# Algoritma ve Veri Yapısı Değerlendirmesi

Bu projede, klasik sabit diziler yerine dinamik veri yapıları tercih edilmiştir. Bu sayede algoritmalar daha esnek, yeniden kullanılabilir ve gerçek dünya senaryolarına uygun hale getirilmiştir. Aşağıda kullanılan veri yapılarının katkıları özetlenmiştir:

## Bağlı Liste (CustomLinkedList)

* Kullanım: Kullanıcının eklediği sayıları sırasıyla saklamak için kullanılmıştır.
* Avantajı: Sabit uzunlukta olmayan, dinamik olarak büyüyebilen bir yapıdır.
* Projeye Katkısı: Son ekleneni sil gibi işlemler RemoveLast() sayesinde doğrudan ve verimli şekilde yapılmıştır.

## Yığın (Stack)

* Kullanım: Silinen düğümler, Stack veri yapısında saklanmıştır.
* Avantajı: LIFO prensibi sayesinde son silinen düğüm en üstte tutulur.
* Projeye Katkısı: Silinen düğümleri listeleyerek geçmişi görme ve geri alma yeteneği sağlar.

## Binary Search Tree (BST)

* Kullanım: Sayıların saklandığı ve üzerinde gezinme, ekleme, silme gibi işlemlerin yapıldığı temel ağaç yapısıdır.
* Avantajı: Ortalama O(log n) karmaşıklıkla arama, ekleme ve silme işlemlerini destekler.
* Projeye Katkısı: Görsel olarak ağaç yapısı üzerinde algoritmaların işleyişini gösterir.

## Kuyruk (Queue)

* Kullanım: Level-order traversal sırasında BFS algoritmasının uygulanmasında kullanılmıştır.
* Avantajı: FIFO mantığı ile ağaç seviyelerini sırayla işler.
* Projeye Katkısı: Düğümler seviyeler halinde vurgulanarak öğretici yapı sunar.

## Özet Karşılaştırma Tablosu

|  |  |  |
| --- | --- | --- |
| Veri Yapısı | Kullanım Amacı | Diziye Göre Avantajı |
| LinkedList | Eklenenleri sıralı tutmak | Yeniden boyutlandırmaya gerek kalmaz, esneklik |
| Stack | Silinenleri saklamak | Geri alma gibi işlemleri kolaylaştırır |
| Queue | Level-order gezinme (BFS) | Düzgün sıra işleme, seviyeli gezinme sağlar |
| BST | Sayıları verimli şekilde saklamak | Arama, ekleme ve silme işlemleri daha hızlı |