TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ TRI THỨC

MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỒ ÁN ĐO ĐỘ PHỨC TẠP CỦA CÁC THUẬT TOÁN

Tháng 7, 2021

MỤC LỤC

1. THÔNG TIN NHÓM SỐ 11

STT	MSSV	Họ và tên
1	20127335	Phạm Huy Cường Thịnh
2	20127544	Nguyễn Tuấn Kiệt
3	20127559	Nguyễn Hoàng Luân
4	20127466	Cao Nhật Đức

Giảng viên hướng dẫn: Thầy Bùi Huy Thông và Cô Nguyễn Ngọc Thảo

2. PHẦN GIỚI THIỆU

Nhóm số 11 chúng em từ lớp CLC07 xin trình bày với thầy đồ án đo độ phức tạp của thuật toán. Đồ án mục đích để tính toán, chạy thuật toán cũng như đo thời gian của một thuật toán, cụ thể ở đây là những thuật toán sắp xếp và đưa ra những kết luận về các thuật toán.

Chúng em đã học được cách làm sao để đo được thuật toán, hiểu được những nguyên lí cơ bản của các loại sắp xếp, tính toán được đầu vào và đầu ra, hiểu được trường hợp tốt nhất và trường hợp xấu nhất khi đưa một đầu vào.

Bọn em xin chân thành cảm ơn thầy cô hướng dẫn chúng em hoàn thành được trọn vẹn đồ án.

Về phần cấu hình máy để đo:

CPU: Intel Core i5 – 1135G7 @ 2.40 GHz.

RAM: 8Gb DDR4.

SSD: 512 Gb.

GPU: Intel Iris Xe Graphics.

3. TRÌNH BÀY THUẬT TOÁN

a) Thuật toán Bubble sort

Ý tưởng: 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.

Các bước:

B1: Gán i=0; j=n-1;

B2: So sánh a[j] và a[j-1]. Đổi chỗ a[j] và a[j-1] nếu a[j] < a[j-1].

B3: Tiếp tục giảm j và thực hiện bước 2 đến khi j<i thì dừng. Gán j=n-1 và tăng i thêm 1 và lặp lại cho đến khi mảng đã được sắp xếp.

Độ phức tạp:

-Trường hợp tốt: O(n)

-Trường hợp trung bình: O(n^2)

-Trường hợp xấu: O(n^2)

-Không gian bộ nhớ sử dụng: O(1)

Thuật toán cải tiến:

Shaker sort: Sau khi đưa phần tử nhỏ nhất về đầu mảng sẽ đưa phần tử lớn nhất về cuối dãy. Do đưa các phần tử về đúng vị trí ở cả hai đầu nên Shaker Sort sẽ giúp cải thiện thời gian sắp xếp dãy số do giảm được độ lớn của mảng đang xét ở lần so sánh kế tiếp.

Shaker sort nhận diện được mảng đã sắp xếp nhưng Bubble sort thì không. Trong trường hợp mảng có ngẫu nhiên phần tử với thứ tự đảo lộn thì Bubble Sort và Shaker sort cho thời gian sắp xếp gần tương đương nhau. Shaker sort chỉ có ưu thế hơn Bubble sort khi mảng gần được sắp xếp hoàn chỉnh.

b) Thuật toán Heap sort

Ý tưởng:

Đầu tiên chúng ta hãy xác định một Cây nhị phân hoàn chỉnh. Cây nhị phân hoàn chỉnh là cây nhị phân trong đó mọi cấp, ngoại trừ cấp cuối cùng, được lấp đầy hoàn toàn và tất cả các nút ở bên trái càng xa càng tốt.

Một Binary Heap là một cây nhị phân hoàn chỉnh trong đó các mục được lưu trữ theo một thứ tự đặc biệt sao cho giá trị trong nút cha lớn hơn (hoặc nhỏ hơn) so với giá trị trong hai nút con của nó. Cái trước được gọi là max heap và cái sau được gọi là min-heap. Heap có thể được biểu diễn bằng một cây hoặc mảng nhị phân.

Các bước:

B1: Xây dựng một heap tối đa từ dữ liệu đầu vào.

B2: Tại thời điểm này, mục lớn nhất được lưu trữ ở gốc của heap. Thay thế nó bằng mục cuối cùng của heap, sau đó giảm kích thước của heap đi 1. Cuối cùng, ta có một gốc heap.

B3: Lặp lại bước 2 trong khi kích thước của heap lớn hơn 1.

Độ phức tạp:

-Trường hợp xấu nhất: O(nlogn)

-Trường hợp tốt nhất: O(n)

-Trường hợp trung bình: O(nlogn)

-Trường hợp xấu nhất (Bộ nhớ):O(n) tổng thể và O(1) auxiliary

c) Thuật toán Selection sort

Ý tưởng:

Mô phỏng cách sắp xếp tự nhiên nhất trong thực tế

Chọn phần tử nhỏ nhất trong mảng lấy ra và đặt vào vị trí đầu tiên của mảng.

Xem dãy hiện tại chỉ có n-1 phần tử(mảng bắt đầu từ vị trí phần tử thứ hai).

Lặp lại bước 1 cho đến khi mảng chỉ còn 1 phần tử.

Các bước:

B1: i = 0.

B2: Tìm phần tử nhỏ nhất a[min] trong dãy hiện hành từ a[i+1] đến a[n-1].

B3: Hoán đổi a[min] với a[i].

B4: Nếu i < n - 1 thì thì i = i + 1 và lặp lại bước 2.

B5: kết thúc.

Độ phức tạp:

Trường hợp tốt nhất: O(n^2) phép so sánh, O(0) phép hoán đổi.

Trường hợp trung bình: O(n^2) phép so sánh, O(n) phép hoán đổi.

Trường hợp xấu nhất: O(n^2)phép so sánh, O(n) phép hoán đổi.

độ phức tạp không gian: O(1).

Thuật toán cải tiến:

Dummy selection sort: thay vì thêm một biến pos để lưu vị trí phần tử nhở nhất hiện tại thì ở dị bản này thì a[i] sẽ luôn hoán đổi với một phần tử a[j] đến khi nó là nhỏ nhất.

biến thể này được sử dụng khi chi phí sắp xếp không là tiêu chí đánh giá.

d) Thuật toán Insertion sort

Ý tưởng:

Xét dãy a[0],a[1],...,a[n-1].

xem a[0] là một dãy có thứ tự, chèn a[1] vào dãy có thứ tự a[0] sao cho dãy mới có hai phần tử a[0] và a[1] có thứ tự.

tương tự chèn a[2],a[3]...a[n-1].

Các bước:

B1: i = 1;//giả sử có đoạn a[1] đã được sắp

B2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào

B3: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]

B4: a[pos] = x; //có đoạn a[1]..a[i] đã được sắp

B5: i = i+1;

Độ phức tạp:

Trường hợp tốt nhất: O(n) phép so sánh, O(0) phép hoán đổi.

Trường hợp trung bình: O(n^2) phép so sánh, O(n^2) phép hoán đổi.

Trường hợp xấu nhất: O(n^2)phép so sánh, O(n^2) phép hoán đổi.

độ phức tạp không gian: O(1).

Thuật toán cải tiến:

Thuật toán này có một cải tiến gọi là chèn nhị phân, dùng thuật toán tìm kiếm nhị phân để tìm ra vị trí chèn thích hợp.

Cải tiến này có độ phức tạp O(nlog(n)) phép so sánh.

e) Thuật toán Merge Sort

Ý tưởng:

Thuật toán này chia mảng được sắp xếp thành 2 nửa.

Tiếp tục lặp lại điều này trên các nửa mảng đã chia. Cuối cùng hợp nhất các nửa đó thành một mảng đã được sắp xếp.

Hàm merge() được sử dụng để hợp nhất hai nửa của một mảng. Hàm hợp nhất (arr, l, m, r) là quan trọng nhất.

Quy trình sẽ hợp nhất hai nửa của mảng thành một mảng sắp xếp,

nửa mảng là arr [l… m] và arr [m + 1… r] sau khi hợp nhất, sẽ hình thành một mảng được sắp xếp duy nhất.

Các bước:

mergeSort (arr [], l, r)

Nếu r > l

B1: Tìm chỉ số ở giữa mảng để chia mảng thành 2 nửa: giữa

m = (l + r) / 2.

B2: Gọi đệ quy mergeSort trong nửa đầu: mergeSort (arr, l, m).

B3: Gọi đệ quy mergeSort cho nửa sau: mergeSort (arr, m + 1, r).

B4: Kết hợp 2 nửa của mảng đã sắp xếp trong (2) và (3): hợp nhất (arr, l, m, r).

Độ phức tạp:

Trường hợp xấu nhất: O(n log n)

Trường hợp tốt nhất: O(n log n) (typical)

Hiệu suất trung bình: O(n log n)

Tổng hiệu suất không gian trong TH xấu nhất: O(n) tổng thể,

O(1) auxiliary với danh sách liên kết

f) Thuật toán Quick sort

Ý tưởng:

Nó hoạt động bằng cách chọn một phần tử 'pivot' từ mảng và

phân chia các phần tử khác thành hai mảng con, theo cho dù chúng nhỏ hơn hay lớn hơn pivot. Các mảng con là sau đó được sắp xếp một cách đệ quy. Điều này có thể được thực hiện tại chỗ, yêu cầu nhỏ lượng bộ nhớ bổ sung để thực hiện việc sắp xếp.

Các bước:

Mục tiêu của công việc này là: Cho một mảng và một phần tử x làm trục quay.

B1: Đặt x thay cho mảng đã sắp xếp.

B2: Di chuyển tất cả các phần tử của mảng nhỏ hơn x sang bên trái vị trí của x và di chuyển tất cả các phần tử của mảng lớn hơn x ở bên phải của x.

B3: Khi đó chúng ta sẽ có 2 mảng con: mảng bên trái x và mảng bên phải của x.

B4: Tiếp tục làm việc với từng mảng con (chọn trục, phân đoạn) cho đến khi mảng được sắp xếp.

Độ phức tạp:

Trường hợp xấu nhất: O(n^2).

Trường hợp tốt nhất: O(n log n) (simple partition)

or O(n) (three-way partition and equal keys).

Hiệu suất trung bình: O(n log n).

Tổng hiệu suất không gian trong TH xấu nhất: O(n) auxiliary(naive), O(log n) auxiliary(Hoare 1962).

g) Thuật toán Radix sort

Ý tưởng

Radix sort sắp xếp theo thứ tự các phần tử trong 1 mảng lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm,…

Radix sort dùng Counting sort như một chương trình con để sắp xếp.

Các bước:

B1: Tìm phần tử lớn nhất trong mảng, chọn X là số các cơ số của phần tử lớn nhất, từ đó sẽ đi qua các bước duyệt.

B2: Duyệt các cơ số từ hàng đơn vị, hàng chục, hàng trăm bằng cách dùng Counting sort.

Độ phức tạp:

Trường hợp xấu nhất: O(2mn) = O(n)

Phức tạp thời gian: O(wn)

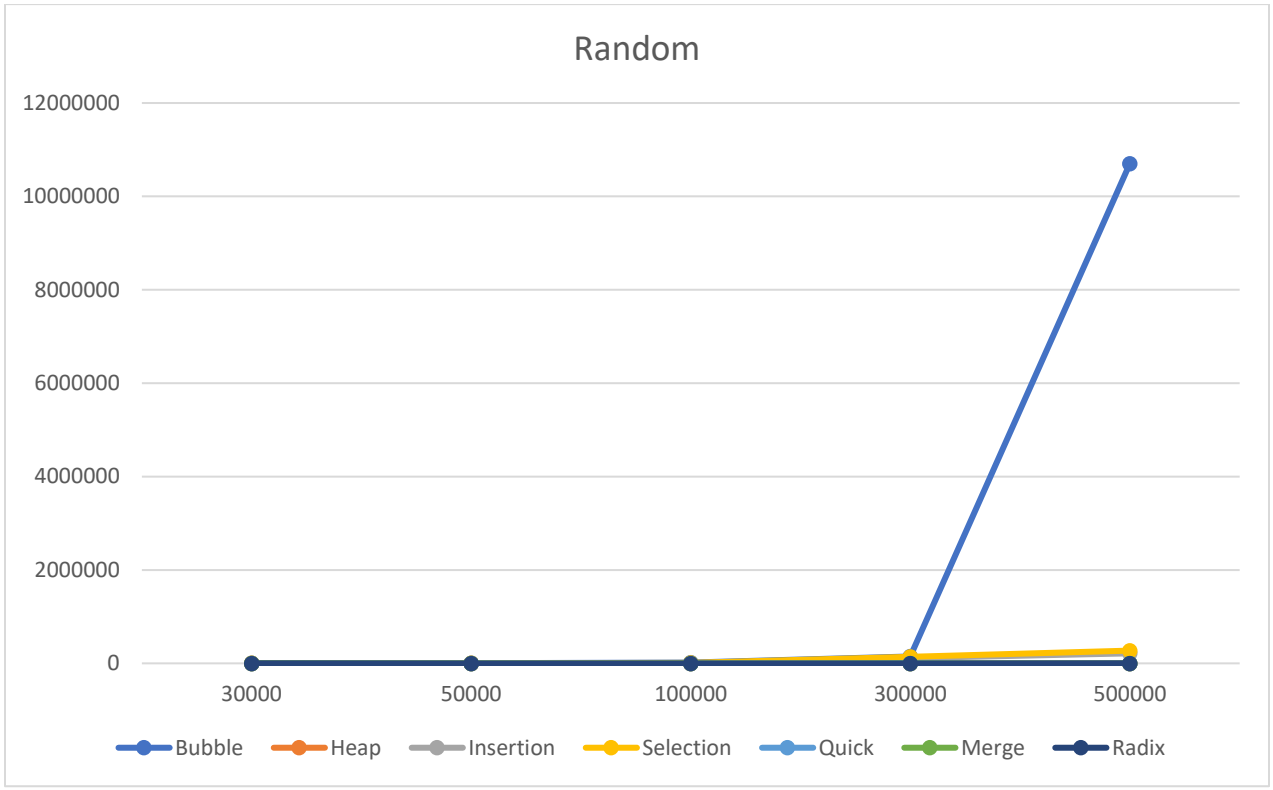
Phức tạp không gian: O(w + n)

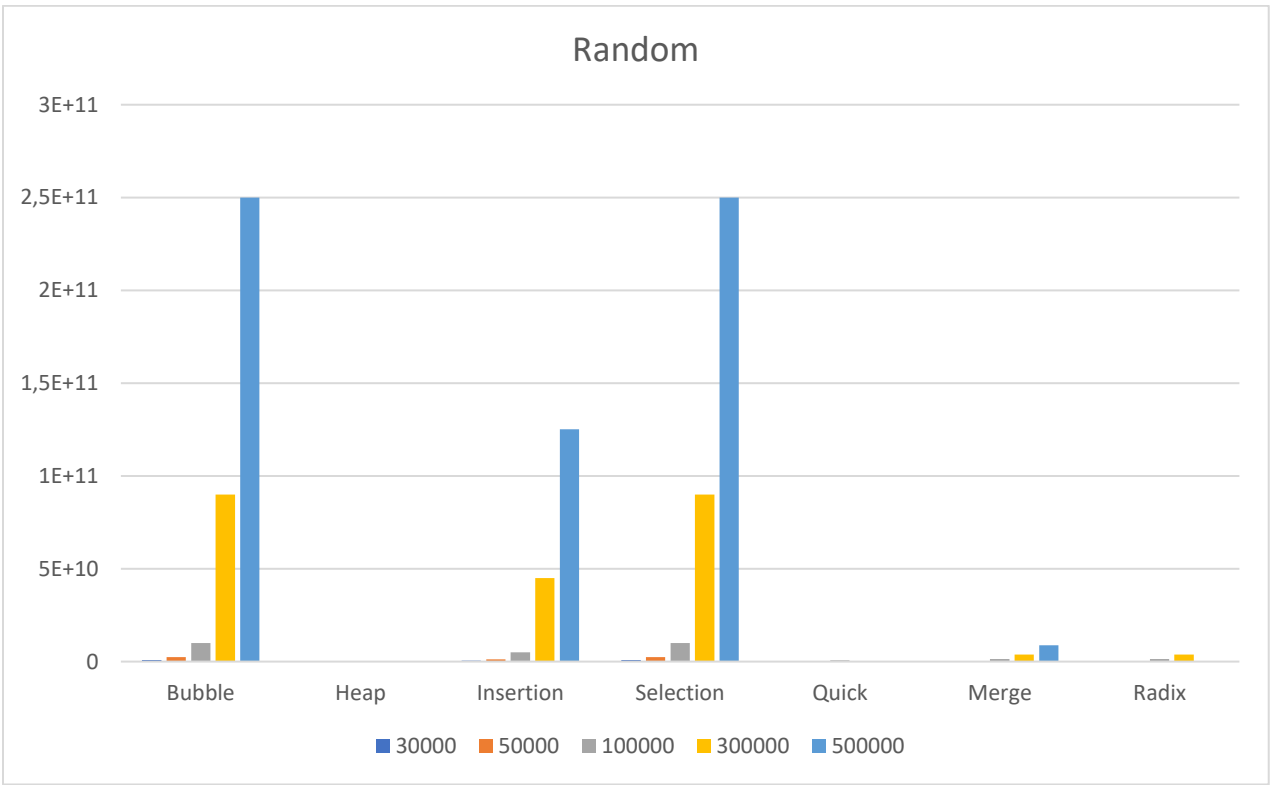
4. KẾT QUẢ VÀ KẾT LUẬN

a) Dữ liệu thứ tự ngẫu nhiên

Data Order: Random										
Data Size	30000		50000		100000		300000		500000	
Result	Time	Com	Time	Com	Time	Com	Time	Com	Time	Com
Bubble	1.252	900029999	3505	2500049999	14552	10000099999	146316	90000299999	1070130	2,5E+11
Heap	8	2120405	14	3721626	27	7944785	89	26188838	142	46608332
Insertion	726	446111017	1484	1254649467	6031	5001877104	115074	44990082861	214502	1,25113E+11
Selection	914	900029999	2405	2500049999	11339	10000099999	140251	90000299999	273333	2,5E+11
Quick	4	940079	6	1653218	12	685706359	35	10880525	58	18909269
Merge	9	100208526	12	228125233	24	1365065559	78	3873426643	122	8812438374
Radix	2	100718596	4	228975303	7	1366765629	24	3878526713	36	27409339

Nhận xét: Selection sort và Bubble sort chạy nhiều phép so sánh nhất, Quick sort chạy ít phép so sánh nhất. Bubble sort chạy với thời gian lâu nhất, Radix sort chạy với thời gian nhanh nhất

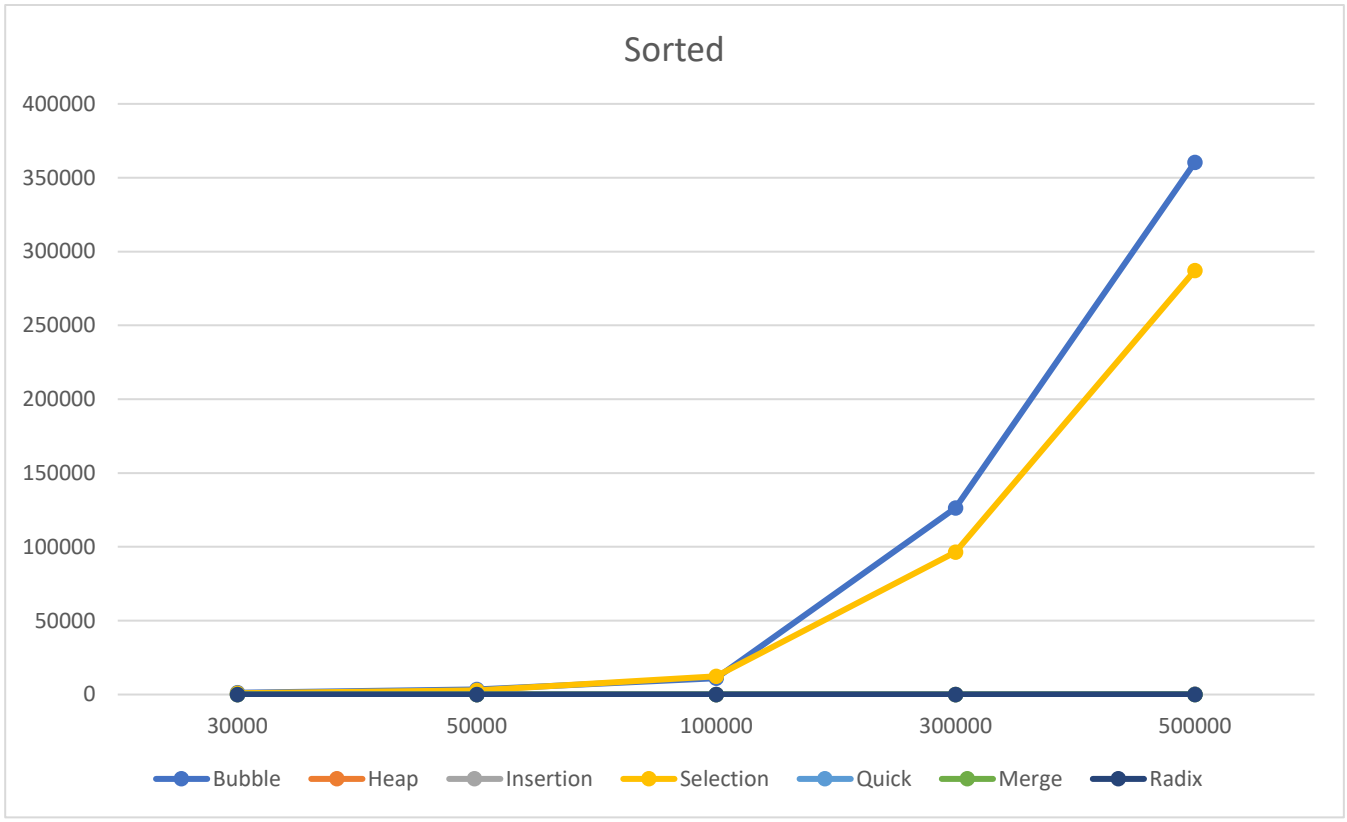

Random
12000000
10000000
8000000
6000000
4000000
2000000
0
30000
50000
100000
300000
500000
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix

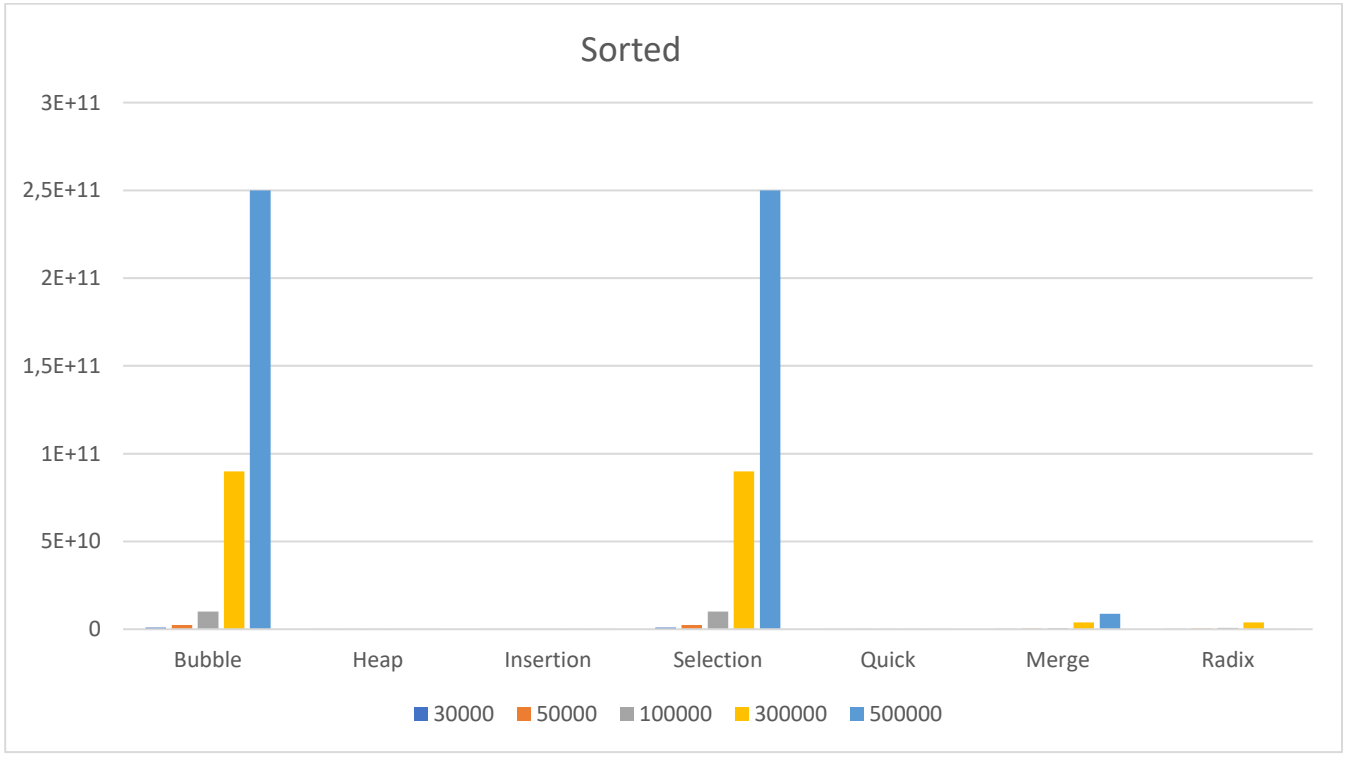

Random
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix
30000
50000
100000
300000
500000

b) Dữ liệu thứ tự đã sắp xếp

Data Order: Sorted										
Data Size	30000		50000		100000		300000		500000	
Result	Time	Com	Time	Com	Time	Com	Time	Com	Time	Com
Bubble	1.222	900029999	3.386	2500049999	17033	10000099999	126267	90000299999	360321	2,5E+11
Heap	8	2206648	10	3875351	29	8265080	72	27113230	116	46904886
Insertion	0	89998	0	149998	1	299998	1	89998	3	1499998
Selection	891	900029999	2702	2500049999	12341	10000099999	96525	90000299999	287171	2,5E+11
Quick	5	101236908	10	229921920	24	1993226	58	6227156	93	37982215
Merge	1	200302266	2	456036380	4	680586612	12	3866229218	18	8808051356
Radix	2	200812336	3	456886450	10	682286682	28	3872229302	45	47982299

Nhận xét: Selection sort và Bubble sort chạy nhiều phép so sánh nhất, Insertion sort chạy ít phép so sánh nhất. Bubble sort chạy với thời gian lâu nhất, Insertion sort chạy với thời gian nhanh nhất

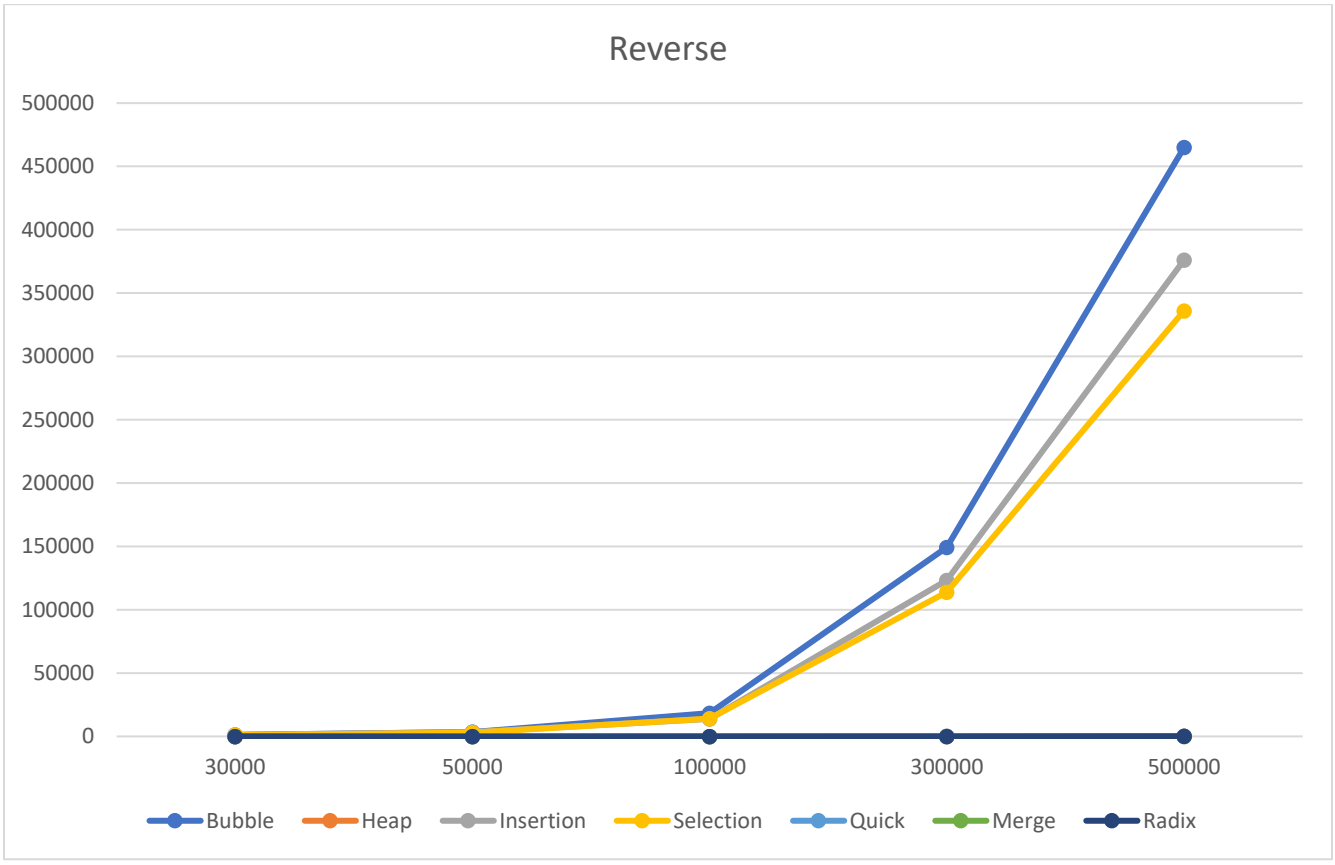

Sorted
400000
350000
300000
250000
200000
150000
100000
50000
0
30000
50000
100000
300000
500000
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix

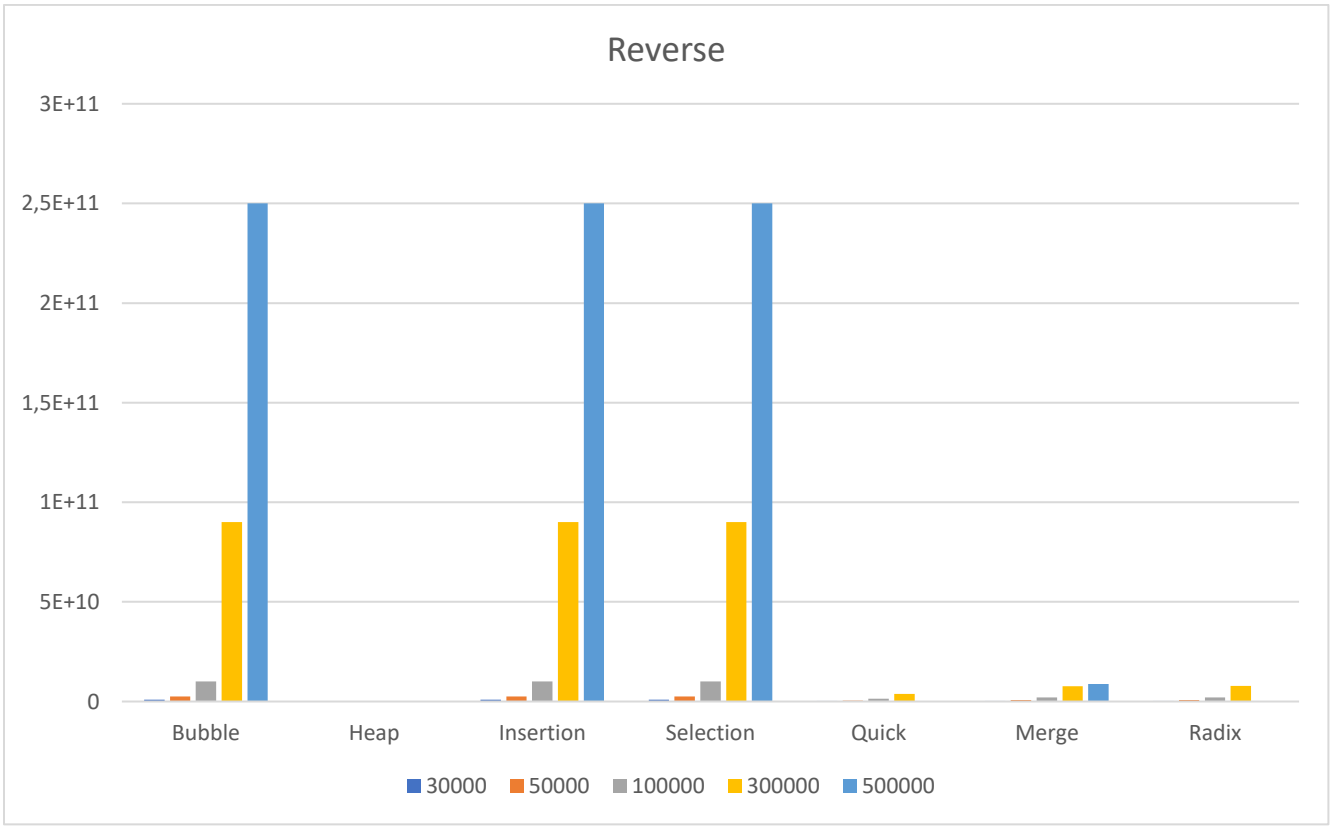

Sorted
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix
30000
50000
100000
300000
500000

c) Dữ liệu thứ tự sắp xếp ngược

Data Order: Reverse										
Data Size	30000		50000		100000		300000		500000	
Result	Time	Com	Time	Com	Time	Com	Time	Com	Time	Com
Bubble	1239	900029999	3436	2500049999	18256	10000099999	149269	90000299999	464915	2,5E+11
Heap	6	2206753	12	3874809	24	8264684	141	27113321	125	46905180
Insertion	1174	900029999	3500	2500049999	13861	10000099999	123231	90000299999	376046	2,5E+11
Selection	1056	900029999	2954	2500049999	14040	10000099999	113806	90000299999	336009	2,5E+11
Quick	5	201360658	8	457883078	22	1368858867	57	3878756480	99	59055189
Merge	1	300464239	2	684066977	4	2047591132	11	7739177357	25	8808642075
Radix	2	300974309	4	684917047	10	2049291202	26	7745177441	49	69055273

Nhận xét: Insertion sort, Selection sort và Bubble sort chạy nhiều phép so sánh nhất, Quick sort chạy ít phép so sánh nhất. Bubble sort chạy với thời gian lâu nhất, Merge sort chạy với thời gian nhanh nhất.

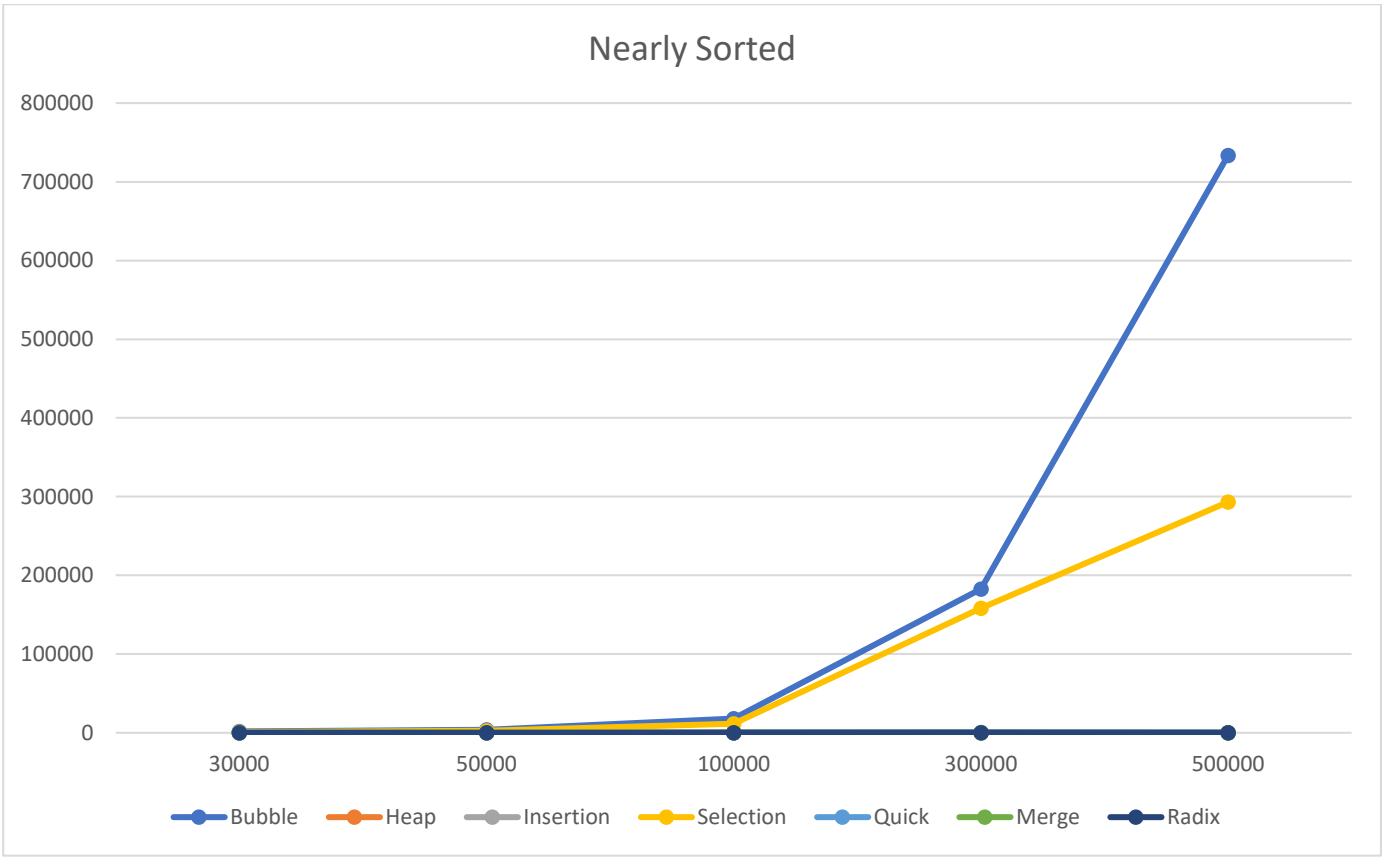

Reverse
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
30000
50000
100000
300000
500000
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix

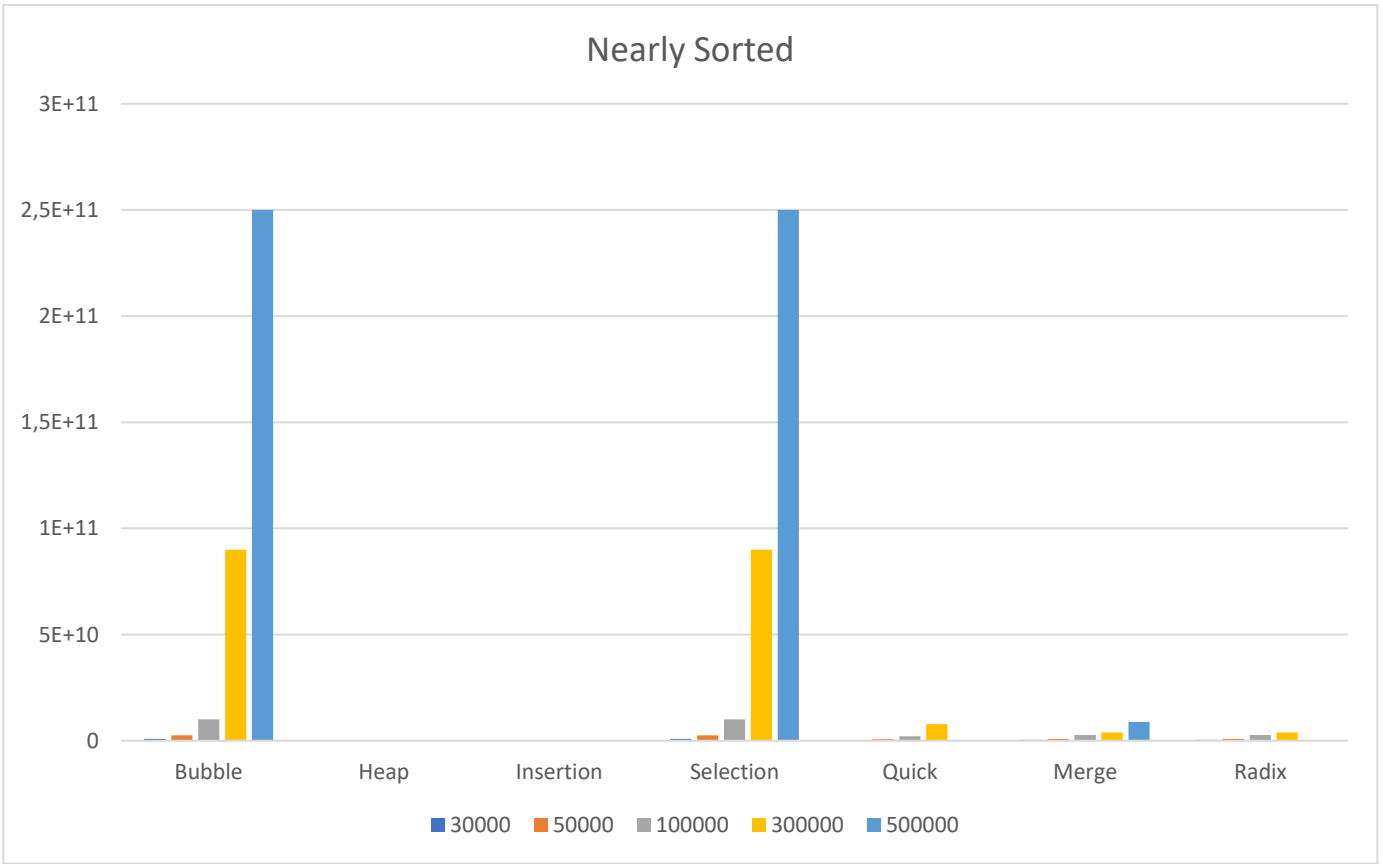

Reverse
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix
30000
50000
100000
300000
500000

d) Dữ liệu thứ tự gần được sắp xếp

Data Order: Nearly Sorted										
Data Size	30000		50000		100000		300000		500000	
Result	Time	Com	Time	Com	Time	Com	Time	Com	Time	Com
Bubble	1785	900029999	3672	2500049999	18049	10000099999	182887	90000299999	733456	2,5E+11
Heap	7	2206648	12	3875351	23	8265080	78	27113407	148	46904985
Insertion	1	408854	1	548174	9	801602	5	1268738	4	2034402
Selection	978	900029999	2518	2500049999	11511	10000099999	158131	90000299999	293165	2,5E+11
Quick	5	301492669	9	685863684	19	2051284472	56	7751404653	96	79628181
Merge	1	400606876	4	912033703	5	2729938819	11	3866278528	18	8808112128
Radix	2	401116946	1	912883773	7	2731638889	26	3872278612	46	89628265

Nhận xét: Selection sort và Bubble sort chạy nhiều phép so sánh nhất, Insertion sort chạy ít phép so sánh nhất. Bubble sort chạy với thời gian lâu nhất, Insertion sort chạy với thời gian nhanh nhất.


Nearly Sorted
800000
700000
600000
500000
400000
300000
200000
100000
0
30000
50000
100000
300000
500000
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix


Nearly Sorted
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
Bubble
Heap
Insertion
Selection
Quick
Merge
Radix
30000
50000
100000
300000
500000

5. TỔ CHỨC CODE VÀ GHI CHÚ CODE

Chúng em chia làm 5 file để tổ chức: file MainSourceOfProject.cpp để chạy hàm main và phần command line, file CountCompareSource.cpp chứa những hàm của code đếm số lượng so sánh của thuật toán sort, file TimeOfSorting.cpp chứa những hàm của code đếm thời gian chạy của thuật toán sort, file AlgorithmPresentation.cpp chứa những code của các thuật toán sort, file AllOfFunction.h chứa các prototype và các thư viện để chạy.

Những thư viện chúng em sử dụng: iostream, time.h, fstream, string, string.h , cmath.

6. TRÍCH DẪN VÀ THAM KHẢO

https://vi.wikipedia.org/wiki/Wikipedia

https://www.geeksforgeeks.org/

https://stackoverflow.com/