Analysing Sorts

Dữ liệu đã được sắp xếp tăng dần (2, 5, 8, 10, 12)	
Sort Name	Time (nano second)
Bubble Sort	2554
Select Sort	5507
Insert Sort	3756

Dữ liệu đã được sắp xếp giảm dần (2, 5, 8, 10, 12)	
Sort Name	Time nano second)
Bubble Sort	8004
Select Sort	6383
Insert Sort	6910

Dữ liệu đã được sắp xếp ngẫu nhiên (2, 5, 8, 10, 12)	
Sort Name	Time
Bubble Sort	7001
Select Sort	5818
Insert Sort	5437

Nhận Xét:

- với dữ liệu sắp xếp tăng dần: bubble sort là hiệu quả nhất, nhanh hơn 32% so với Insert Sort và 50% so với Select Sort.
- với dữ liệu sắp xếp giảm dần thì bubble sort lại kém hiệu quả nhất.
- Trong trường hợp ngẫu nhiên thì Insert sort là thuật tóan hiệu quả hơn, trong thực tế nó là thuật toán hiệu quả nhất với các mảng có kich thước nhỏ.

Bubble Sort Best Case

Best case

2	5	8	10	12

Time of compare: n - 1 = O(n)
Time of swap: 0 = O(1)

Đầu vào trường hợp tốt nhất là một mảng đã được sắp xếp. Khi đó, sắp xếp chèn có thời gian chạy tuyến tính (tức là, O (*n*)) (nếu ta sử dụng cờ flag để kiểm tra sau mỗi vòng lặp xem mảng đã được sắp xếp chưa).

```
bubbleSort(float a[], n):
  for (int i = 1; i < n - 1; i++):
    flag = 0
    for (j = 0; j < n -1; j++):
      if(a[j]>a[j+1]):
        swap(a[j], a[j+1])
        flag = 1;
      if(flag == 0): return
```

Bubble Sort Worst Case

worst case

12	10	8	5	2

Time of compare: $O(n^2)$
Time of swap: $O(n^2)$

Đầu vào trường hợp xấu nhất là một mảng được sắp xếp theo thứ tự ngược lại. Khi đó mọi lần lặp của vòng lặp bên trong sẽ quét và swap hai phần tử liền kề nhau . Điều này cho phép sắp xếp chèn một thời gian chạy bậc hai (tức là, O (n^2)).

```
bubbleSort(float a[], n):
  for (int i = 1; i < n - 1; i++):
    flag = 0
    for (j = 0; j < n -1; j++):
      if(a[j]>a[j+1]):
        swap(a[j], a[j+1])
        flag = 1;
      if(flag == 0): return
```

Bubble Sort Average Case

Trường hợp trung bình : O (n^2)

```
bubbleSort(float a[], n):
  for (int i = 1; i < n - 1; i++):
    flag = 0
    for (j = 0; j < n -1; j++):
      if(a[j]>a[j+1]):
        swap(a[j], a[j+1])
        flag = 1;
      if(flag == 0): return
```

Selection Sort Best Case, Worst Case, Average Case

Best case

2	5	8	10	12

Worst case

12	10	8	5	2

Time of compare: = $O(n^2)$
Time of swap: 1 = O(1)

```
selectionSort(float a[], n):
  int j, j, k
  for(i = 0; i < n - 1; i++):
    for(j=k=i; j < n; j++):
      if a[j] < a[k]:
        k = j
    swap(a[i], a[j])
```

Trong mỗi vòng lặp thuật toán cần tìm phần tử có giá trị nhỏ nhất rồi đặt nó vào đúng vị trí sắp xếp, điều này làm cho thuật toán có độ phức tạp $O(n^2)$

Insertion Sort Best Case

Best case

2	5	8	10	12

Time of compare: n - 1 = O(n)
Time of swap: 0 = O(1)

```
insertionSort(float a[], n):
  for (int i = 1; i < n; i++):
    k = i - 1
    temp = a[i]
    while (k >= 0 && temp < a[k]):
      a[k + 1] = a[k]
      k--
    a[k + 1] = temp
```

Đầu vào trường hợp tốt nhất là một mảng đã được sắp xếp. Trong trường hợp này, sắp xếp chèn có thời gian chạy tuyến tính (tức là, O (*n*)). Trong mỗi lần lặp, phần tử còn lại đầu tiên của đầu vào chỉ được so sánh với phần tử ngoài cùng bên phải của phần con đã sắp xếp của mảng.

Insertion Sort Worst Case

Worst case

12	10	8	5	2

Time of compare: $O(n^2)$
Time of swap: $O(n^2)$

```
insertionSort(float a[], n):
  for (int i = 1; i < n; i++):
    k = i - 1
    temp = a[i]
    while (k >= 0 && temp < a[k]):
      a[k + 1] = a[k]
      k--
    a[k + 1] = temp
```

Đầu vào trường hợp xấu nhất là một mảng được sắp xếp theo thứ tự ngược lại. Khi đó mọi lần lặp của vòng lặp bên trong sẽ quét và dịch chuyển toàn bộ phần con đã được sắp xếp của mảng trước khi chèn phần tử tiếp theo. Điều này cho phép sắp xếp chèn một thời gian chạy bậc hai (tức là, O (n^2)).

Insertion Sort Average Case

Trường hợp trung bình cũng là bậc hai.

Sắp xếp chèn là một trong những thuật toán nhanh nhất để sắp xếp các mảng rất nhỏ, nhỏ hơn một ngưỡng nhất định (phổ biến là mảng có khoảng 10 phần tử).

```
insertionSort(float a[], n):
  for (int i = 1; i < n; i++):
    k = i - 1
    temp = a[i]
    while (k >= 0 && temp < a[k]):
      a[k + 1] = a[k]
      k--
    a[k + 1] = temp
```

Comparing Algorithm Sorts

	Bubble Sort	Insertion Sort	Selection Sort
Min compare	O(n) (Ascending)	O(n) (Ascending)	$O(n^2)$
Max compare	$O(n^2)$ (Descending)	$O(n^2)$ (Descending)	$O(n^2)$
Min swap	O(1) (Ascending)	O(1) (Ascending)	$O(n^2)$
Max swap	$O(n^2)$ (Descending)	$O(n^2)$ (Descending)	$O(n^2)$