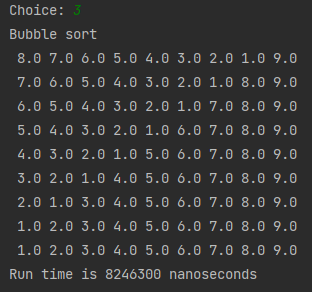
**1. Kết quả thực nghiệm**

***1.1. Với dữ liệu sắp xếp theo thứ tự ngược lại***

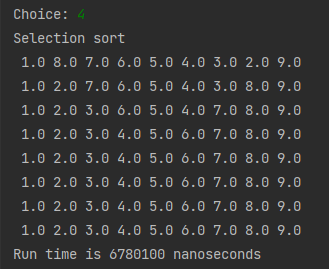
Dữ liệu đầu vào là mảng 9 phần tử nhập từ bàn phím

{9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0}

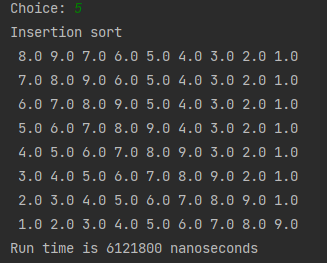
- Kết quả cho thuật toán Sắp xếp Bubble Sort là 8.246.300 nano giây.



- Thời gian cho thuật toán Sắp xếp Selection Sort là 6.780.100 nano giây.



- Thời gian cho thuật toán Sắp xếp Insertion Sort là 6.121.800 nano giây.

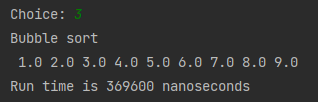


***1.2. Với dữ liệu là mảng đã được sắp xếp***

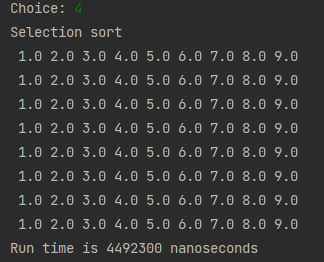
Dữ liệu đầu vào là mảng 9 phần tử nhập từ bàn phím

{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}

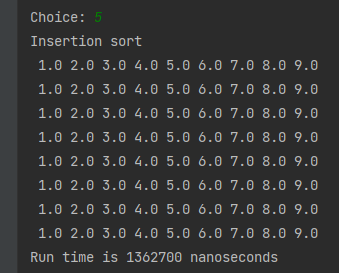
- Thời gian cho thuật toán Sắp xếp Bubble Sort là 369.600 nano giây.



- Thời gian cho thuật toán Sắp xếp Selection Sort là 4.492.300 nano giây.



- Thời gian cho thuật toán Sắp xếp Insertion Sort là 1.362.700 nano giây.

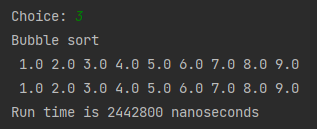


***1.3. Với dữ liệu có xáo trộn ngẫu nhiên***

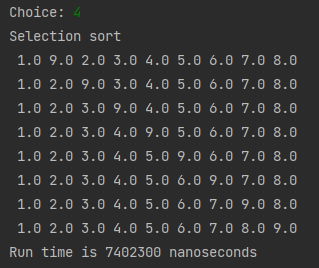
Dữ liệu đầu vào là mảng 9 phần tử nhập từ bàn phím

{9.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}

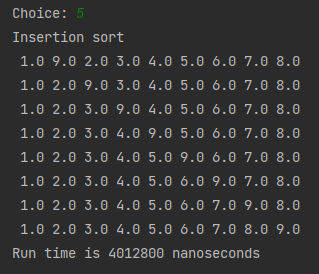
- Thời gian cho thuật toán Sắp xếp Bubble Sort là 2.442.800 nano giây.



- Thời gian cho thuật toán Sắp xếp Selection Sort là 7.402.300 nano giây.



- Thời gian cho thuật toán Sắp xếp Insertion Sort là 4.012.800 nano giây.



**2. Nhận xét kết quả**

*Bảng kết quả (thời gian nano giây)*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Bubble Sort** | **Selection Sort** | **Insertion Sort** |
| Mảng đã được sắp xếp | 369.600 | 4.492.300 | 1.362.700 |
| Mảng sắp xếp theo thứ tự ngược lại | 8.246.300 | 6.780.100 | 6.121.800 |
| Mảng sắp xếp ngẫu nhiên | 2.442.800 | 7.402.300 | 4.012.800 |

Từ bảng kết quả trên ta thấy:

- Thời gian chạy nhỏ nhất cho cả 3 thuật toán trong trường hợp mảng đã được sắp xếp theo thứ tự mong muốn.

- Thời gian chạy lớn nhất của thuật toán Bubble Sort và Insertion Sort trong trường hợp mảng sắp xếp theo thứ tự ngược lại.

- Thời gian chạy lớn nhất của thuật toán Selection Sort trong một trường hợp mảng sắp xếp ngẫu nhiên.

**3. Tính độ phức tạp thời gian**

**3.1. Độ phức tạp thời gian thuật toán Bubble Sort**

public float[] bubbleSort(float b[]) {

boolean end;

int j = b.length - 1;

do{

end = true;

for(int i = 0; i < j; i++) {

if(b[i]>b[i+1]){

swap(i,i+1,b);

end = false;

}

}

j--;

} while(!end);

return b;

}

Với n là số phần tử của mảng cần sắp xếp.

- Trường hợp độ phức tạp thời gian tốt nhất là trường hợp mảng đã được sắp xếp theo thứ tự mong muốn, lúc này thuật toán chỉ chạy n-1 lần phép so sánh (b[i]>b[i+1]).

=> Độ phức tạp thời gian trường hợp tốt nhất là O(n).

- Trường hợp độ phức tạp thời gian xấu nhất là trường hợp mảng sắp xếp theo thứ tự ngược lại. Khi đó phép so sánh (b[i]>b[i+1]) và số phép hoán vị swap(i,i+1,b) thực hiện:

(n-1) + (n-2) + (n -3) + … + 1 = n\*(n-1)/2 (lần)

=> Độ phức tạp thời gian trường hợp xấu nhất là O(n2).

**3.2. Độ phức tạp thời gian thuật toán Selection Sort**

public float[] selectionSort(float b[]) {

long startTime = System.nanoTime();

for(int i = 0; i < b.length - 1; i++) {

int minIndex = i;

for (int j = i +1; j < b.length; j++) {

if(b[j] < b[minIndex]) {minIndex = j;}

}

if (minIndex != i) {

swap(minIndex, i, b);

}

display(b);

}

return b;

}

Với n là số phần tử của mảng cần sắp xếp.

Phép so sánh b[j] < b[minIndex] sẽ luôn thực hiện:

(n-1) + (n-2) + (n -3) + … + 1 = n\*(n-1)/2 (lần)

=> Độ phức tạp thời gian của thuật toán này luôn là O(n2) trong tất cả các trường hợp.

- Trường hợp độ phức tạp thời gian tốt nhất là trường hợp mảng đã được sắp xếp theo thứ tự mong muốn, khi đó không có phép hoán vị nào xảy ra.

- Trường hợp độ phức tạp thời gian xấu nhất là trường hợp phần tử sau khi sắp xếp đứng sau cùng thì lại đứng đầu tiên, các phần tử tiếp theo đã sắp theo thứ tự mong muốn. Khi đó có n-1 phép hoán vị xảy ra.

Ví dụ sắp xếp tăng dần mảng {9, 1, 2, 3, 4, 5, 6, 7, 8}, lúc này sẽ phải thực hiện 8 phép hoán vị.

- Trường hợp độ phức tạp thời gian trung bình là trường hợp mảng sắp xếp theo thứ tự ngược lại khi đó có n/2 phép hoán vị xảy ra.

Ví dụ sắp xếp tăng dần mảng {9, 8, 7, 6, 5, 4, 3, 2, 1}, lúc này phải thực hiện 9/2 = 4 phép hoán vị, đó là: swap(9,1), swap(8,2), swap(7,3), swap(6,4).

**3.3. Độ phức tạp thời gian thuật toán Insertion Sort**

public float[] insertionSort(float b[]) {

for (int i = 1; i < b.length; i++) {

float current = b[i];

int j = i - 1;

while(j>=0 && b[j] >current) {

b[j+1] = b[j];

j--;

}

b[j+1] = current;

}

return b;

}

Với n là số phần tử của mảng cần sắp xếp.

- Trường hợp độ phức tạp thời gian tốt nhất là trường hợp mảng đã được sắp xếp, khi đó sẽ xảy ra n-1 phép so sánh (j>=0 && b[j] >current).

=> Độ phức tạp thời gian trong trường hợp tốt nhất là O(n).

- Trường hợp độ phức tạp thời gian xấu nhất là trường hợp mảng sắp xếp theo thứ tự ngược lại khi đó có số phép so sánh: 1+2+3+…+(n-1) = n\*(n-1)/2.

=> Độ phức tạp thời gian trong trường hợp xấu nhất là O(n2).