**Insertion Sort**

* Ý tưởng: Ta sẽ lấy từng phần tử của mảng rồi tìm vị trí thích hợp để chèn vào
* Các bước:

B1: Lấy một phần tử từ mảng ra

B2: So sánh phần tử đó với các phần tử liền trước nó. Nếu phần tử hiện tại nhỏ hơn phần tử liền trước nó thì di chuyển phần tử liền trước lên một ô và tiếp tục so sánh phần tử hiện tại với phần tử liền trước tiếp theo trong mảng. Lặp lại cho tới khi gặp phần tử liền trước lớn hơn phần tử hiện tại hoặc chạy hết mảng thì ta chèn phần tử hiện tại vào ô trống đã chừa ra.

B3: Lặp lại bước một với phần tử tiếp theo trong mảng cho tới khi mảng kết thúc.

* Độ phức tạp thời gian:
* Worst case: O(n2) khi mảng sắp xếp ngược
* Average case: O(n2)
* Best case: O(n) khi mảng đã được sắp xếp sẵn
* Độ phức tạp không gian: O(1)
* Biến thể:
* Shell sort.
* Binary Insertion Sort: sử dụng Binary Search để tìm vị trí thích hợp để chèn phần tử vào sau đó đôn các phần tử lên cho toi
* Library Sort: chừa ra các ô trống để việc chèn phần tử diễn ra với tốt độ nhanh hơn. Điều này dẫn tới việc giảm tốc độ chạy xuống còn O(nlogn) tuy nhiên tăng chi phí không gian lên O(n).

Ref:

* Insertion Sort: <https://brilliant.org/wiki/insertion>
* Library Sort: <https://ieeexplore.ieee.org/abstract/document/7443165/authors#authors>
* Insertion Sort Variants: <https://en.wikipedia.org/wiki/Insertion_sort>

Selection Sort

* Ý tưởng: Tìm phần tử nhỏ nhất trong mảng sau đó đổi chỗ với phần tử đầu mảng để tạo ra một mảng tăng dần
* Các bước:
* Bước 1: Chọn phần tử đầu mảng
* Bước 2: Tìm phần tử nhỏ nhất trong khoảng phần tử đã chọn cho tới cuối mảng
* Bước 3: Đổi chỗ phần tử đã chọn ở bước 1 với phần tử nhỏ nhất tìm được ở bước 2.
* Bước 4: Lặp lại bước 2 với phần từ tiếp theo của mảng.
* Độ phức tạp thời gian:
* Worst case: O(n2)
* Average case: O(n2)
* Best case: O(n2)
* Độ phức tạp không gian: O(1)
* Biến thể
* Heap Sort.
* Bingo Sort: Sau khi tìm được giá trị nhỏ nhất trong mảng thì ta đem tất cả các phần tử có cùng giá trị đó ra sau mảng. Thuật toán này sẽ chạy tốt hơn Selection Sort nếu có nhiều phần tử có giá trị trùng nhau trong mảng.
* Double Selection Sort: Như cái tên thì thuật toán này chạy như Selection Sort nhưng sẽ tìm giá trị nhỏ nhất và lớn nhất của mảng và đổi chỗ với phần tử đầu và cuối của mảng.
* Ref:
* Selection Sort: <https://nguyenvanhieu.vn/thuat-toan-sap-xep-selection-sort/>
* Bingo Sort: <https://xlinux.nist.gov/dads/HTML/bingosort.html>
* Selection Sort Variant: <https://en.wikipedia.org/wiki/Selection_sort#Variants>

Bubble Sort

* Ý tưởng: Là thuật toán sắp xếp đơn giản nhất. Ta chỉ việc so sánh các phần tử liền kề nhau xem đã đúng vị trí chưa nếu chưa thì đổi vị trí hai phần tử.
* Các bước:
* Bước 1: Lặp lại từ i = 0 tới n.
* Bước 2: Lặp lại từ j = 0 tới n – i – 1.
* Bước 3: So sánh phần tử ở vị trí j với phần tử ở vị trí j + 1. Nếu sai vị trí thì ta đổi chỗ.
* Độ phức tạp thời gian:
* Worst case: O(n2)
* Average case: O(n2)
* Best case: O(n2)
* Độ phức tạp không gian: O(1)
* Biến thể
* Shaker Sort.
* Odd-even Sort: thay vì kiểm tra các phần tử liền kề nhau thì ta kiểm tra các phần tử chẵn-lẻ ở gần nhau. Nếu sai vị trí thì đổi chỗ hai phần tử đó.
* Ref:
* Bubble Sort: <https://www.geeksforgeeks.org/bubble-sort/>
* Odd-even Sort: <https://en.wikipedia.org/wiki/Odd–even_sort>

Heap Sort

* Ý tưởng: Lấy ý tưởng từ cấu trúc Heap. Xem mảng như một cây nhị phân hoàn chỉnh sau đó hiểu chỉnh thành cấu trúc Heap. Phần tử lớn nhất hoặc nhỏ nhất của Heap chắc chắn sẽ là root của Heap. Do đó để sắp xếp ta chỉ cần đổi chỗ root với vị trí cuối cùng sau đó hiệu chỉnh mảng (trừ phần tử cuối cùng) lại thành Heap và lặp lại cho tới khi Heap còn 0 phần tử.
* Các bước:
* Bước 1: Biến Heap thành Max Heap bắt các Heapify từ dưới lên
* Bước 2: Đổi chỗ root và phần tử cuối cùng của mảng sau đó Heapify lại mảng (trừ phần tử cuối cùng).
* Bước 2: Lặp lại bước 2 cho tới khi Heap còn 0 phần tử
* Độ phức tạp thời gian:
* Worst case: O(nlogn)
* Average case: O(nlogn)
* Best case: O(nlogn)
* Độ phức tạp không gian: O(1)
* Biến thể
* Ternary Heap Sort: sử dụng ternary heap (mỗi nhánh có 3 phần tử con).
* Out-of-place Heap Sort: thay vị thay root với vị trí cuối mảng thì ta sẽ thay với giá trị -∞ sau đó tiếp tục Heapify cả mảng thay vì mảng mất một phần tử cuối như trên cho tới khi heap chỉ có các giá trị -∞. Điều này làm giảm số lần so sánh của vì -∞ không thể di chuyển lên được nữa.
* Ref:
* Heap Sort: <https://duongdinh24.com/thuat-toan-heap-sort/>
* Heap Sort Variant: <https://en.wikipedia.org/wiki/Heapsort#Variations>

Merge Sort:

* Ý tưởng: Là thuật toán chia để trị. Ta liên tục chia mảng thành hai nửa mảng con và sau đó kết hợp các mảng con lại theo thứ tự cần sắp xếp để có được mảng sắp xếp hoàn chỉnh.
* Các bước:
* Bước 1: Chia mảng thành hai nửa mảng con. Lặp lại cho tới khi không chia được nữa
* Bước 2: Lần lượt ghép các mảng con đã chia lại theo thứ tự cần sắp xếp cho tới khi mảng hoàn chỉnh.
* Độ phức tạp thời gian:
* Worst case: O(nlogn)
* Average case: O(nlogn)
* Best case: O(nlogn)
* Độ phức tạp không gian: O(n)
* Biến thể
* 3-Way Merge Sort: Chia mảng ra làm 3 thay vì 2.
* Bottom-up Variant: Chia mảng ra thành mảng có độ dài là 1-2-4-8-… cho tới khi độ dài mảng con bằng mảng chính.
* Ref:
* Merge Sort: <https://nguyenvanhieu.vn/thuat-toan-sap-xep-merge-sort/>
* Merge Sort Complexity: <https://stackoverflow.com/questions/10342890/merge-sort-time-and-space-complexity>
* 3-Way Merge Sort: [www.geeksforgeeks.org/3-way-merge-sort](http://www.geeksforgeeks.org/3-way-merge-sort)
* Merge Sort Variants: <https://stackoverflow.com/questions/35201963/differences-in-variations-of-mergesort>