Kodun zaman karmaşıklığını adım adım hesaplamak için her bir fonksiyonun karmaşıklığını analiz edelim:

generate fonksiyonu:

size kez döngüye giriyor ve her bir döngü adımında bir işlem gerçekleştiriyor: bir dizi elemanına rastgele bir değer atama.

Bu nedenle, bu fonksiyonun karmaşıklığı O(size) olarak hesaplanabilir.

function1 fonksiyonu:

İki adet for döngüsü kullanılıyor. İç içe döngüler olduğu için ikinci döngü, ilk döngünün tekrar sayısına bağlı olarak çalışır.

Her bir döngü adımında, sabit sayıda işlem gerçekleştirilir (elemanları karşılaştırma ve takas etme).

Bu nedenle, bu fonksiyonun karmaşıklığı O(size^2) olarak hesaplanabilir.

function2 fonksiyonu:

Bir for döngüsü kullanılıyor, bu nedenle döngü, size kez çalışır.

Her bir döngü adımında sabit sayıda işlem yapılır.

Bu nedenle, bu fonksiyonun karmaşıklığı O(size) olarak hesaplanabilir.

function3 fonksiyonu:

İki adet for döngüsü kullanılıyor, bu nedenle ikinci döngü, birinci döngünün tekrar sayısına bağlı olarak çalışır.

İç içe bir for döngüsü daha var, bu nedenle bu döngü de birinci ve ikinci döngülere bağlı olarak çalışır.

Her bir döngü adımında sabit sayıda işlem yapılır.

Bu nedenle, bu fonksiyonun karmaşıklığı O(size^3) olarak hesaplanabilir.

print1, print2, print3 fonksiyonları:

Üç adet for döngüsü kullanılıyor, ancak bu döngüler sabit sayıda işlem yapar.

Bu nedenle, bu fonksiyonların karmaşıklığı O(1) olarak hesaplanabilir.

Toplam karmaşıklık:

generate: O(size)

function1: O(size^2)

function2: O(size)

function3: O(size^3)

print1, print2, print3: O(1)

Ana fonksiyonda her bir fonksiyon sırasıyla çağrıldığından, toplam karmaşıklık en yüksek karmaşıklığı olan fonksiyonla aynıdır, yani O(size^3).

Not: size değişkeni G\_SIZE ile aynı olduğu için, kodun zaman karmaşıklığı O(G\_SIZE^3) olarak da ifade edilebilir.