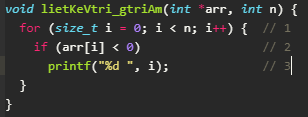


Ta thầy toàn bộ hàm **lietKeGtriChan** chỉ gồm một lệnh lặp (1). Trong lệnh lặp (1) có lệnh điều kiện (2) và trong lệnh (2) có lệnh (3). Chúng ta bắt đầu tính độ phứt tạp từ trong ra ngoài:

Ở lệnh (3) chỉ tốn O(1) thời gian, việc so sánh **arr[i]%2==0** chỉ tốn O(1) thời gian. Do đó lệnh (2) tốn O(1) thời gian.

Ở vòng lặp (1) do lặp n lần nên tốn O(n) thời gian

* T(n) = O(1.n) = O(n)

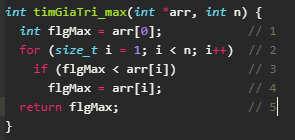


Ta thầy toàn bộ hàm **lietKeVtri\_gtriAm** chỉ gồm một lệnh lặp (1). Trong lệnh lặp (1) có lệnh điều kiện (2) và trong lệnh (2) có lệnh (3). Chúng ta bắt đầu tính độ phứt tạp từ trong ra ngoài:

Ở lệnh (3) chỉ tốn O(1) thời gian, việc so sánh **arr[i]<0** chỉ tốn O(1) thời gian. Do đó lệnh (2) tốn O(1) thời gian.

Ở vòng lặp (1) do lặp n lần nên tốn O(1.n) = O(n)

* T(n) = O(n)



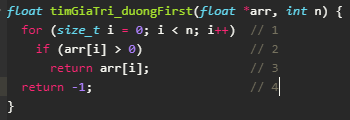
Ở hàm **timGiaTri\_max** có lệnh (1), (2), (5) liên tiếp nhau nên độ phức tạp của hàm chính là độ phức tạp lớn nhất trong 3 lệnh đó:

Ở lệnh gán (1) và lệnh return (5) đều chỉ tốn O(1) thời gian

Lồng trong lệnh lặp (2) có lệnh (3) và trong lệnh (3) có lệnh (4). Ở lệnh (4) chỉ tốn O(1) thời gian, việc so sánh **flgMax<arr[i]** chỉ tốn O(1) thời gian. Do đó lệnh (3) tốn O(1) thời gian.

Ở lệnh lặp (2) do lặp n lần nên lệnh (2) tốn O(1.n)

* T(n) = O(max(1, n, 1)) = O(n)

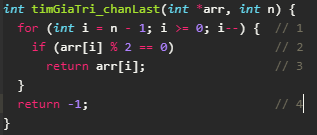


Ta thấy lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm **timGiaTri\_duongFirst** là O(max(f(n), g(n)))

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 arr <= 0.

* T(n) = O(1.n) = O(n)

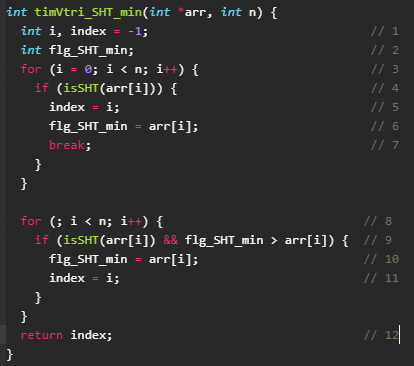


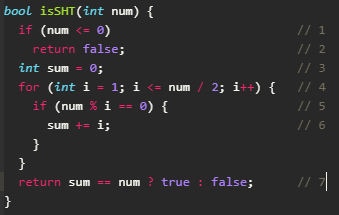
Ta thấy lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm **timGiaTri\_chanLast** là O(max(f(n), g(n)))

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 arr không chi hết cho 2.

* T(n) = O(1.n) = O(n)





Trong hàm **isSHT** có (1), (3), (4), (7) liên tiếp nhau. Do đó độ phức tạp của hàm chính là độ phức tạp lớn nhất trong 4 lệnh trên

Ở lệnh (1), (3), (7) đều chỉ tốn O(1) thời gian

Trong lồng lệnh (4) có lệnh điều kiện (5), trong lệnh (5) có lệnh (6). Lệnh (6) chỉ tốn O(1), việc so sánh **num%i==0** tốn O(1). Lệnh lập (4) lập nên O() ở trường hợp xấu nhất. Do đó độ phức tập của lệnh (4) là O() = O(n)

* T(n) = O(max(n, 1, 1, 1)) = O(n)

Trong hàm **timVtri\_SHT\_min** có lệnh (1), (2), (3), (8), (12) liên tiếp nhau. Do đó độ phứt tạp của hàm chình là độ phức tạp lớn nhất trong 5 lệnh trên

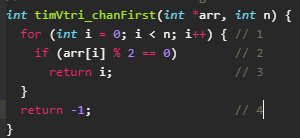
Các lệnh (1), (2) và (12) đều tốn O(1) thời gian thực hiện

Ở lệnh lặp (3), (8) phụ thuộc vào giá trị của i hiện hành.

Trong lồng lệnh (3) có lệnh lệnh (4), trong lệnh (4) có 3 lệnh (5), (6), (7) liên tiếp nhau. Cả 3 lệnh (5), (6), (7) đều tốn O(1) thời gian. Việc so sánh **isSHT(arr[i])** tốn O(1) thời gian ở trường hợp tốt nhất, O(n) ở trường hợp xấu nhất. Lệnh (4) lặp có i từ 0 đến n nên thời gian thực hiện của lệnh (3) = O(n2)

Ở lệnh lặp (8) thực hiện tùy thuộc vào giá trị i hiện hành. Do đó lệnh lặp (8) tốn O(1) thời gian ở trường hợp tốt nhất và O(n2) (tính độ phức tạp tương tự như tính độ phức tạp ở lệnh (3))

* T(n) = O(max(n2, n2, 1, 1, 1)) = O(n2)

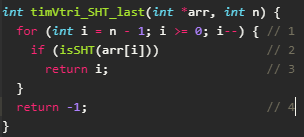


Ta thấy lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm **timVtri\_chanFirst** là O(max(f(n), g(n)))

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 arr không chi hết cho 2.

* T(n) = O(1.n) = O(n)

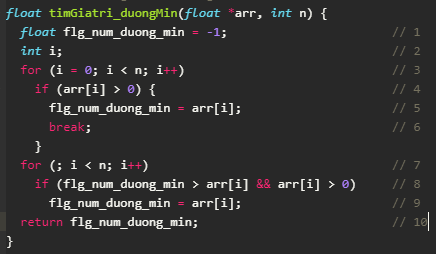


Ta thấy lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm **timVtri\_SHT\_last** là O(max(f(n), g(n)))

Ở lệnh (4) tốn O(1) thời gian

Trong lồng lệnh (1) có lệnh điều kiện (2), lồng trong lệnh (2) chứa lệnh (3). Ở lệnh (3) chỉ tốn O(1), ở lệnh điều kiện (2) tốn O(n). Lệnh (2) lặp có i từ n-1 đến 0 mỗi lần lặp thực hiện 1 lần do đó lệnh lặp (1) tốn O(n.(n)) = O(n2)

* T(n) = O(n2)



Ta thầy ở hàm **timGiatri\_duongMin** có lệnh (1), (2), (3), (7) và (10) liên tiếp nhau. Do đó độ phức tạp của hàm chính là độ phức tạp lớn nhất của các lệnh đó:

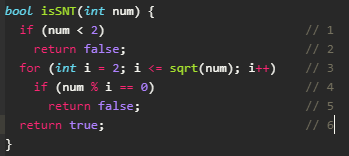
Ở các lệnh gán (1), khai bao (2), return (10) đều tốn O(1) thời gian thực hiện.

Ở lệnh lặp (3) và (7) phụ thuộc vào giá trị i hiện hành

Trong lệnh lặp (3) có lệnh điều kiện (4), trong lệnh (4) có 2 lệnh (5), (6) liên tiếp nhau. Cả 2 lệnh (5) và (6) đều chỉ tốn O(1) thời gian thực hiện. Đối với lệnh điều kiện (4) tốn O(1). Do đó lệnh lặp (3) có thể tốn O(1) ở trường hợp tốt nhất và O(n) ở trường hợp xấu nhất.

Ở lệnh lặp (7) thực hiện tùy thuộc vào giá trị i hiện hành. Do đó lệnh lặp (7) tốn O(1) thời gian ở trường hợp tốt nhất và O(n) (tính độ phức tạp tương tự như tính độ phức tạp ở lệnh (3))

* T(n) = O(max(n, n, 1, 1, 1)) = O(n)



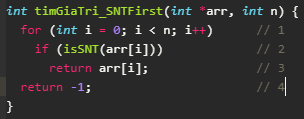
Ở hàm **isSNT** có lệnh (1), (3) và (6) liên tiếp nhau. Do đó độ phức tạp của hàm chính là độ phức tạp lớn nhất trong 3 lệnh trên

Đối với lệnh return (6) chỉ tốn O(1)

Trong lệnh điều kiện (1) có lệnh return (2) tốn O(1) thời gian thực hiện

Lồng trong vòng lặp (3) có lệnh điều kiện (4), trong lệnh (4) có lệnh return (5) tốn O(1). Lệnh điều kiện (4) được lập có i từ 2 đến nên do đó độ phức tạp ở lệnh (3) có O(1) khi trường hợp tốt nhất và O()

* T(n) = O()



Ta thấy trong hàm **timGiaTri\_SNTFirst** có 2 lệnh (1), (4) liên tiếp nhau.

Ở lệnh (4) tốn O(1)

Trong lệnh lặp (1) có lệnh điều kiện (2) và trong lệnh (2) có lệnh return (3) có O(1). Ở lệnh (2) tốn O().

Ở lệnh lặp (1) thực hiện n lần mỗi lần thực hiện lần ở trường hợp xấu nhất, O(1) ở trường hợp tốt nhất. Do đó độ phức tạp của lệnh (1) là O(n)

* T(n) = 1 + n = O(n)