Verilen C++ kodunda;

generate: Rastgele değerlere sahip bir A\_SIZE tam sayısı oluşturur.

function1: Bubble sort algoritmasını kullanarak yapılan bir sıralama algoritmasını generate ile oluşturulan diziyi sıralar.

function2: Burada Kadane algoritması kullanılmıştır. Kadane algoritması max alt dizi toplamını bulmak için kullanılır. Rastgele oluşturulan dizinin max alt dizi toplamlarının algoritmasının hesabıdır.

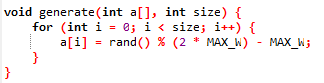
function3: Herhangi iki düğüm arasındaki en kısa yolu bulmak için Floyd-Warshall algoritmasını kullanmış bir fonksiyondur. Burada en kısa yolu bulmak için Dijkstra algoritmasını da kullanabiliriz.

print1 : Bir tam sayı dizisi yazdırmak için oluşturulmuştur.

print2: Matris yazdırma işlemi için oluşturulmuştur.

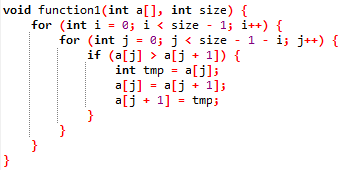
print3: Function3’te bulunan en kısa yolun matrisini yazdırmak için oluşturulmuştur.

time değişkeni oluşturup çalışma zamanı hesabı ile kodumuz sonlandırılmıştır.



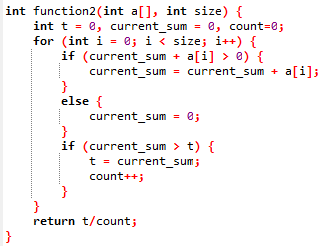
O(1) O(n)

size=n dersek burada her bir döngü adımında sabit bir işlem yapılmakta. Sabit işlem için zaman karmaşıklığı O(1)’dir. For döngüsünün zaman karmaşıklığı ise O(n)’dir. Buradaki kod parçacığının tamamının zaman karmaşıklığı ise O(n)\*O(1)=O(n)’dir.



 (n-1)+(n-2)…+2+1 n-1

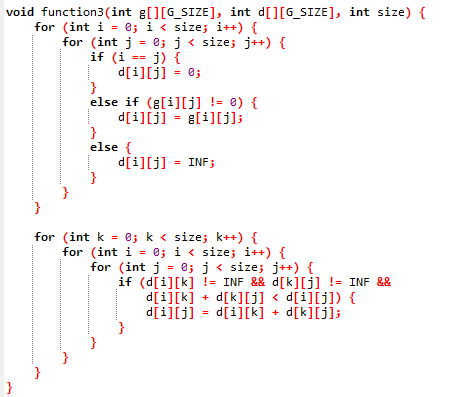
Burada size=n ilk başta da demiştik. İçteki döngüde her bir satır sabit sayıda işlem yapar bu da O(1) zaman karmaşıklığına sahiptir. Dıştaki döngü n-1 kez işlem yapar ve her döngü adımında içteki döngüde n-1,n-2,n-3…2,1 kez çalışır. Toplam işlem sayısı ise (n-1+1)\*(n-1)/2=n\*(n-1)/2 bu da O() zaman karmaşıklığı olur.





 O(n)

 Burada döngü n kez çalışır her bir döngü sabit işlem yapar. Sabit işlemlerin zaman karmaşıklığı O(1)’dir. İşlemler ise n kez tekrarlanacağından bu kod parçacığının zaman karmaşıklığı O(n)’dir.

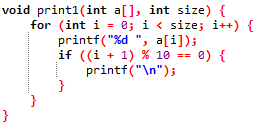
 Burada size=n dersek içteki for döngüsü ne kez çalışacaktır. Dışta bulunan for döngüsü de n kez çalışacaktır. İçerisinde bulunan işlemler de sabit çalışmalara sahiptir yani O(1). Buradaki zaman karmaşıklığı n\*n den O()’dir.

O()

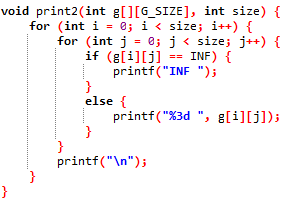


Burada size=n ve içi içe bulunan üç tane for döngümüz bulunmaktadır ve her for döngüsü n kez çalışacaktır. İf içerisinde bulunan işlemler ise sabit çalışacağından o işlemlerin zaman karmaşıklığı O(1)’dir. İç içe bulunan for döngülerinin zaman karmaşıklığı ise n\*n\*n den O() olur.

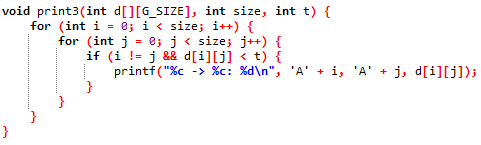
Fonksiyonun zaman karmaşıklığına bakarsak ilk işlemimizin zaman karmaşıklığı O() ve ikinci kısımda zaman karmaşıklığı O()’tür. Toplam zaman karmaşıklığı ise burada O() olur.



Yukarıdaki kod parçacığımızda for içindeki işlemler sabit işlemlerdir ve sabit işlemlerimizin zaman karmaşıklığı O(1)’dir. For dögümüz ise n kez çalışacaktır ve zaman karmaşıklığımız O(n) olur.



Yukarıda görülen kod parçacığında iki adet iç içe for döngüsü bulunmaktadır. Bu for döngülerinin her biri n kez çalışacaktır. Her bir iç döngü adımında ise sabit zamanlı işlem yapılır ve bu işlemin zaman karmaşıklığı O(1)’dir. Bu kod parçacığımızın toplam zaman karmaşıklığı O()’dir.

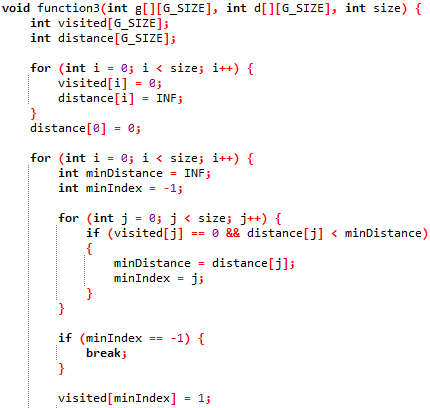


Yukarıda görülen kod parçacığında iki adet iç içe for döngüsü bulunmaktadır. Her bir for döngüsü n kez çalışacaktır. Yazdırma işlemi olan printf ise sabit zamanlı işlemdir ve zaman karmaşıklığı O(1)’dir. Toplam zaman karmaşıklığımız ise n\*n den O()’dir.

Yukarıdan aşağı kodun toplam çalışma zamanına bakılacak olursa O(n)+O()+O(n)+O()+O(n)+O()+O() bu ifadenin sadeleştirilmiş hali olarak baktığımızda toplam zaman karmaşıklığımız O() olacaktır.

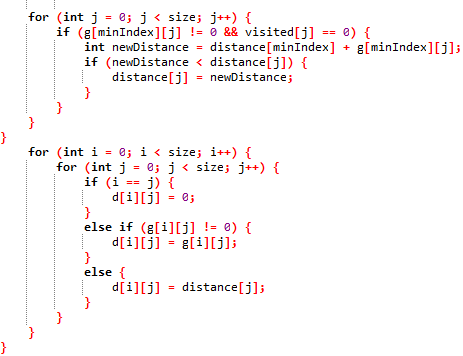
Verilen kodumuzda function3’te kullanılan Floyd-Warshall algoritmasının yerine en kısa yolu bulmak için kullanılan farklı algoritmalar da mevcuttur. Mesela bunlar Dijkstra algoritması , Bellman-Ford algoritması , A\* arama algoritması ve Johnson algoritmalarıdır. Function3 ‘ü geliştirmek için ben dijkstra algoritmasını kullandım.

Kodum sadece function3 kısmında değişiklik yaptım diğer kısımların zaman karmaşıkları aynı bu yüzden function3’ün zaman karmaşıklığına bakalım:





O()





O()

Function3’teki zaman karmaşıklığında size=n dedim. Fonksiyonumuzdaki ilk for döngüsü n kez çalışacaktır bu yüzden zaman karmaşıklığı O(n)’dir.

İkinci for döngüsüne baktığımız zaman iç içe for döngüleri mevcuttur. Burada dıştaki for döngüsü n kez çalışacak ve içteki for döngüsü de n kez çalışacaktır. Zaman karmaşıklığımız ise n\*n ‘den O() olmaktadır.

Minumum mesafeye sahip mesafe güncelleyen for döngümüz de n kez çalışmakta ve zaman karmaşıklığı O(n) olmaktadır.

Son olarak d dizisini dolduran iç içe for döngümüz bulunmaktadır. İçteki for döngümüz n kez çalışacak ve dıştaki döngümüz de n kez çalışacaktır. Zaman karmaşıklığımız ise n\*n ‘den O() olacaktır.

Bu fonksiyonumuz için toplam zaman karmaşıklığına bakarak sadeleşmiş bir şekilde yazdığımızda ise zaman karmaşıklığımız O() olmaktadır.

Kodumuzun geneline baktığımızda sadece function3’te değişiklik yaptım. Değişiklik yapmadan önce zaman karmaşıklığımız O() iken değişiklik yaptıktan sonra zaman karmaşıklığımız O() olmaktadır.

Çalışma zamanlarının karşılaştırılmasına baktığımız zaman ise kodumuzda geliştirmeler yapılmadan önceki halinde ve geliştirilmeden yapıldıktan sonraki halinde de çalışma zamanı kod her çalıştığında farklılık göstermektedir. Bunun sebebi ise rastgele sayılar üretilmesi olabilir.

Duygu TEKER 1210505072