BÁO CÁO: THUẬT TOÁN SẮP XẾP

Bubble	Insertion	Selection	BinaryInsertion	Heap	Quick	Merger
X	X	X	X	X	X	X

1. Bubble sort:

_ Ý tưởng: 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.

_ Thuật toán:

Giả sử dãy cần sắp xếp có *n* phần tử

Bướ 1: ta so sánh hai phần tử đầu, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau.

Bước 2: Tiếp tục làm như vậy với cặp phần tử thứ hai và thứ ba và tiếp tục cho đến cuối tập hợp dữ liệu, nghĩa là so sánh (và đổi chỗ nếu cần) phần tử thứ *n*-1 với phần tử thứ *n*. Sau bước này phần tử cuối cùng chính là phần tử lớn nhất của dãy.

Bước 3: Sau đó, kiểm tra quay lại so sánh (và đổi chố nếu cần) hai phần tử đầu cho đến khi gặp phần tử thứ *n*-2....

Ghi chú: Nếu trong một lần duyệt, không phải đổi chỗ bất cứ cặp phần tử nào thì danh sách đã được sắp xếp xong.

_ Độ phức tạp: O(n^2)

_ Độ phức tạp không gian O(1)

2. Inserion sort:

_ Ý tưởng: Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

_Thuật toán:

Bước 1: Khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử(phần tử đầu tiên, phần tử có chỉ số 0)

Bước 2: Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử.

Bước 3: Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

_ Độ phức tạp: O(n^2)

_ Độ phức tạp không gian O(1)

3. Selection sort

_ Ý tưởng:

Thuật toán selection sort sắp xếp một mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất(giả sử với sắp xếp mảng tăng dần) trong đoạn đoạn chưa được sắp xếp và đổi cho phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng.

_ Thuật toán

Bước 1:Thiết lập giá trị nhỏ nhất (min) về vị trí số 0.

Bước 2: Tạo vòng lặp for để di chuyển ranh giới của sorted list và unsorted list**.**

Bước 3: Tìm phần tử nhỏ nhất trong list chưa được sắp xếp.

Bước 4: Sau khi tìm được phần tử nhỏ nhất thì đổi chỗ phần tử đó với phần tử đầu tiên. Ở bước này chúng ta cần viết một hàm Swap(), hàm này mình sẽ viết ở bên dưới.

Bước 5: Lặp lại cho tới khi list được sắp xếp xong.

_ Độ phức tạp: O(n^2)

_ Độ phức tạp không gian O(1)

4. Binary Insertion Sort

_ Ý tưởng: Giống như insertion sort, thuật toán sắp xếp thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao (bằng cách tìm ra vị trí của nó theo tìm kiếm nhị phân) sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

_Thuật toán:

Bước 1: Khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử(phần tử đầu tiên, phần tử có chỉ số 0)

Bước 2: Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao (bằng cách tìm ra vị trí của nó theo tìm kiếm nhị phân) sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử.

Bước 3: Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

_ Độ phức tạp thời gian: O(nlog(n))

5. Heap sort

_ Thuật toán: Giải thuật Heapsort còn được gọi là giải thuật vun đống dựa trên cấu trúc heap, 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.
- 1 mảng các phần tử chưa được sắp xếp.

Trong mảng chưa được sắp xếp, các phần tử lớn nhất sẽ được tách ra và đưa 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.

_ Thuật toán

Bước 1: Sắp xếp chúng thành một max heap (có thể dạng cấu trúc cây nhị phân) a[0….n-1]

Bước 2:Đổi chỗ phần tử gốc (MAX) với phần tử cuối của mảng (mảng mà đang xây max Heap lúc nãy) là a[0] và a[n-1], sau đó xây lại max Heap dựa trên nhánh mất cân bằng với dãy a[0..n-2] và làm cho đến khi a[0..0]

_ Độ phức tạp thời gian: O(nlog(n))

6. Quick sort

_ Ý tưởng: Thuật toán sắp xếp quick sort là một thuật toán chia để trị (Divide and Conquer algorithm). Nó chọn một phần tử trong mảng làm điểm đánh dấu (pivot). Thuật toán sẽ thực hiện chia mảng thành các mảng con dựa vào pivot

đã chọn. Bên trái nhỏ hơn pivot, bên phải lớn hơn pivot. Và tiếp tục làm với bên trái và bên phải cho đến khi đã sắp xếp

_ Thuật toán: mảng a[0..n-1]

Bước 1: Chọn pivot là phần tử đâu tiên (có thể chọn bất kỳ), sau đó đưa tất cả phân từ nhỏ hơn nó sang trái, lớn hơn sang phải.

Mảng a[0.. pivot ..n-1]

Bước 2: Gọi đệ quy bước 1 cho bên trái và phải của pivot, với điều kiện dừng mảng a[left..right] mà left >= right

_ Độ phức tạp thời gian: O(nlog(n))

_ Độ phức tạp không gian: O(log(n))

7. Merger sort

_ Ý tưởng: một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp

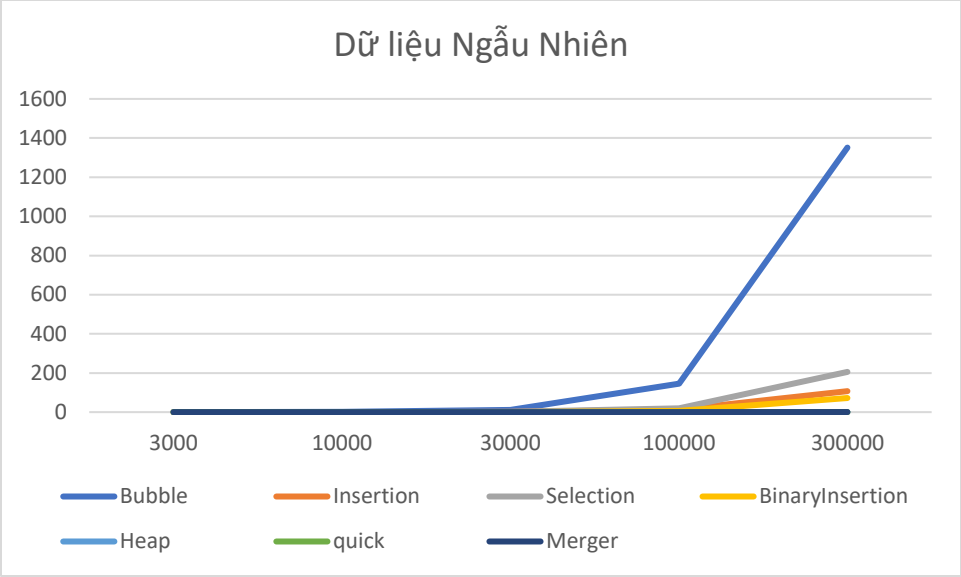
_ Thuật toán:

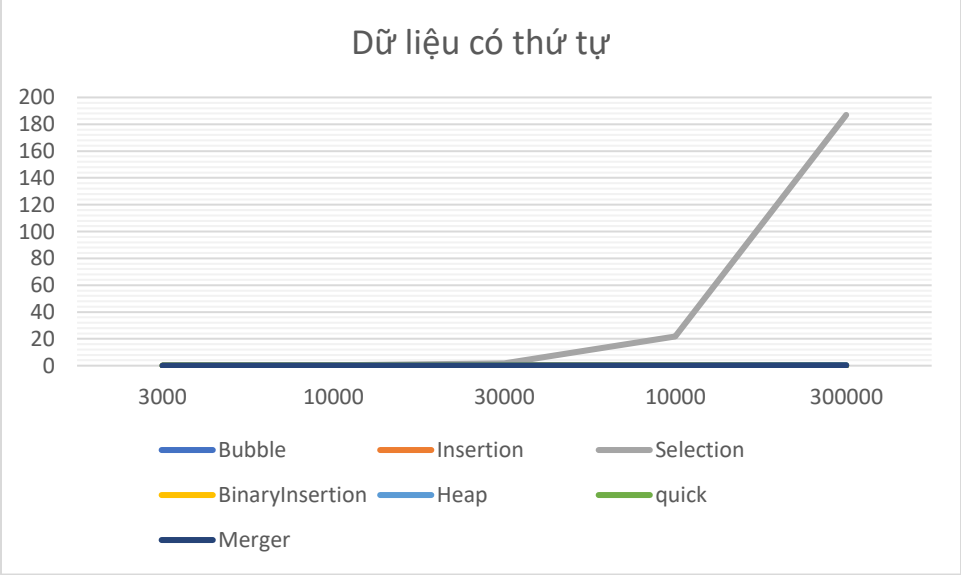
Bước 1: Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa

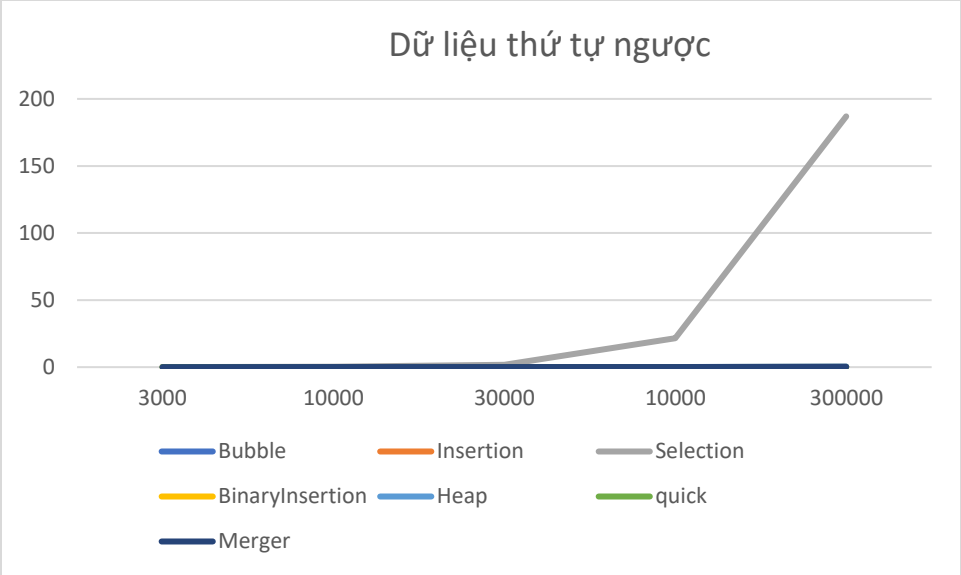
Bước 2: Gọi đệ quy hàm mergeSort cho nửa đầu tiên(hay là qua lại bước 1 thực hiện với nửa đầu tiên)

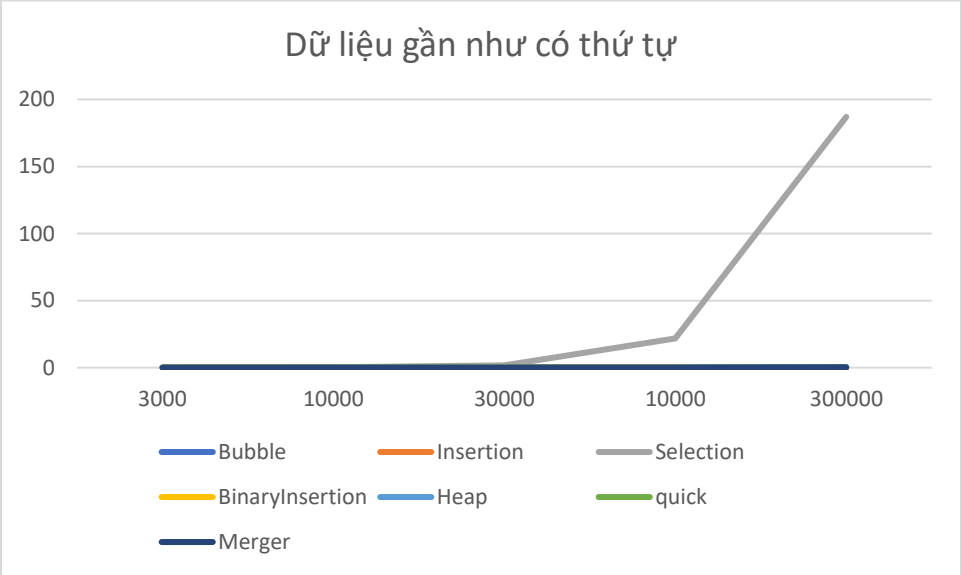
Bước 3: Gọi đệ quy hàm mergeSort cho nửa thứ hai (hay là qua lại bước 1 thực hiện với nửa thứ hai)

Bước 4. Gộp 2 nửa mảng đã sắp xếp ở bước 2 và 3 sao cho vẫn thứ tự tăng dần.


Dữ liệu Ngẫu Nhiên
1600
1400
1200
1000
800
600
400
200
0
3000
10000
30000
100000
300000
Bubble
Insertion
Selection
BinaryInsertion
Heap
quick
Merger


Dữ liệu có thứ tự
200
180
160
140
120
100
80
60
40
20
0
3000
10000
30000
10000
300000
Bubble
Insertion
Selection
BinaryInsertion
Heap
quick
Merger





Nhìn chung thì quick sort nhanh nhất, selection sort chậm nhất

_ Thuật toán ổn định: heap sort, quick sort, merger sort, binary Insertion sort

_ Thuật toán không ổn định: bubble sort, selection sort, insertion sort