SIRALAMA ALGORİTMALARI GÖRSELLEŞTİRİCİSİ

Zeynep Gül Aslan , İsmail Kocatürk

Bilişim Sistemleri Mühendisliği Teknoloji Fakültesi/Kocaeli Üniversitesi
191307049@kocaeli.edu.tr ,201307045@kocaeli.edu.tr

***Özet*—Bu proje kapsamında, kullanıcının sıralama algoritmalarını ve grafik türlerini kullanarak bir listenin sıralanmasını ve sıralama işleminin adımlarını görsel bir şekilde görmesini sağlamaktır. Proje, kullanıcı arayüzü (GUI) hazırlanarak, masaüstü uygulaması olarak geliştirilmiştir.**

Anahtar kelimeler—GUI, Sıralama Algoritmaları

***Abstract*— Within the scope of this project, the aim is to allow the user to visualize the sorting of a list using sorting algorithms and different types of graphs. The project is developed as a desktop application with a Graphical User Interface (GUI).**

Keywords— GUI, Sorting Algorithms

1. GIRIŞ

Sol panelde, kullanıcının sıralanacak listeyi manuel olarak girebilme veya boyutunu belirleyerek rasgele bir liste oluşturma seçeneği olmalıdır. Ayrıca, kullanıcının animasyon hızını ölçeklendirebilmesi mümkün olmalıdır. Sıralama algoritmaları olarak Seçme Sıralaması, Kabarcık Sıralaması, Ekleme Sıralaması, Birleştirme Sıralaması ve Hızlı Sıralama seçenekleri sunulmalıdır. Grafik tipleri olarak Dağılım, Sütun ve Kök grafikleri kullanılmalıdır. Ayrıca, oluştur, başla, dur ve sıfırla butonları bulunmalıdır. Oluştur butonuna basıldığında verilen liste değerleri ve grafik türüne göre arayüz oluşturulmalı, başla butonuna basıldığında animasyon belirtilen isterlere uygun olarak başlamalı, dur butonuna basıldığında animasyon durdurulmalı ve tekrar basıldığında animasyona devam etme seçeneği sunulmalı, sıfırla butonuna basıldığında liste temizlenmelidir.

Ana panelde, sol panelde yapılan seçimlere göre görsel bir arayüz sunulmalıdır. Seçilen algoritmanın sıralama işlemi gerçekleştirilirken, karşılaştırma adımlarını anlaşılır hale getirmek için renk kodları kullanılmalıdır. Karşılaştırılan değerler aynı renk kodunu paylaşmalı ve sıralanmış ve sıralanmamış değerler farklı renk kodlarıyla tanımlanmalıdır. Ayrıca, her adımda yapılan karşılaştırma sayısı arka planda tutulmalı ve animasyon üzerinde yapılan değişikliklere paralel olarak bu sayı artırılmalıdır. Belirli periyotlarla karşılaştırma sayısı güncellenerek ana panelde gösterilmelidir. Sıralama işlemi tamamlandığında, karşılaştırma sayısı ve algoritmanın karmaşıklık analizi sonucu ekrana yazdırılmalıdır.

1.1 Arayüzün Tasarlanması

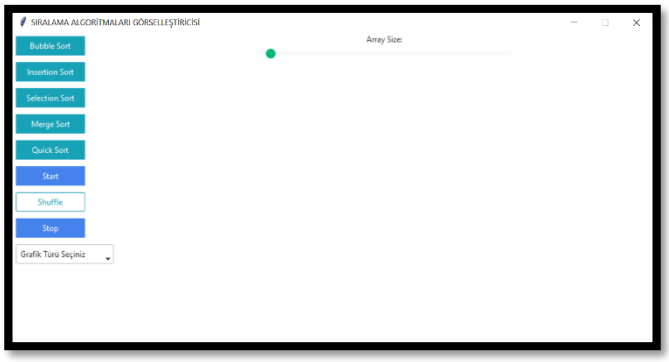
Bu projede, “Tkinter” kütüphanesini kullanarak bir GUI (Grafiksel Kullanıcı Arayüzü) tasarımını oluşturuldu. GUI, sıralama algoritmalarını görselleştirmek için gerekli bileşenleri içeriyor.

Tasarımın sol panelinde aşağıdaki bileşenler bulunuyor:

- **`ttk.Frame`**: Sol paneli ayırmak için kullanılıyor.
- **`ttk.Combobox`**: Grafik türünü seçmek için biraçılır menü.
- **`ttk.Scale`**: Animasyon hızını ve eleman sayısını ayarlamak için birkaydırıcı.
- **`ttk.Button`**: "Shuffle" (Karıştır), "Start" (Başlat), "Stop" (Dur) ve Sıralama Algoritmalarının butonları.

Sıralama algoritmalarını temsil eden beş düğme (`ttk.Button`) bulunuyor: "Selection Sort", "Bubble Sort", "Insertion Sort", "Merge Sort" ve "Quick Sort".

Ana panelde ise bir `Canvas` yer alıyor. Bu bileşen, seçilen grafik türüne bağlı olarak sıralama işlemini görsel olarak göstermek için kullanılacak bir grafik görüntüleyici sağlıyor.



Şekil-1 Kullanıcının fotoğraf seçtiği yer.

1.2 Sıralama Algoritmaları

Seçme Sıralaması (Selection Sort): Bu sıralama algoritması, bir listeyi sıralamak için kullanılan basit bir yöntemdir. Listede en küçük elemanı bulup listenin başına yerleştirir, ardından kalan elemanlar arasından en küçüğü bulup sıralanan kısmın sonuna ekler. Bu işlem, listenin tamamı sıralanana kadar tekrarlanır. Seçme sıralaması, özellikle küçük boyutlu listelerde etkili bir şekilde çalışır, ancak büyük listelerde performansı düşebilir.

```
elif self.sorts['selection'] is True:
    for i in range(len(self.data) - 1):
        min_index = i

        for j in range(i + 1, len(self.data)):
            self.display(self.N, self.data,
                         ['yellow' if a == min_index or a == i else 'green' if a <= i else 'dodgerblue' for a in
                          range(self.N)])
            time.sleep(self.speed)
            if self.data[min_index] > self.data[j]:
                self.display(self.N, self.data,
                             ['red' if a == min_index or a == j else 'green' if a <= i else 'dodgerblue' for a
                              in range(self.N)])
                time.sleep(self.speed)
                min_index = j
        if min_index != i:
            self.data[i], self.data[min_index] = self.data[min_index], self.data[i]
            self.display(self.N, self.data,
                         ['lime' if a == min_index or a == i else 'green' if a <= i else 'dodgerblue' for a in
                          range(self.N)])
            time.sleep(self.speed)
    self.display(self.N, self.data, self.__sorted_array)
```

Şekil-2 Selection Sort butonuna basıldığında çalışan kod.

Kabarcık Sıralaması (Bubble Sort): Kabarcık sıralaması, listeyi sıralamak için karşılaştırmalar ve yer değiştirmeler yaparak çalışan bir sıralama algoritmasıdır. Karşılaştırma sonucunda yanlış sıralanmış bir çift eleman bulunduğunda, bu elemanlar yer değiştirilir. Bu işlem liste üzerinde tekrarlanır ve en büyük elemanlar yavaş yavaş listenin sonuna doğru yerleşir. Bu süreç, liste tamamen sıralanana kadar devam eder. Kabarcık sıralaması, basit bir yapıya sahip olmasına rağmen, büyük listelerde etkili bir şekilde çalışmaz ve performans açısından diğer sıralama algoritmalarına göre daha yavaş olabilir.

```
def __bubbleSort(self):
    pass

def stop(self):
    self.sorting = False

def start(self):
    if self.sorts['bubble'] is True:
        for i in range(self.N - 1):
            for j in range(self.N - 1 - i):

                self.display(self.N, self.data,
                             ['purple' if a == j or a == j + 1 else 'green' if a > self.N - 1 - i else 'dodgerblue'
                              for a in range(self.N)])
                time.sleep(self.speed)
                if self.data[j] > self.data[j + 1]:
                    self.display(self.N, self.data,
                                 ['red' if a == j or a == j + 1 else 'green' if a > self.N - 1 - i else 'dodgerblue'
                                  for a in range(self.N)])
                    time.sleep(self.speed)
                    self.data[j], self.data[j + 1] = self.data[j + 1], self.data[j]
                    self.display(self.N, self.data, [
                        'lime' if a == j or a == j + 1 else 'green' if a > self.N - 1 - i else 'dodgerblue' for a in
                        range(self.N)])
                    time.sleep(self.speed)
        self.display(self.N, self.data, self.__sorted_array)
```

Şekil-3 Bubble Sort butonuna basıldığında çalışan kod.

Ekleme Sıralaması (Insertion Sort): Ekleme sıralaması, bir listenin sıralanması için kullanılan bir algoritmadır. Listenin sıralanmış kısmı ve sıralanmamış kısmı olmak üzere iki bölüm üzerinde çalışır. Sıralanmamış kısmın elemanları, sıralanmış kısma uygun konuma yerleştirilir. Bu işlem, listenin tamamı sıralanana kadar tekrarlanır. Ekleme sıralaması, küçük boyutlu listelerde etkili bir şekilde çalışır ve özellikle neredeyse sıralı olan listelerde yüksek performans gösterebilir.

```
elif self.sorts['insertion'] is True:
    for j in range(1, len(self.data)):
        key = self.data[j]
        i = j - 1
        self.display(self.N, self.data,
                     ['purple' if a == i or a == i + 1 else 'green' if a <= j else 'dodgerblue' for a in
                      range(self.N)])
        time.sleep(self.speed)
        while i >= 0 and self.data[i] > key:
            self.data[i + 1] = self.data[i]
            self.display(self.N, self.data,
                         ['yellow' if a == i else 'green' if a <= j else 'dodgerblue' for a in range(self.N)])
            time.sleep(self.speed)
            i -= 1
        self.data[i + 1] = key
    self.display(self.N, self.data, self.__sorted_array)
```

Şekil-4 Insertion Sort butonuna basıldığında çalışan kod.

Birleştirme Sıralaması (Merge Sort): Birleştirme sıralaması, "böl ve fethet" prensibiyle çalışan bir sıralama algoritmasıdır. Listenin elemanları sürekli olarak ikiye bölünür, ardından her bir alt liste ayrı ayrı sıralanır. Daha sonra, sıralanmış alt listeler birleştirilir ve nihai olarak tam bir sıralı liste elde edilir. Birleştirme sıralaması, büyük listelerde yüksek performans gösteren verimli bir sıralama algoritmasıdır. Ancak, bellek kullanımı nedeniyle diğer sıralama algoritmalarına göre daha fazla kaynak gerektirebilir.

```
def mergesort(self, a, front, last):
    if front < last:
        mid = (front + last) // 2

        self.mergesort(a, front, mid)
        self.mergesort(a, mid + 1, last)

        self.display(self.N, self.data, self.__default_colours)

        rj = mid + 1
        if a[mid] <= a[mid + 1]:
            return

        while front <= mid and rj <= last:
            self.display(self.N, self.data,
                         ['yellow' if x == front or x == rj else 'dodgerblue' for x in range(self.N)])
            time.sleep(self.speed)
            if a[front] <= a[rj]:
                self.display(self.N, self.data,
                             ['lime' if x == front or x == rj else 'dodgerblue' for x in range(self.N)])
                time.sleep(self.speed)
                front += 1
            else:
                self.display(self.N, self.data,
                             ['red' if x == front or x == rj else 'dodgerblue' for x in range(self.N)])
                time.sleep(self.speed)
                temp = a[rj]
                i = rj
                while i != front:
                    a[i] = a[i - 1]
                    i -= 1
                a[front] = temp
                self.display(self.N, self.data,
                             ['lime' if x == front or x == rj else 'dodgerblue' for x in range(self.N)])
                time.sleep(self.speed)
```

Şekil-5 Merge Sort butonuna basıldığında çalışan kod.

Hızlı Sıralama (Quick Sort): Hızlı sıralama, "böl ve fethet" prensibine dayanan bir sıralama algoritmasıdır. İlk adımda, bir pivot eleman seçilir ve listenin diğer elemanları pivot elemanına göre küçük veya büyük olarak bölünür. Bu işlem, liste tamamen sıralanana kadar tekrarlanır. Her bölme adımında pivot elemanın doğru konuma yerleştirildiği garanti edilir. Hızlı sıralama, ortalama durumlarda hızlı bir şekilde çalışır ve büyük listelerde etkili performans sağlar. Ancak, en kötü durumda (örneğin, listenin zaten sıralı olduğu durumda) performansı düşebilir ve diğer sıralama algoritmalarına göre daha fazla hafıza gerektirebilir.

```
def partition(self, a, i, j):
    l = i

    pivot = a[i]
    piv_index = i

    while i < j:
        while i < len(a) and a[i] <= pivot:
            i += 1
            self.display(self.N, self.data,
                         ['purple' if x == piv_index else 'yellow' if x == i else "dodgerblue" for x in
                          range(self.N)])
            time.sleep(self.speed)
        while a[j] > pivot:
            j -= 1
        if i < j:
            self.display(self.N, self.data, ['red' if x == i or x == j else "dodgerblue" for x in range(self.N)])
            time.sleep(self.speed)
            a[i], a[j] = a[j], a[i]
            self.display(self.N, self.data, ['lime' if x == i or x == j else "dodgerblue" for x in range(self.N)])
            time.sleep(self.speed)
    a[j], a[l] = a[l], a[j]
    return j

def quicksort(self, a, i, j):
    if i < j:
        x = self.partition(a, i, j)
        self.quicksort(a, i, x - 1)
        self.quicksort(a, x + 1, j)
```

Şekil-6 Quick Sort butonuna basıldığında çalışan kod.

2. BENZER ÇALIŞMALAR VE LITARATÜR TARAMASI

Sıralama algoritmalarının görselleştirilmesiyle ilgili literatürde çeşitli çalışmalar bulunmaktadır. Bu çalışmalar genellikle algoritmaların adımlarını, işleyişini ve performansını daha iyi anlamak için görsel araçlar ve animasyonlar kullanmayı hedefler. İşte bu alanda yapılan bazı çalışma örnekleri:

- Görsel Animasyonlar: Sıralama algoritmalarının adımlarını görsel olarak anlatan animasyonlar, kullanıcıların algoritmaların çalışma prensiplerini daha iyi anlamalarına yardımcı olur. Bu animasyonlar genellikle renkli grafikler, geçiş efektleri ve ilerleme göstergeleri gibi görsel öğeleri kullanır.

- Etkileşimli Simülasyonlar: Kullanıcılara sıralama algoritmalarını kendi deneyimleyebilecekleri etkileşimli simülasyonlar sunulur. Bu simülasyonlar, kullanıcının adımları kontrol etmesine ve farklı parametreleri değiştirmesine olanak tanır. Kullanıcılar, algoritmanın çalışmasını gözlemleyerek ve deneyerek algoritmanın performansını anlama imkânı bulurlar.

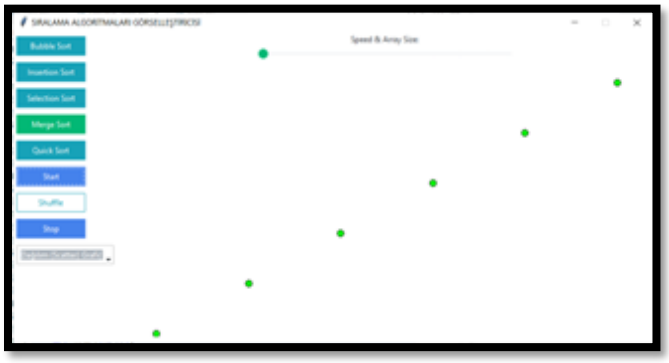
- Web Tabanlı Uygulamalar: Sıralama algoritmalarının görselleştirilmesi için web tabanlı uygulamalar da yaygın olarak kullanılır. Bu uygulamalar, kullanıcıların web tarayıcıları üzerinden sıralama algoritmalarını seçip çalıştırabilmelerini sağlar. Kullanıcılar, gerçek zamanlı olarak algoritmanın adımlarını ve sonuçlarını görüntüleyebilir.

- Eğitim Araçları: Sıralama algoritmalarının görselleştirilmesi, eğitim amaçlı araçların geliştirilmesinde de kullanılır. Bu tür araçlar, öğrencilere algoritmaların nasıl çalıştığını daha iyi anlamaları için görsel öğeler ve interaktif özellikler sağlar. Öğrenciler, adımları takip ederek algoritmaları anlayabilir ve daha iyi bir şekilde öğrenebilirler.

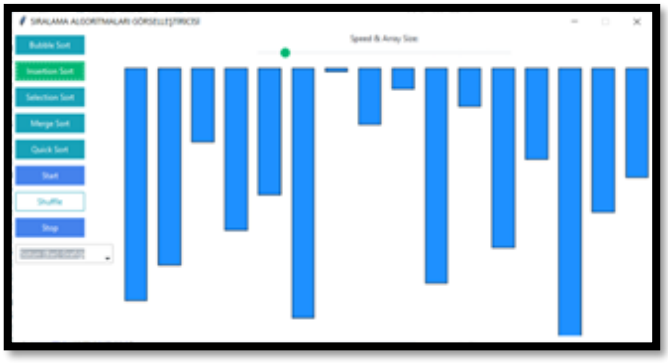
3. PROJEYİ GELİŞTİRİRKEN KULLANDIĞIM YAZILIMSAL MİMARİ YÖNTEM VE TEKNİKLER

Projeyi geliştirirken Geleneksel Süreç Modellerinden Arttırımsal (Augmentative) Model'i kullandık.

Arttırımsal Modelde üretilen ve uygulamaya alınan her ürün sürümü birbirini içeren ve giderek artan sayıda işlev içerecek biçimde geliştirilmektedir. Öncelikle ürüne ilişkin çekirdek bir kısım geliştirilerek uygulamaya alınmakta, ardından yeni işlevsellikler eklenerek yeni sürümler elde edilmektedir. Tıpkı bu ödevi yaptığım hafta boyunca her gün gelişimini izlemem gibi.



Şekil-7 Kök grafiği görünümü.



Şekil-8 Sütün grafiği görünümü.

4. KARŞILAŞTIĞIM ZORLUKLAR

- GUI tasarımında panel düzenini kurmakta zorlandım.
- PyQt kütüphanesini kullanmakta zorlandım.
- Grafikleri görüntülemekte zorlandım.

KAYNAKÇA

[1] Github
[2] ChatGPT
[3] Her Yönüyle Python Fırat Özgül