký nào lớn hơn thì chuỗi đó lớn hơn, ký tự nào nhỏ hơn thì chuỗi đó nhỏ hơn. Nếu 2 ký tự bằng nhau thì lại xét qua ký tự tiếp theo và cứ thế lặp lại. Và có thể nhanh chóng gọi ra theo các phép operator so sánh >, <, >=, <=, ==, != đã được hỗ trợ trong string của C++. Nên nói chung nếu chỗ này dùng string sẽ rất gọn thay vì dùng số.

Source code để các bạn tham khảo thấy rõ những gì mình phân tích ở trên:

[#include](https://www.facebook.com/hashtag/include?__eep__=6&__cft__%5b0%5d=AZXdnblie2h4uMktEbXj_Joo2vkOMUxjUjFAg5MV_z4gECF-7XbN16AqvMMpzl0F5Z6mu3BzK4Ftjyjkvfd6pJyygt3q998ZrDJ-0oi-MGcSCqkudmTx5TEIx1TPftEsTo0p0Jo3CdSQUehshk_KCXry&__tn__=*NK-R) <bits/stdc++.h>

using namespace std;

int main(){

int n;

cin >> n;

string a[n];

for(int i = 0; i < n; ++i){

cin >> a[i];

}

for(int i = 0; i < n; ++i){

for(int j = i + 1; j < n; ++j){

if(a[j] + a[i] > a[i] + a[j]){

swap(a[i], a[j]);

}

}

}

for(int i = 0; i < n; ++i){

cout << a[i];

}

return 0;

}

Đánh giá độ phức tạp của cách làm này:

+ Độ phức tạp không gian (Space Complexity): O(N) với N là số lượng phần tử của mảng a.

+ Độ phức tạp thời gian (Time Complexity): O(N) của bước đọc mảng từ input + O(N^2) của bước sắp xếp giảm dần dùng thuật toán Interchange Sort + O(N) của bước duyệt mảng in ra kết quả. Tổng quát theo quy tắc cộng BigO có thể kết luận theo thằng lớn nhất tức là O(N^2) của bước sắp xếp với N là số lượng phần tử của mảng a.

Vậy ta có thể thấy độ phức tạp thời gian lớn nhất đến từ bước đi sắp xếp mảng giảm dần. Vì giới hạn tối đa của N chỉ là 100 theo như đề bài đưa ra, nó quá nhỏ thế nên kể cả các bạn nào đi dùng những thuật toán sắp xếp không tối ưu như Bubble Sort, Interchange Sort, Selection Sort ... với độ phức tạp O(N^2) thì cũng không sợ bị TLE với bài này nhé vì 100^2 cũng chỉ 10^4 thì nó vẫn quá nhỏ so với ngưỡng (3 đến 5)\*10^7. Nhưng với mình thì mình thích dùng hàm sort trong STL để vừa tối ưu chỉ O(N \* logN) mà vừa gọn lẹ để gọi ra. Cũng như nếu đề bài này mà cho ngưỡng giới hạn N có thể tối đa lên đến 10^4 hoặc cao hơn nữa thì các bạn phải dùng những thuật toán sắp xếp tối ưu với độ phức tạp O(N \* logN) chứ nếu dùng những thuật toán sắp xếp không tối ưu với độ phức tạp O(N^2) như ở trên thì sẽ bị TLE ngay. Dưới đây là code nếu áp dụng hàm sort trong STL để các bạn tham khảo.

[#include](https://www.facebook.com/hashtag/include?__eep__=6&__cft__%5b0%5d=AZXdnblie2h4uMktEbXj_Joo2vkOMUxjUjFAg5MV_z4gECF-7XbN16AqvMMpzl0F5Z6mu3BzK4Ftjyjkvfd6pJyygt3q998ZrDJ-0oi-MGcSCqkudmTx5TEIx1TPftEsTo0p0Jo3CdSQUehshk_KCXry&__tn__=*NK-R) <bits/stdc++.h>

using namespace std;

bool cmp(const string &a, const string &b){

return b + a < a + b;

}

int main(){

int n;

cin >> n;

string a[n];

for(int i = 0; i < n; ++i){

cin >> a[i];

}

sort(a, a + n, cmp);

for(int i = 0; i < n; ++i){

cout << a[i];

}

return 0;

}

Trên trang web luyencode ở bài tập này phía dưới comment có bạn comment cách giải quyết dùng cấu trúc dữ liệu multiset chỉ đổ dữ liệu vào sau đó xuất các phần tử trong đó ra và hệ thống đã báo AC với code đó. Tuy nhiên mình thấy rằng code đó chưa được chuẩn, sẽ bị sai ví dụ điển hình trong tình huống test mảng 4 số sau: {98, 97, 96, 9} lẽ ra đáp án đúng số lớn nhất tạo ra được phải là: 9989796 nhưng code đó của bạn ấy ra đáp án là: 9897969. Hệ thống vẫn báo AC là do may mắn chưa có test nào khiến ra đáp án sai giống như mình đưa ra ở trên thôi. Nên các bạn lưu ý nhé.

