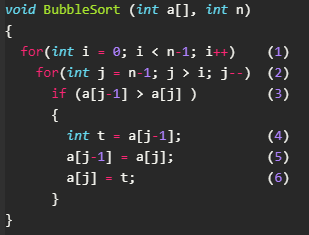


Ta thấy lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm **LinearSearch** là O(max(f(n), g(n)))

Ở lệnh (4) có O(1)

Lồng trong lệnh (1) là lệnh (2), lồng trong lệnh (2) là lệnh (3). Ở lệnh (2) có O(1) - ở trường tốt nhất, và O(n) - ở trường hợp xấu nhất khi tất cả các phần tử của mảng a đều khác x.

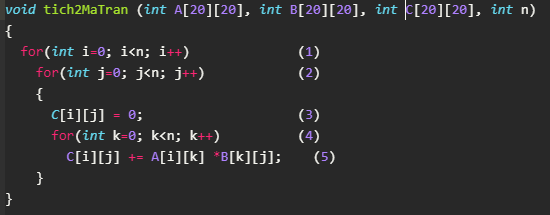
Suy ra: T(n) = O(1.n) = O(n)



Ta thấy toàn bộ hàm **BubbleSort** chỉ gồm một lệnh lặp (1). Lồng trong lệnh (1) là lệnh (2). Lồng trong lệnh (2) là lệnh (3) và lồng trong lệnh (3) là 3 lệnh nối tiếp nhau lần lượt là (4), (5), (6). Chúng ta tiền hành tính độ phức tạp trừ trong ra:

Các lệnh (4), (5), (6) chỉ đều tốn O(1) thời gian, việc so sánh a[j-1] > a[j] chỉ tốn O(1) thời gian, do đó lệnh (3) tốn O(1) thời gian

Vòng lặp (2) thực hiện (n-i) lần, mỗi lần O(1), do đó vòng lặp (2) tốn O((n-i).1) = O(n-i). Vòng lặp (2) lặp có i chạy từ 1 đến n-1 nên thời gian thực hiện cảu vòng lặp (1) là



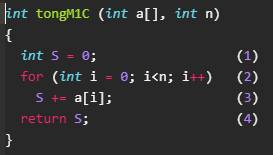
Trong toàn bộ hàm **tich2MaTran** chỉ gồm lệnh lặp (1). Lồng trong lệnh (1) có lệnh lặp (2). Trong lồng lệnh (2) có 2 lệnh nối tiếp nhau lần lượt là (3), (4). Lồng trong lệnh 4 có 1 lệnh. Ta thực hiện tính độ phức tạp từ trong ra:

Lệnh (5) chỉ tốn O(1) thời gian. Nhưng lệnh (5) được lập n lần, do đó độ phức tạp của lệnh (4) là O(n.1) = O(n).

Ở lênh (3) chỉ tốn O(1) thời gian. Lệnh (2) tốn O(n) thời gian, do đó độ phức tạp của lệnh (2) là O(n.n.1) = O(n2)

Ở lệnh lặp (1) tốn O(n)

Suy ra T(n) = O(n.n2) = O(n3)



Ta thấy ở hàm **tongM1C** có 3 lệnh liên tiếp nhau lần lượt là (1), (2), (4). Do đó độ phức tạp của hàm **tongM1C** chính là độ phức tạp lớn nhất trong 3 lệnh đó.

Ở lệnh (1) và (4) đều chỉ tốn O(1)

Ở lệnh lặp (2) lặp có i chạy từ 0 đến n-1. Suy ra độ phức tạp của lệnh (2) là O(n)

Suy ra T(n) = O(n.1.1) = O(n)