**T.C**

**KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**YAZILIM LAB DERSİ**

**ALGORİTMA ANALİZİ PROJESİ**

**HAZIRLAYANLAR**

**Selime Selin CAN – 210501005**

**Zeynep İrem AKYALÇIN - 210501008**

**DERS SORUMLUSU**

**DR. ÖĞR. ÜYESİ NUR BANU ALBAYRAK**

**7 NİSAN 2024**

# AMAÇ

Bu proje, verilen sıralı olmayan tam sayı dizisi üzerinde çalışarak, dizinin her iki elemanı arasındaki fark değerlerinden en küçüğünü ve en küçük n/2 fark değerlerini bulmayı amaçlamaktadır. Bu amaç doğrultusunda, iki farklı algoritma tasarlanmış ve her bir algoritmanın karmaşıklık analizi yapılmıştır. Projede hedeflenen, verilen şartlara uygun ve etkili algoritmaların tasarlanması ve bu algoritmaların performanslarının analiz edilmesidir. Bu analizler, hangi algoritmanın hangi koşullar altında daha verimli olduğunu belirlemeye yardımcı olacaktır.

1. **GEREKSİNİM ANALİZİ**

### 2.1 Arayüz Gereksinimleri

* Kullanıcı arayüzü gereksinimleri aşağıdaki gibi belirlenmiştir:
  + Kullanıcı, programı çalıştırdığında bir tam sayı dizisi girmelidir.
  + Program, kullanıcının seçimine bağlı olarak en küçük fark değerini bulmalıdır.
* Donanım arayüzü gereksinimleri bulunmamaktadır.

### 2.2 Fonksiyonel Gereksinimler

* Program, kullanıcının seçimine bağlı olarak iki farklı algoritma kullanarak en küçük fark değerini bulmalıdır.
* Algoritmaların karmaşıklık analizi, en iyi ve en kötü durumlar için O, θ ve Ω notasyonları kullanılarak yapılmalıdır.
* Program, belirtilen karmaşıklık analizlerini doğrulamak için rastgele üretilen girdilerle test edilmelidir.

1. **TASARIM**

### 3.1 Mimari Tasarım

* Mimari tasarım aşağıdaki şekilde yapılmıştır:
  + Proje, temelde iki ana modül içermektedir: Algoritma Modülü ve Kullanıcı Arayüzü Modülü.
  + Algoritma Modülü, en küçük fark değerlerini bulmak için toplamda dört farklı algoritma içerir.
  + Kullanıcı Arayüzü Modülü, kullanıcının programla etkileşimini sağlar ve girdi alır, sonuçları gösterir.

### 3.2 Kullanılacak Teknolojiler

* Yazılım, C programlama dili kullanılarak geliştirilecektir.
* Harici kütüphaneler kullanılmayacaktır.
* Projede başka teknolojiler kullanılmayacaktır.

### 3.3 Veritabanı Tasarımı

* Projede veritabanı kullanılmayacaktır.
* Dolayısıyla ER diyagramı eklenmemiştir.

### 3.4 Kullanıcı Arayüzü Tasarımı

* Kullanıcı arayüzü, konsol tabanlı olacaktır.
* Kullanıcıya, programın çalışması için gerekli adımlar adım adım açıklanacaktır.
* Kullanıcıya girdi almak için scanf() fonksiyonu kullanılacaktır.
* Program sonucu, konsol ekranına yazdırılacaktır.

1. **UYGULAMA**

Algoritmaların Pseudo Kodları aşağıda bulunmaktadır.

* **1.Seçenek 1.Algoritma**

1. Başla

2. Bir dizi oluştur ve maksimum 100 elemana kadar depolama kapasitesi belirle

3. Eleman sayısını tutmak için bir değişken oluştur ve 0 olarak başlat

4. Kullanıcıdan eleman sayısını al

5. Eğer eleman sayısı 2'den küçükse, "En az iki eleman girmelisiniz." mesajını yazdır ve programdan çık

6. Kullanıcıdan elemanları alarak diziyi doldur

7. Diziyi sıralamak için bir sıralama fonksiyonu çağır

7.1. Bubble sort algoritmasını kullanarak diziyi sırala

7.2. İki ardışık elemanın değerini karşılaştır ve gerekirse yer değiştir

8. En küçük farkı bulmak için bir fonksiyon çağır

8.1. Diziyi sırala

8.2. En küçük farkı bulmak için her ardışık eleman arasındaki farkları kontrol et

8.3. En küçük farkı güncelle

9. En küçük farkı ekrana yazdır

10. Bitir

**Karmaşıklık Analizleri – 1. Seçenek 1. Algoritma**

### En İyi Durum Analizi:

En iyi durumda, dizi zaten sıralı olacaktır. Bu durumda, dizi elemanları arasındaki farkları hesaplamak için sadece bir dizi taraması yapılacaktır. Dolayısıyla, bu durumda algoritmanın karmaşıklığı O(n) olacaktır.

### En Kötü Durum Analizi:

En kötü durumda, dizi tamamen ters sıralı olacaktır. Bu durumda, diziyi sıralamak için Bubble Sort algoritması kullanılacaktır. Bubble Sort'un en kötü durum karmaşıklığı O(n^2) olduğundan, diziyi sıralama işlemi O(n^2) karmaşıklığına sahiptir. Ardından, sıralı dizi üzerinde bir dizi taraması yapılacak ve ardışık elemanlar arasındaki farklar hesaplanacaktır. Bu da O(n) karmaşıklığına sahiptir. Dolayısıyla, toplam karmaşıklık en kötü durumda O(n^2) olacaktır.

### Θ ve Ω Karmaşıklığı Analizleri:

En iyi durumda ve en kötü durumda karmaşıklık aynı olduğundan, bu algoritmanın Θ(n) karmaşıklığına sahip olduğunu söyleyebiliriz. Ayrıca, her iki durumda da en azından bir dizi taraması yapılacağı için Ω(n) karmaşıklığına sahip olduğunu da söyleyebiliriz.

En iyi durumda, algoritmanın doğrusal bir karmaşıklığı olduğu görülürken, en kötü durumda karmaşıklığın karesel olduğu görülür.

* **1.Seçenek 2.Algoritma**

1. Fonksiyon EnKucukFark\_SiralamaYapmadan(dizi, n)

1.1. minimumFark değişkenini sonsuz olarak ayarla

1.2. Her i için 0'dan n-1'e kadar döngü oluştur

1.3 Her j için i+1'den n'e kadar döngü oluştur

1.4. Eğer |dizi[j] - dizi[i]| değeri minimumFark'tan küçükse

1.5. minimumFark'a |dizi[j] - dizi[i]| değerini ata

1.6. minimumFark değerini döndür

1. Fonksiyon sonu
2. Fonksiyon main()

3.1. dizi[100] ve n değişkenlerini tanımla

3.2. Kullanıcıdan n'yi al

3.3. Eğer n < 2 ise

3.4. "En az iki eleman girmelisiniz." mesajını ekrana yazdır ve programı sonlandır

3.5. Elemanları kullanıcıdan al

3.6. En küçük farkı hesapla ve ekrana yazdır

4. Fonksiyon sonu

**Karmaşıklık Analizleri – 1. Seçenek 2. Algoritma**

**En İyi Durum Analizi:**

En iyi durumda, her iki for döngüsü de tam olarak n/2 kez dönecektir. Bu durumda, algoritmanın karmaşıklığı O(n^2) olacaktır.

**En Kötü Durum Analizi:**

En kötü durumda, iç içe geçmiş iki for döngüsü de tam olarak n kez dönecektir. Bu durumda, algoritmanın karmaşıklığı O(n^2) olacaktır.

**Θ ve Ω Karmaşıklığı Analizleri:**

En iyi durumda ve en kötü durumda karmaşıklık aynı olduğundan, bu algoritmanın Θ(n^2) karmaşıklığına sahip olduğunu söyleyebiliriz. Ayrıca, her iki durumda da en azından bir dizi taraması yapılacağı için Ω(n^2) karmaşıklığına sahip olduğunu da söyleyebiliriz.

Her iki durumda da, algoritmanın kare zaman karmaşıklığına sahip olduğu görülmektedir. Bu nedenle, bu algoritma büyük veri setlerinde etkin olmayabilir ve performans sorunlarına neden olabilir.

* **2. Seçenek 1. Algoritma**

1. Başla

2. Kullanıcıdan eleman sayısını al ve n değişkenine ata

3. Eğer n < 2 ise

3.1. "En az iki eleman girmelisiniz." mesajını yazdır ve programı sonlandır

4. Elemanları tutmak için bir dizi oluştur ve bellekte yer ayır

5. Kullanıcıdan elemanları alarak diziyi doldur

6. Diziyi sıralamak için Bubble Sort algoritmasını kullan

6.1. Her eleman için, diziyi karşılaştır ve gerekirse yer değiştir

7. Dizinin ilk n/2 elemanını alarak her bir ardışık elemanın farkını hesapla

8. Farkları sırala

9. En küçük n/2 farkı döndür

10. Bitir

**Karmaşıklık Analizleri – 2. Seçenek 1. Algoritma**

**En İyi Durum Analizi:**

En iyi durumda, dizinin her iki yarısı da sıralı olacaktır. Diziyi sıralamak için Bubble Sort algoritması O(n^2) zaman karmaşıklığına sahiptir. Ardından, sıralı farklar dizisi üzerinde bir dizi taraması yapılacaktır, bu da O(n) zaman karmaşıklığına sahiptir. Dolayısıyla, toplam karmaşıklık en iyi durumda O(n^2) olacaktır.

**En Kötü Durum Analizi:**

En kötü durumda, her iki yarının sıralı olmaması durumunda, diziyi sıralamak için Bubble Sort algoritması O(n^2) zaman karmaşıklığına sahiptir. Ardından, sıralı farklar dizisi üzerinde bir dizi taraması yapılacaktır, bu da O(n) zaman karmaşıklığına sahiptir. Dolayısıyla, toplam karmaşıklık en kötü durumda O(n^2) olacaktır.

**Θ ve Ω Karmaşıklığı Analizleri:**

En iyi durumda ve en kötü durumda karmaşıklık aynı olduğundan, bu algoritmanın Θ(n^2) karmaşıklığına sahip olduğunu söyleyebiliriz. Ayrıca, her iki durumda da en azından bir dizi taraması yapılacağı için Ω(n^2) karmaşıklığına sahip olduğunu da söyleyebiliriz.

Her iki durumda da, algoritmanın kare zaman karmaşıklığına sahip olduğu görülmektedir. Bu nedenle, bu algoritma büyük veri setlerinde etkin olmayabilir ve performans sorunlarına neden olabilir.

* **2. Seçenek 2. Algoritma**

1. Başla

2. Maksimum 100 elemana kadar depolama kapasitesi olan bir dizi ve eleman sayısı değişkeni tanımla

3. Kullanıcıdan eleman sayısını al

4. Eğer eleman sayısı 2'den küçükse, "En az iki eleman girmelisiniz." mesajını yazdır ve programdan çık

5. Elemanları kullanıcıdan alarak diziyi doldur

6. Tüm farkları hesaplamak için bir dizi oluştur ve bellek ayır

7. Her bir i için 0'dan n-1'e kadar döngü oluştur

7.1. Her bir j için i+1'den n'e kadar döngü oluştur

7.2. Elemanlar arasındaki farkı hesapla ve farklar dizisine kaydet

8. Farkları sırala

9. İlk n/2 farkın ortalamasını hesapla

10. Ortalama farkı ekrana yazdır

11. Bitir

**Karmaşıklık Analizleri – 2. Seçenek 2. Algoritma**

### En İyi Durum Analizi:

En iyi durumda, her iki for döngüsü de tam olarak n/2 kez dönecektir. Farkları hesaplamak için iç içe geçmiş iki for döngüsü kullanıldığından, bu durumda algoritmanın karmaşıklığı O(n^2) olacaktır.

### En Kötü Durum Analizi:

En kötü durumda, iç içe geçmiş iki for döngüsü de tam olarak n\*(n-1)/2 kez dönecektir. Bu durumda, algoritmanın karmaşıklığı O(n^2) olacaktır.

### Θ ve Ω Karmaşıklığı Analizleri:

En iyi durumda ve en kötü durumda karmaşıklık aynı olduğundan, bu algoritmanın Θ(n^2) karmaşıklığına sahip olduğunu söyleyebiliriz. Ayrıca, her iki durumda da en azından bir dizi taraması yapılacağı için Ω(n^2) karmaşıklığına sahip olduğunu da söyleyebiliriz.

Bu analizler, algoritmanın farklı durumlarda nasıl performans göstereceğini anlamamıza yardımcı olur. Her iki durumda da, algoritmanın kare zaman karmaşıklığına sahip olduğu görülmektedir. Bu nedenle, bu algoritma büyük veri setlerinde etkin olmayabilir ve performans sorunlarına neden olabilir.

**5. ALGORİTMALARIN KARŞILAŞTIRILMASI**

Öncelikle 1. Seçenekte oluşturmamız istenen iki algoritmayı karşılaştıralım.

**Algoritma 1:**

Bu algoritma, diziyi sıralamak için bubble sort kullanır ve ardından ardışık elemanlar arasındaki en küçük farkı bulur. Bubble sort algoritması en kötü durumda O(n^2) zaman karmaşıklığına sahiptir, ancak bu durumda dizi en fazla 100 eleman içerdiği için genellikle pratikte etkili olabilir. Ardışık elemanlar arasındaki en küçük farkı bulmak için O(n) süre gereklidir, bu nedenle toplam zaman karmaşıklığı genellikle O(n^2) olur.

**Algoritma 2:**

Bu algoritma, dizideki tüm elemanlar arasındaki farkları inceleyerek en küçük farkı bulur. İki iç içe döngü kullanarak bu işlemi gerçekleştirir. Bu durumda, karmaşıklık O(n^2) olur.

**Karşılaştırma:**

* Karmaşıklık: Her iki algoritmanın da karmaşıklığı O(n^2) olarak belirlenmiştir. Ancak Algoritma 1'in pratikte daha etkili olma olasılığı daha yüksektir, çünkü Algoritma 1, bubble sort algoritmasını kullanarak sıralama işlemini gerçekleştirirken, Algoritma 2 tüm elemanlar arasındaki farkları doğrudan hesaplar.
* Hız: Algoritma 1, sıralama işlemi için bubble sort kullanırken, Algoritma 2 tüm elemanlar arasındaki farkları tek tek hesaplar. Bu nedenle, pratikte Algoritma 1'in daha hızlı olma olasılığı daha yüksektir.
* Verimlilik: Her iki algoritma da farkları doğru şekilde bulur, ancak Algoritma 1 genellikle daha az işlem yaparak sonuca ulaşır, bu da daha verimli olduğu anlamına gelir.

Sonuç olarak, verilen durumda Algoritma 1, daha hızlı ve daha verimli bir seçenek gibi görünmektedir.

2. Seçenekteki algoritmaları karşılaştıralım.

**Algoritma 1:**

Bu algoritma, diziyi sıralayarak başlar ve ardından dizinin ilk n/2 elemanını alır. Bu işlem O(n log n) zaman alır. Ardından, bu elemanlar arasındaki farkları hesaplar ve bunları sıralar. Farklar sıralandığında, en küçük n/2 farkı bulmak için O(n) süre gerekir. Bu nedenle, algoritmanın toplam karmaşıklığı O(n log n) + O(n) = O(n log n) olur.

**Algoritma 2:**

Bu algoritma, iki iç içe döngü kullanarak tüm elemanlar arasındaki farkları hesaplar. Bu işlem O(n^2) zaman alır. Ardından, tüm farkları sıralamak için qsort() fonksiyonu kullanılır, bu da O(n^2 log n^2) = O(n^2 log n) zaman alır. Son olarak, sıralı farklar dizisinin ilk n/2 elemanının ortalaması hesaplanır, bu da O(n) süre alır. Bu nedenle, algoritmanın toplam karmaşıklığı O(n^2 log n) olur.

**Karşılaştırma:**

* Karmaşıklık: Algoritma 1, O(n log n) karmaşıklığına sahiptir, ancak Algoritma 2 O(n^2 log n) karmaşıklığına sahiptir. Bu nedenle, Algoritma 1 daha verimlidir.
* Hız: Algoritma 1, genellikle Algoritma 2'ye göre daha hızlıdır, çünkü sıralama işlemi O(n log n) zaman alırken, Algoritma 2'nin sıralama için kullanılan qsort işlevi O(n^2 log n) zaman alır.
* Verimlilik: Algoritma 1'in daha verimli olduğu söylenebilir, çünkü daha az işlem yaparak sonuca ulaşır. Ayrıca, Algoritma 1, ek bellek kullanımı açısından da daha iyi performans gösterebilir, çünkü Algoritma 2'nin tüm farklar için ayrı bir bellek alanı ayırması gerekmektedir.

Sonuç olarak, verilen durumda Algoritma 1, daha hızlı ve daha verimli bir seçenek gibi görünmektedir.

**6.TEST VE DOĞRULAMA**

* Herhangi bir ayrı test uygulaması geliştirilmedi, ancak algoritmalar farklı girdilerle manuel olarak test edildi.
* Algoritmaların her biri için farklı durumlar ve sınırlamalar göz önünde bulundurularak testler yapıldı.
* Her bir algoritmanın doğru çalışmasını sağlamak için birden fazla farklı girdi kullanılarak testler tekrarlandı.

**7.KAYNAKÇA**

* <https://www.geeksforgeeks.org/bubble-sort/>
* <https://www.geeksforgeeks.org/recursive-bubble-sort/?ref=lbp>
* <https://www.programiz.com/dsa/bubble-sort>
* <https://edu.gcfglobal.org/en/computer-science/algorithms/1/>
* <https://birhankarahasan.com/algoritma-analizi-nedir-zaman-karmasikligi-big-o-gosterimi>
* <http://cagataykiziltan.net/programin-calisma-hizi-ve-algoritma-verimliligi/zaman-karmasikligi-ve-buyuk-o-notasyonu-time-complexity-and-big-o-notation/>
* <http://cagataykiziltan.net/algoritmalar/1-siralama-algoritmalari/1-araya-sokma-siralamasi/>

Formun Üstü

Formun Altı