BÁO CÁO THUẬT TOÁN SẮP XẾP (SORT)

Họ và tên: Võ Thế Minh

MSSV: 18120211

Lớp: 18CTT2

GVHD: Phan Thị Phương Uyên

*Lưu ý: Phần cài đặt thuật toán đã có trong code – cô có thể chạy trực tiếp trên project của em

1.1 Ý tưởng – thuật toán – đánh giá thuật toán

1. Selection sort

Tên thuật toán	Ý tưởng	Đánh giá
1.Selection sort	Sắp xếp một mảng bằng cách đi tìm phần tử lớn nhất(hoặc nhỏ nhất) đưa lên đầu và lần lượt cho các phần tử nhỏ nhất còn lại	Thuật toán ít phải đổi chỗ các phần tử nhất trong số các thuật toán sắp xếp(n lần hoán vị) nhưng có độ phức tạp so sánh là $O(n^2)$ ($n^2/2$ phép so sánh). **Thuật toán tốn thời gian gần như bằng nhau đối với mảng đã được sắp xếp cũng như mảng chưa được sắp xếp**.
2.Insertion sort	Thuật toán sắp xếp bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự người chơi bài rút lần lượt từ quân thứ 2, so với các quân đứng trước nó để chèn vào vị trí thích hợp	Thuật toán sử dụng trung bình $n^2/4$ phép so sánh và $n^2/4$ lần hoán vị, $n^2/2$ phép so sánh và $n^2/2$ lần hoán vị trong trường hợp xấu nhất, n-1 phép so sánh và 0 lần hoán vị trong trường hợp tốt nhất.

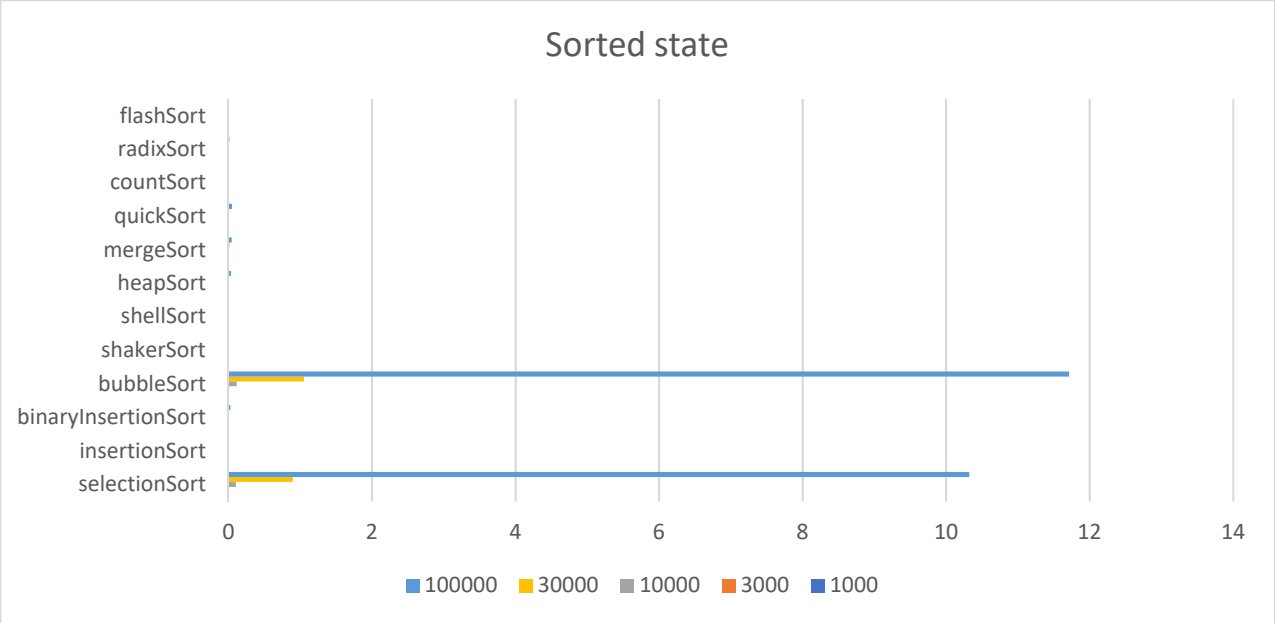
		Thuật toán thích hợp đối với mảng đã được sắp xếp một phần hoặc mảng có kích thước nhỏ.
3.Binary Insertion sort	**Là insertionSort** được cải tiến ở phần tìm phần tử thích hợp để chèn vào nhờ việc sử dụng binarySearch	Toàn bộ thuật toán vẫn có thời gian chạy tệ nhất trong trường hợp chạy O (n^2) do một loạt các phép đổi cho mỗi lần chèn
4.Bubble sort	Thuật toán sắp xếp bubble sort thực hiện **sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự**(số sau bé hơn số trước với trường hợp sắp xếp tăng dần) cho đến khi dãy số được sắp xếp	Độ phức tạp: O(n^2) **Dễ cài đặt thích hợp cho các vùng dữ liệu nhỏ**
5.Shaker sort	**Thuật toán sắp xếp shakerSort là một cải tiến của Bubble Sort. Sau khi đưa phần tử nhỏ nhất về đầu dãy, thuật toán sẽ giúp chúng ta đưa phần tử lớn nhất về cuối dãy**. Do đưa các phần tử về đúng vị trí ở cả hai đầu nên thuật toán sắp xếp cocktail sẽ giúp cải thiện thời gian sắp xếp dãy số	Độ phức tạp cho trường hợp tốt nhất là O(n). Độ phức tạp cho trường hợp xấu nhất O(n^2). Độ phức tạp trong trường hợp trung bình là O(2n). **Thuật toán nhận diện được mảng đã sắp xếp**
6.Shell sort	Shell Sort là một giải thuật sắp xếp mang lại hiệu quả cao dựa trên giải thuật **sắp xếp chèn (Insertion Sort)**. Giải thuật này tránh các trường hợp phải tráo đổi vị trí của hai phần tử xa nhau trong giải thuật sắp xếp chọn (nếu như phần tử nhỏ hơn ở vị trí bên phải khá xa so với phần tử lớn hơn bên trái)	Việc đánh giá giải thuật Shell sort hiện nay rất phức tạp, thậm chí 1 số chưa được chứng minh.Nhưng có 1 điều chắc chắn là **hiệu quả của thuật toán phụ thuộc vào dãy các độ dài được chọn**
7.Heap sort	Giải thuật Heapsort còn được gọi là giải thuật vun đống, nó có thể được xem như bản cải tiến của Selection sort khi chia các phần tử thành 2 mảng con, 1 mảng các phần tử đã được sắp xếp và mảng còn lại các phần tử chưa được sắp xếp. Trong mảng chưa được sắp xếp, các phần tử lớn nhất sẽ được tách ra và đưa	**Giống như insertionSort** Độ phức tạp trong trường hợp xấu nhất là O (n log n)

	vào mảng đã được sắp xếp. Điều cải tiến ở Heapsort so với Selection sort ở việc sử dụng **cấu trúc dữ liệu heap** thay vì tìm kiếm tuyến tính (linear-time search) như Selection sort để tìm ra phần tử lớn nhất	
8.Merge sort	Ý tưởng chúng ta sẽ **chia mảng lớn thành những mảng con nhỏ hơn rồi so sánh** bằng cách chia đôi mảng lớn và chúng ta tiếp tục chia đôi các mảng con cho tới khi mảng con nhỏ nhất chỉ còn 1 phần tử. Sau đó chúng ta sẽ tiếng hành so sánh 2 mảng con có cùng mảng cơ sở. Khi so sánh chúng sẽ vừa sắp xếp vừa ghép 2 mảng con đó lại thành mảng cơ sở, chúng ta tiếp tục so sánh và ghép các mảng con lại đến khi còn lại mảng duy nhất thì đó là mảng đã được sắp xếp	Ưu điểm: Sắp sếp nhanh hơn so với các thuật toán cơ bản (Insertion Sort, Selection Sort, Interchage Sort), nhanh hơn Quick Sort trong một số trường hợp. Nhược điểm: thuật toán khó cài đặt, **không nhận dạng được mảng đã được sắp.** Hiệu quả được tính bằng công thức O(n log n)
9.Quick sort	Quick sort là một trong những thuật toán chia để trị. Quick sort **chia một mảng lớn của chúng ta thành hai mảng con nhỏ hơn: mảng có phần tử nhỏ và mảng có phần tử lớn**. Sau đó Quick sort có thể sort các mảng con này bằng phương pháp đệ quy	Trường hợp tốt: O(nlog(n)) Trung bình: O(nlog(n)) Trường hợp xấu: O(n^2)
10.Counting sort (Thuật toán này em chưa tìm hiểu kĩ mong cô thông cảm)	Ý tưởng của thuật toán là đếm trong khoảng [0..M] có bao nhiêu giá trị 0 (giả sử có C(0) giá trị), bao nhiêu giá trị 1 (giả sử có C(1) giá trị),… có bao nhiêu giá trị M (giả sử có C(M) giá trị). Sau đó tôi sắp xếp lại mảng bằng cách đặt C(0) phần tử 0 ở đầu, tiếp theo đặt C(1) phần tử 1 tiếp theo,… và đặt C(M) phần tử M cuối cùng	Độ phức tạp của thuật toán này là O(max(N,M)). **Giá trị của M càng nhỏ thì thuật toán sắp sếp càng nhanh**
11.Radix sort	Khác với tất cả các thuật toán nêu trên, **RadixSort không sử dụng việc so sánh 2 phần tử.** Đầu tiên, **thuật toán sẽ chia các phần tử thành các nhóm, dựa trên**	**Có thể chạy nhanh hơn các thuật toán sắp xếp sử dụng so sánh**. Ví dụ nếu ta sắp xếp các số nguyên 32 bit, và chia nhóm theo 1 bit, thì độ phức tạp là O(N)O(N). Trong trường hợp

	chữ số cuối cùng (hoặc dựa theo bit cuối cùng, hoặc vài bit cuối cùng) Sau đó ta đưa các nhóm lại với nhau, và được danh sách sắp xếp theo chữ số cuối của các phần tử. Quá trình này lặp đi lặp lại với chữ số át cuối cho tới khi tất cả vị trí chữ số đã sắp xếp.	tổng quát, độ phức tạp là O(N∗log(max(ai)))
Flash sort	**Xuất phát từ tư tưởng: với mỗi một phần tử a_i ta có thể tính toán được vị trị gần đúng (thuộc vào một lớp) của nó trong mảng đã sắp xếp một cách trực tiếp từ giá trị $a_i$mà không cần phải so sánh nó với các phần tử khác.Nếu giá trị phần tử lớn nhất là*Max*, nhỏ nhất là *Min* và phân các khóa thành m lớp** Ta dùng mảng phụ L đánh dấu vị trí *m*phân lớp của mảng khóa a. Phân lớp thứ iđược coi là rỗng khi L[i] là vị trí chính xác phần tử cuối của phân lớp trong mảng khóa (hình 1), phân lớp *i* được coi là đã đầy khi mà L[i] là vị trí chính xác phần tử đầu tiên của phân lớp trong mảng khóa Khi bỏ một phần tử vào phân lớp i, ta giảm L[i] đi 1 cho đến khi về đến đúng vị trí của nó thì nghĩa là đầy (hình 2). Vậy để xác định được vị trí xuất phát cũng như vị trí kết thúc của từng phân lớp, ta phải biết được kích thước của từng phân lớp. Ban đầu, các phân lớp được coi là rỗng, L[i] được khởi gán bằng vị trí của phần tử cuối cùng của lớp i. Dễ thấy rằng, vị trí ban đầu của phân lớp thứ	Khó cài đặt **Độ phức tạp tối thiểu**

	nhất sẽ là kích thước của nó, còn của phân lớp m sẽ là n và *L[i]=L[i-1]+ kích thước của phân lớp i*	

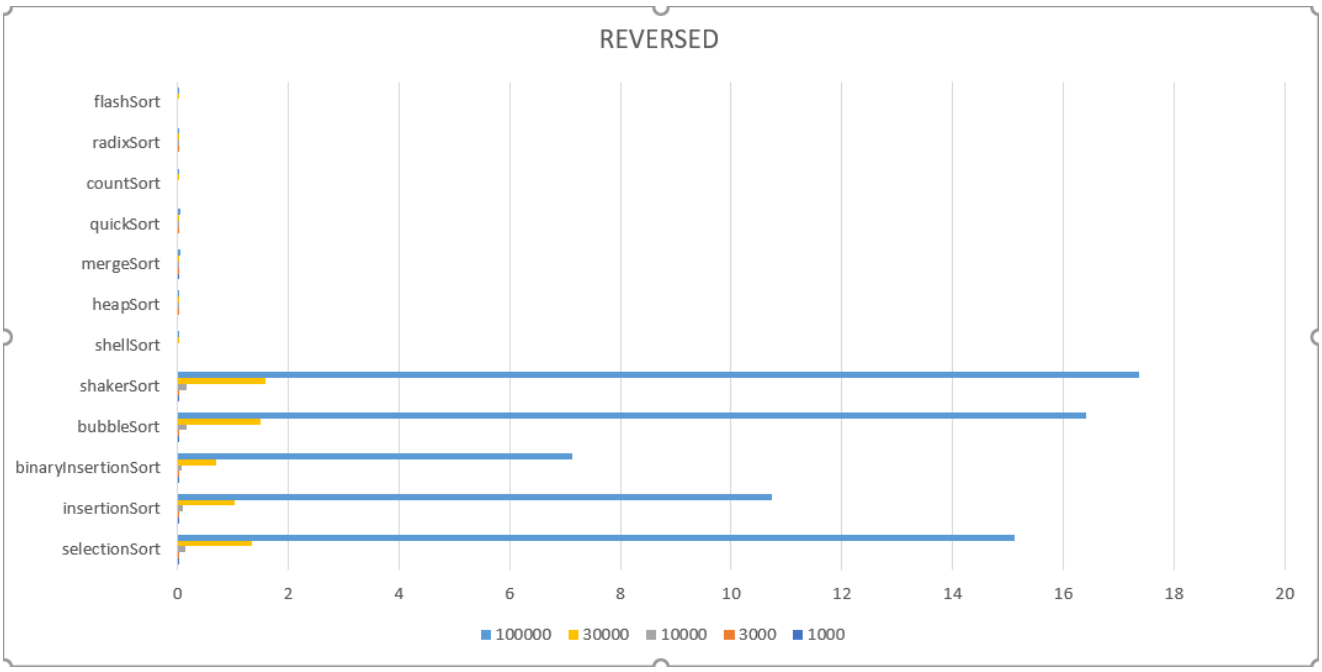
1.2 Trình bày về kết quả thí nghiệm và nhận xét



1.2.1 Tình trạng dữ liệu đã được sắp xếp

Nhận xét: - Sự phân hóa cực kì rõ ràng thuật toán chạy chậm nhất là bubbleSort hiển nhiên vì bubble sort có độ phức tạp là O(n^2) theo sau là selectionSort

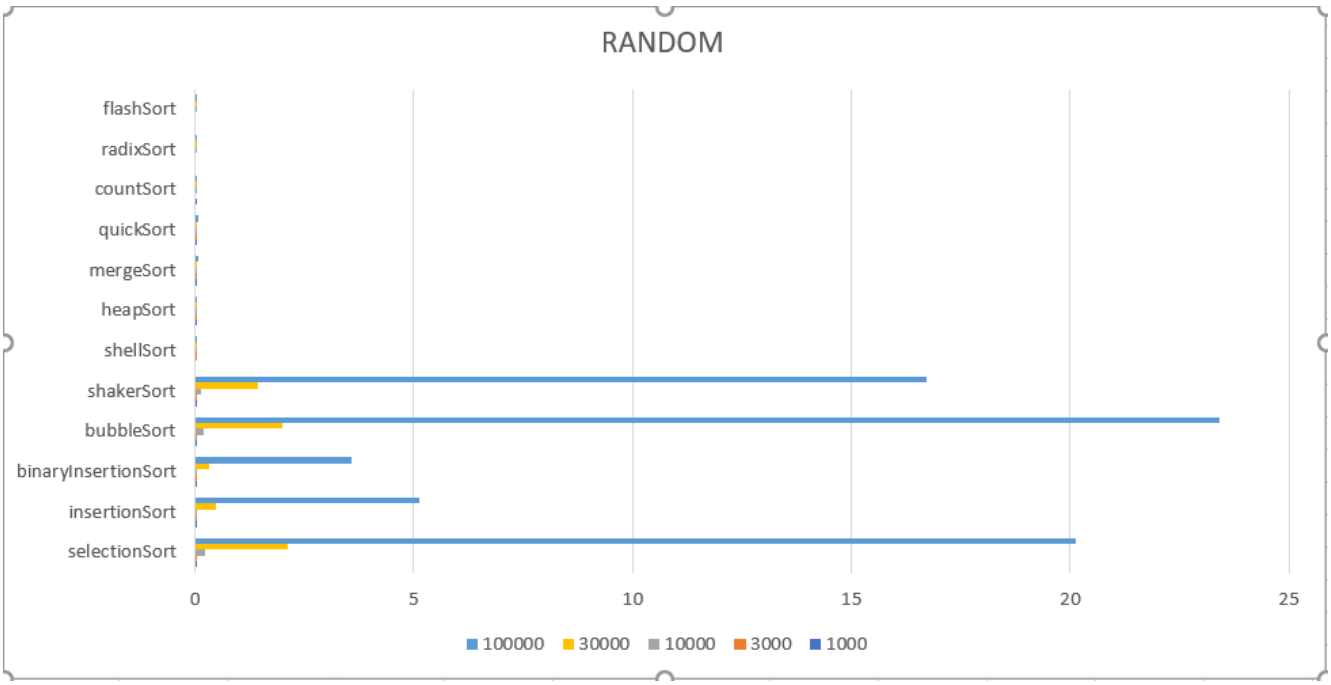
- Các thuật toán còn lại cần thời gian gần như là tương đồng nhau với từng vùng dữ liệu
- insertionSort, binaryInsertionSort và shakerSort thực hiện rất tốt trong tình trạng mảng đã được sắp xếp vì trong tình trạng này độ phức tạp của giải thuật là O(n)
- Trong tình trạng dữ liệu này các thuật toán thực hiện với thời gian tối thiểu vì nó giảm được số lần thực hiện phép so sánh – biến đổi



1.2.2 Tình trạng dữ liệu sắp xếp ngược

Nhận xét: - bubbleSort và selectionSort vẫn tốn nhiều thời gian để sắp xếp nhưng lần này ta có thêm shakerSort, insertionSort và binaryInsertionSort cũng tốn nhiều thời gian, vì cả 3 thuật toán trên chỉ nhận biết được mảng đã được sắp xếp hoặc sắp xếp một phần

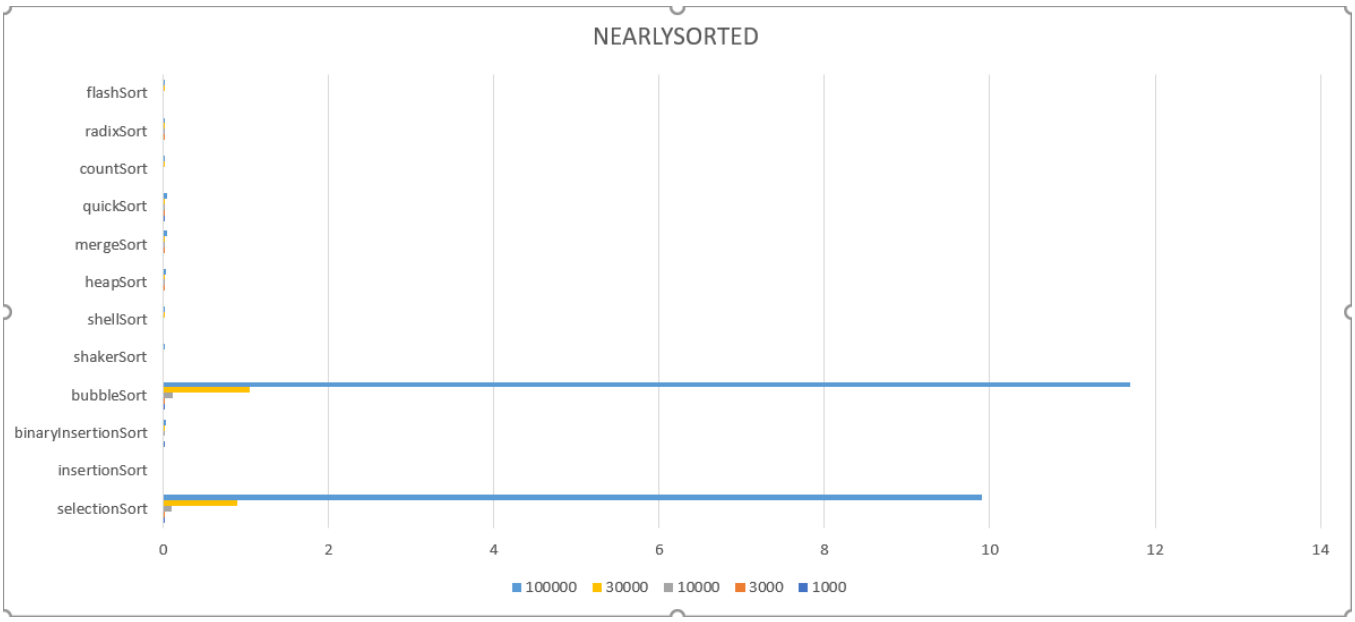
- Trong tình trạng dữ liệu được sắp xếp ngược thì đa số các thuật toán có thời gian thực hiện lâu nhất vì chúng phải thực hiện tối đa các phép so sánh
- Những thuật toán còn lại đều rất nhanh và không có nhiều sự chênh lệch



1.2.3 Tình trạng dữ liệu ngẫu nhiên

Nhận xét: - bubbleSort và selectionSort vẫn tốn nhiều thời gian để sắp xếp

- Không có nhiều sự chênh lệch tình trạng ngẫu nhiêu và sắp xếp ngược
- Những thuật toán khác ngoại trừ flashSort có độ phức tạp là O(n) thì đều có độ phức tạp trung bình là O(n*log(n)) nên thực hiện chương trình rất nhanh



1.2.4 Tình trạng dữ liệu gần như đã được sắp xếp

Nhận xét – nhận xét chung cho 4 đồ thị:

- bubbleSort và selectionSort tốn nhiều thời gian để sắp xếp
- Trong tình trạng mảng đã được sắp xếp hoặc sắp xếp trước một phần thì insertionSort và binaryInsertion thực hiện rất tốt vì giải thuật này có thể nhận biết được mảng đã được sắp xếp hoặc sắp xếp một phần
- Trong tình trạng dữ liệu đã được sắp xếp các thuật toán thực hiện với thời gian tối thiểu vì nó giảm được số lần thực hiện phép so sánh – biến đổi, ngược lại với tình trạng dữ liệu được sắp xếp ngược
- flashSort nhanh trong mọi trường hợp
- shellSort, heapsort, mergeSort, quicksort, countSort, radixSort, có tốc độ không thay đổi nhiều trong cả 4 tình trạng dữ liệu

1.3 Nguồn tham khảo:

https://www.wikipedia.org/

https://www.stdio.vn/

https://www.geeksforgeeks.org/

https://stackoverflow.com/