# So sánh về mặt lý thuyết 3 thuật toán: Bubble Sort, Selection Sort và Insertion Sort

* Ba thuật toán trên đều là quadratic running time sorting algorithms, với hàm cận trên đánh giá độ phức tạp là O(n^2), tức là nếu kích thước mảng lớn gấp 2 lần thì thời gian thực hiện lớn gấp 4 lần, dữ liệu đầu vào lớn gấp 10 lần thì thời gian thực hiện lớn gấp 100 lần. Ba thuật toán này không cần thêm không gian lưu trữ để thực hiện.
* Thuật toán Bubble Sort thực hiện việc so sánh 2 phần tử kế tiếp nhau, nếu 2 phần tử này không đúng thứ tự sắp xếp thì thực hiện việc đổi chỗ 2 phần tử này, thực hiện cho đến khi mảng đã được sắp xếp
  + Việc duyệt, so sánh 2 phần tử và thực hiện đổi chỗ O(N)
  + Sau N-1 lần thực hiện thao tác mô tả ở trên, dãy số sẽ đảm bảo được sắp xếp
  + Độ phức tạp trong trường hợp trung bình và xấu nhất O(N\*(N-1)) = O(N^2)
  + Nhận xét: Sau khi kết thúc vòng lặp thứ i, nếu không có 2 phần tử nào được đổi chỗ đồng nghĩa với mảng đã được sắp xếp sau vòng lặp thứ (i-1) => ta có thể break vòng lặp và kết thúc thuật toán, mảng đã được sắp xếp
  + Best case của thuật toán bubble sort sẽ là khi input đầu vào là mảng đã được sắp xếp => O(N) do chỉ thực hiện 1 vòng lặp O(N) qua các phần tử, bằng một biết flag ta có thể biết được là có 2 phần từ nào được đổi chỗ hay không, và do mảng đã sắp xếp nên dĩ nhiên là sẽ không có 2 phần tử nào được đổi chỗ.
  + Worst Case: Input đầu vào mảng được sắp xếp giảm dần
* Thuật toán Selection Sort
  + Ở lần lặp thì i thì sub-array[0..i-1] đã được sắp xếp
  + Lần lặp thì i sẽ tìm vị trí của phần tử nhỏ nhất trong sub-array[i..n-1] và tiến hành đổi chỗ với phần tử thứ i
  + Sau N-1 lần thực hiện các thao tác mô tả ở trên thì mảng được sắp xếp
  + Best, Average và Worst Case đều là O(N^2)
  + Ưu điểm của thuật toán này là chỉ thực hiện tối đa N lần phép toán swap, sẽ có lợi thế lớn nếu như phải thực hiện thao tác IO với ổ đĩa cứng, tuy nhiên phần thực nghiệm dữ liệu được lưu trữ ở bộ nhớ RAM nên ưu điểm này không được thể hiện
  + Worst Case: Input đầu vào mảng được sắp xếp giảm dần
* Thuật toán Insertion Sort
  + Ở lần lặp thì i thì sub-array[0..i-1] đã được sắp xếp
  + Thực hiện vòng lặp tìm vị trí thích hợp để chèn phần tử array[i] vào trong mảng sub-array[0..i-1] để sau khi chèn vào thì mảng sub-array[0..i] được sắp xếp đúng thứ tự
    - Duyệt j = i-1 về 0, nếu arr[j] <= arr[i] thì j+1 là vị trí mà arr[i] được chèn vào => break vòng lặp => best case của vòng lặp là O(1), worst là O(N)
  + Best case là khi input đầu vào mảng đã được sắp xếp sẵn, O(N)
  + Worst case: input đầu vào mảng được sắp xếp giảm dần
  + Average: O(N^2)

# Kết quả thực nghiệm và nhận xét

Cấu hình máy tính đánh giá thực nghiệm 3 thuật toán Bubble Sort, Selection Sort và Insertion Sort:



Các test case, mô tả bộ dữ liệu và thời gian chạy thực tế tương ứng

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Kích thước mảng | Mô tả | Thời gian chạy |
| 1 | n = 10000 | Một hoán vị ngẫu nhiên của n số nguyên [1, 2, 3, …, n] | * Bubble Sort: 259ms * Selection Sort: 91ms * Insertion Sort: 39ms |
| 2 | n = 100000 | Một hoán vị ngẫu nhiên của n số nguyên [1, 2, 3, …, n] | * Bubble Sort: 29549ms * Selection Sort: 3783ms * Insertion Sort: 1692ms |
| 3 | n = 10000 | Hoán vị với các phần tử đã được sắp xếp giảm dần: [n, n-1,...3, 2, 1] | * Bubble Sort: 179ms * Selection Sort: 84ms * Insertion Sort: 55ms |
| 4 | n = 100000 | Hoán vị với các phần tử đã được sắp xếp giảm dần: [n, n-1,...3, 2, 1] | * Bubble Sort: 17685ms * Selection Sort: 8339ms * Insertion Sort: 3525ms |
| 5 | n = 10000 | Hoán vị với các phần tử được sắp xếp tăng dần (1, 2, 3, 4, … n] | * Bubble Sort: 0ms * Selection Sort: 83ms * Insertion Sort: 0ms |
| 6 | n = 100000 | Hoán vị với các phần tử được sắp xếp tăng dần (1, 2, 3, 4, … n] | * Bubble Sort: 0ms * Selection Sort: 8448ms * Insertion Sort: 1ms |
| 7 | n = 10000 | Mảng gồm n số thực được tạo ngẫu nhiên | * Bubble Sort: 232ms * Selection Sort: 83ms * Insertion Sort: 17ms |
| 8 | n = 100000 | Mảng gồm n số thực được tạo ngẫu nhiên | * Bubble Sort: 26408ms * Selection Sort: 8349ms * Insertion Sort: 1740ms |
| 9 | n = 10000 | Mảng gồm n số thực được tạo ngẫu nhiên và sắp xếp giảm dần | * Bubble Sort: 184ms * Selection Sort: 83ms * Insertion Sort: 34ms |
| 10 | n = 100000 | Mảng gồm n số thực được tạo ngẫu nhiên và sắp xếp giảm dần | * Bubble Sort: 17572ms * Selection Sort: 8212ms * Insertion Sort: 3539ms |
| 11 | n = 10000 | Mảng gồm n số thực được tạo ngẫu nhiên và sắp xếp tăng dần | * Bubble Sort: 0ms * Selection Sort: 83ms * Insertion Sort: 0ms |
| 12 | n = 100000 | Mảng gồm n số thực được tạo ngẫu nhiên và sắp xếp tăng dần | * Bubble Sort: 0ms * Selection Sort: 8466ms * Insertion Sort: 1ms |
| 13 | n = 10000 | Mảng gồm n số 1.0 | * Bubble Sort: 0ms * Selection Sort: 83ms * Insertion Sort: 0ms |
| 14 | n = 100000 | Mảng gồm n số 1.0 | * Bubble Sort: 0ms * Selection Sort: 8379ms * Insertion Sort: 0ms |

Nhận xét qua 14 test cases:

* Thuật toán Insertion Sort tỏ ra hiệu quả nhất
* Với dữ liệu được sắp xếp sẵn (tăng dần hoặc tất cả các phần tử bằng nhau), Bubble Sort và Insertion Sort rơi vào best case với thời gian thực hiện O(N), trong khi Selection Sort vẫn là O(N^2), cực kì kém hiệu quả trong trường hợp này
* Tuy nhiên Selection Sort vẫn outperform Bubble Sort trong tất cả trường hợp khác (do thực hiện ít thao tác swap hơn)
* Với các trường hợp dữ liệu được sinh ra ngẫu nhiên, hoặc trong trường hợp dữ liệu được sắp xếp giảm dần, cùng độ phức tạp O(N^2) nhưng thuật toán Bubble Sort tỏ ra kém hiệu quả nhất, Selection Sort đứng ở vị trí thứ 2 và tốt nhất vẫn là Insertion Sort

|  |  |  |  |
| --- | --- | --- | --- |
| Thuật toán | Worst-Case | Avg-Case | Best-Case |
| Bubble Sort | O(N^2) | O(N^2) | O(N) |
| Selection Sort | O(N^2) | O(N^2) | O(N^2) |
| Insertion Sort | O(N^2) | O(N^2) | O(N) |