1. **Selection Sort (Sắp xếp chọn):**
2. Ý tưởng: Cho 1 mảng gồm 5 phần tử a1, a2, …, an. Tìm vị trí của phần tử nhỏ nhất trong n phần tử ban đầu và đưa giá trị của phần tử đó về đầu mảng. Tiếp tục với n - 1 phần tử còn lại. Và đến khi còn lại 1 phần tử thì phần dừng và phần tử đó chắc chắn là số lớn nhất. Cuối cùng, mảng đã được sắp xếp theo thứ tự tăng dần.
3. Thuật toán:

* Bước 1: Cho mảng gồm n phần tử a1, a2, …, an và i = 1.
* Bước 2:
* Nếu i nhỏ hơn hoặc bằng n - 1 🡺 Tiếp tục bước kế tiếp.
* Ngược lại 🡺 Chuyển đến bước 6.
* Bước 3: Cho pos = i và j = i + 1.
* Bước 4:
* Nếu j nhỏ hơn hoặc bằng n 🡺 So sánh giá trị apos và aj.
* Nếu apos > aj (cặp phần tử nghịch thế) 🡺 Cập nhật lại giá trị pos bằng với j.
* Tăng j lên thêm 1 và quay lại bước 4.
* Ngược lại 🡺 Chuyển sang bước 5.
* Bước 5:
* Nếu i khác pos thì hoán vị hai giá trị ai và apos.
* Tăng i lên thêm 1 và quay lại bước 2.
* Bước 6: Dừng chương trình.

1. Algorithm’s review:

* Độ phức tạp về thời gian:
* Độ phức tạp về không gian:

1. **Insertion Sort (Sắp xếp chèn):**
2. Ý tưởng: Cho một mảng gồm n phần tử a1, a2, …, an.

* Chia mảng ban đầu thành 2 mảng con a1, a2, …, ai (gọi là mảng B) và ai + 1, ai + 2, …, an (gọi là mảng C). Và mảng a1, a2, …, ai đã có thứ tự (tăng dần).
* Chọn một phần tử từ mảng C. Để tiện ta chọn ai + 1 (tương đương C0). Và chèn vào mảng B sao cho nó vẫn giữ nguyên thứ tự (tăng dần).
* Tìm vị trí k trong đoạn [1, i] thoả ak - 1 ≤ ai + 1 < ak, có 2 trường hợp xảy ra:
* Nếu tồn tại k: dời tất cả phần tử trong đoạn [k, i] về sau (sang phải) một vị trí. Và sau đó cập nhật lại ak bằng với ai + 1.
* Ngược lại: các phần tử trong đoạn từ [1, i + 1] đã có thứ tự.
* Tiếp tục thực hiện cho đến khi chèn hết phần tử từ mảng C sang mảng B. Kết quả là mảng B sẽ chứa tất cả các phần tử ban đầu theo thứ tự tăng dần.

1. Thuật toán:

* Bước 1: Cho mảng gồm n phần tử a1, a2, …, an và i = 2.
* Bước 2:
* Nếu i nhỏ hơn hoặc bằng n 🡺 Tiếp tục bước kế tiếp.
* Ngược lại 🡺 Chuyển đến bước 6.
* Bước 3: Cho x = ai và k = i - 1.
* Bước 4:
* Nếu k lớn hơn hoặc bằng 1 và ak > x.
* Dời phần tử ak về sau một đơn vị (dịch sang phải 1 đơn vị).
* Giảm k xuống 1 giá trị và quay lại bước 4.
* Ngược lại 🡺 Chuyển sang bước 5.
* Bước 5: Cập nhật giá trị ak + 1 bằng với x và quay lại bước 2.
* Bước 6: Dừng chương trình.

1. Algorithm’s review:

* Độ phức tạp về thời gian:
* Độ phức tạp về không gian:

1. **Bubble Sort (Sắp xếp nổi bọt):**
2. Ý tưởng: Cho một mảng gồm n phần tử a1, a2, …, an.

* Bất đầu duyệt từ cuối mảng đến phần tử thứ 2 (an đến a2).
* Trong mỗi lượt duyệt ta sẽ so sánh phần tử tại vị trí i - 1 và i, nếu 2 phần tử này nghịch thế (ai – 1 > ai đối với thứ tự tăng dần) thì sẽ hoán vị hai giá trị cho nhau.
* Kết thúc lượt duyệt này thì phần tử có giá trị nhỏ nhất sẽ được đẩy lên vị trí đầu của dãy số.
* Tiếp tục lặp lại với dãy số gồm n - 1 phần tử sau cùng (vì a1 đã được xếp đúng vị trí sau lượt duyệt đầu tiên) cho đến khi dãy số chỉ còn phần tử cuối cùng là an.
* Khi đó mảng các phần tử từ a1, a2, …, an đã được sắp xếp theo thứ tự tăng dần.

1. Thuật toán:

* Bước 1: Cho mảng gồm n phần tử a1, a2, …, an và i = 1.
* Bước 2:
* Nếu i nhỏ hơn hoặc bằng n - 1 🡺 Tiếp tục bước kế tiếp.
* Ngược lại 🡺 Chuyển sang bước 5.
* Bước 3: Cho j = n.
* Bước 4:
* Nếu j lớn hơn i.
* So sánh cặp phần tử aj - 1 và aj 🡺 Nếu aj - 1 > aj (nghịch thế) 🡺 hoán vị giá trị của hai phần tử aj - 1 và aj .
* Giảm j xuống 1 giá trị và quay lại bước 4.
* Ngược lại 🡺 Tăng giá trị i lên thêm 1 và quay lại bước 2.
* Bước 5: Dừng chương trình.

1. Algorithm’s review:

* Độ phức tạp về thời gian:
* Độ phức tạp về không gian:

1. **Heap Sort (Sắp xếp vun đống):**
2. Ideals:
3. Algorithms:
4. Algorithm’s review:

* Time complexity:
* Space complexity:

1. **Merge Sort (Sắp xếp trộn):**
2. Ideals:
3. Algorithms:
4. Algorithm’s review:

* Time complexity:
* Space complexity:

1. **Quick Sort (Sắp xếp nhanh):**
2. Ideals:
3. Algorithms:
4. Algorithm’s review:

* Time complexity:
* Space complexity:

1. **Radix Sort (Sắp xếp theo cơ số):**
2. Ideals:
3. Algorithms:
4. Algorithm’s review:

* Time complexity:
* Space complexity: