**Ôn tập phần Độ phức tạp**

**Câu 1:** Đánh giá độ phức tạp của đoạn code chạy trên C++ sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | int search(int a[], int n, int x)  { n lần để so sánh a [i] với x (trường hợp xấu nhất)  int i = 0;  while (i < n && a[i]!= x) {  i++; n lần để tăng i (trường hợp xấu nhất)  if (i == n)  return -1; n lần để so sánh chữ “i” với chữ “n”  }  return i;  } |

Trường hợp tốt nhất: a [0] = x, so sánh a [i] với x 1 lần => độ phức tạp: O (1)

Trường hợp xấu nhất: i = n (không có phần tử nào trong mảng bằng x) => độ phức tạp: O (3n) = O (n)

**=> Độ phức tạp của thuật toán là O (n)**

**Câu 2:** Hãy đánh giá độ phức tạp của đoạn chương trình sau:

int Timkiem(int a[], int n, int x )

{

for(int i = 0; i < n ; i++) n lần để tăng i (trường hợp xấu nhất)

{

if(a[i] == x) n lần để so sánh a [i] với x (trường hợp xấu nhất)

return i;

}

return -1;

}

+) Trường hợp tốt nhất: a [0] = x, so sánh a [i] với x 1 lần => độ phức tạp: O (1)

+) Trường hợp xấu nhất: a [n-1] = x hoặc i = n (không có phần tử nào trong mảng bằng x) => độ phức tạp: O (2n) = O (n)

=> Độ phức tạp của thuật toán là O (n)

**Câu 3:** Hãy đánh giá độ phức tạp của đoạn chương trình sau:

*for (i = 0; i < n; i++)*

*for (j = 0; j < m; j++) n \* m thời gian*

*if (a[ i ][ j ] = = x) return 1;*

*return -1;*

Lần đầu tiên CHO ĐĂNG NHẬP: tôi bắt đầu từ 0 đến n-1: n lần

Vòng lặp for thứ hai: j bắt đầu từ 0 đến m-1: m lần

=> T (n) = n \* m = O (n \* m)

Vì vậy, độ phức tạp của thuật toán là O (n \* m)

**Câu 4:** Hãy đánh giá độ phức tạp của đoạn chương trình sau:

*sum = 0;  
for( i = 0; i < n ; i++)*

*for( j = 0; j < i ; j++)*

*sum++;*

Lần đầu tiên CHO ĐĂNG NHẬP: tôi bắt đầu từ 0 đến n-1: n lần

Vòng lặp for thứ 2:

+) i=0, j=i: 0 time

+) i=1, j bắt đầu từ 0 to 0: 1 lần

+) i=2, j bắt đầu từ 0 to 1: 2 lần

+) i=3, j bắt đầu từ 0 to 2: 3 lần

…

+) i=n-1, j bắt đầu từ 0 đến n-2: n-1 lần

Vì vậy T(n) = n\*(n-1) = n2 – n = O(n2)

Tóm lại, độ phức tạp của thuật toán là :O(n2)