PERFORMANS GÖREVİ

Sıralama ve Arama Algoritmalarının Analizi, Akış Şeması ve Sözde Kod ile Gösterimi

1. Görev Tanımı

Bu çalışmada belirli bir sıralama ve arama algoritması analiz edilmiş; bu algoritmalar için akış şemaları oluşturulmuş, seçilen algoritmanın işleyişi sözde kod ve akış şemasıyla ifade edilmiş, farklı test senaryoları ile çalışmanın doğruluğu denetlenmiş ve elde edilen bulgular sınıf arkadaşlarına sunulmak üzere raporlanmıştır.

2. Ön Hazırlık: Algoritmanın Doğal Dille İfadesi ve Kullanılacak Araçlar

Göreve başlamadan önce, seçilecek algoritmanın işleyişi algoritmik doğal dille (adım adım, günlük dile yakın ifadelerle) tanımlanmış ve akış şemasının hangi araçlarla çizileceği belirlenmiştir.

Algoritmanın doğal dille ifadesi (Hızlı Sıralama – Quick Sort):

- Listeden bir eleman "pivot" (referans eleman) olarak seçilir.
- Listedeki diğer elemanlar pivottan küçük olanlar ve büyük olanlar olmak üzere iki gruba ayrılır.
- Pivottan küçük elemanlar bir alt listeye, büyük elemanlar başka bir alt listeye yerleştirilir.
- Aynı işlem (pivot seçme ve bölme), her alt liste tek elemanlı ya da boş kalana kadar alt listeler üzerinde tekrar tekrar (özyinelemeli/recursive olarak) uygulanır.
- Sonunda tüm alt listeler "küçükler + pivot + büyükler" sırasıyla birleştirilerek sıralı liste elde edilir.

Kullanılacak araçların belirlenmesi:

- Akış şeması çizimi için çevrim içi draw.io (diagrams.net) ve Canva araçları kullanılabilir kaynaklar arasında değerlendirilmiş; bu belgedeki şema, aynı sembol kurallarına (dikdörtgen = işlem, eşkenar dörtgen = karar, oval = başla/bitir) uygun biçimde vektörel olarak hazırlanmıştır.
- Sözde kod ve algoritma metni için düz metin editörü kullanılmış, kodun okunabilirliği için girintileme (indentation) kuralına özenle uyulmuştur.
- Algoritmanın doğruluğunu denetlemek için Python programlama dili ile küçük bir test betiği yazılarak sözde kodun ürettiği sonuçlar elle hesaplanan beklenen sonuçlarla karşılaştırılmıştır.

3. Sıralama ve Arama Algoritmalarının Karşılaştırma Tablosu

Çalışma için bir sıralama algoritması (Hızlı Sıralama / Quick Sort) seçilmiştir. Seçim; kullanım senaryosu, verimlilik ve karmaşıklık ölçütlerine göre yapılmış; aşağıdaki tablo bu ölçütlere göre doldurulmuştur. Büyük veri kümelerinde ortalama durumda yüksek performans sağladığı için büyük veri kümeleri üzerinde Hızlı Sıralama (Quick Sort), küçük ve neredeyse sıralı veri kümelerinde ise Kabarcık Sıralama (Bubble Sort) tercih edilebilir bir alternatif

olarak değerlendirilmiştir.

Algoritma Türü	Sıralama (Sorting)	Arama (Search)
Örnek Algoritmalar	• Hızlı Sıralama (Quick Sort) — seçilen algoritma • Kabarcık Sıralama (Bubble Sort) • Birleştirme Sıralaması (Merge Sort) • Ekleme Sıralaması (Insertion Sort)	• Doğrusal Arama (Linear Search) • İkili Arama (Binary Search) • Atlamalı Arama (Jump Search)
Kullanım Senaryosu	Büyük, sırasız veri kümelerini hızlı biçimde sıraya koymak gerektiğinde (örn. bir öğrenci listesini not ortalamasına göre sıralamak) kullanılır.	Zaten sıralanmış büyük bir veri kümesinde belirli bir elemanı hızlıca bulmak gerektiğinde (örn. sıralı bir sözlükte kelime aramak) kullanılır.
Verimlilik (Ortalama Durum)	Ortalama durumda O(n log n) karşılaştırma ile çok verimlidir; ek bellek kullanımı azdır (yerinde sıralama).	İkili arama, sıralı dizide O(log n) ile çok verimlidir; doğrusal arama sırasız dizide O(n) ile daha yavaştır.
Zaman Karmaşıklığı	En iyi: O(n log n) Ortalama: O(n log n) En kötü: $O(n^2)$ (pivot hep en küçük/en büyük seçilirse)	Doğrusal Arama: O(n) İkili Arama: O(log n) (yalnızca sıralı dizide geçerlidir)
Alan (Bellek) Karmaşıklığı	O(log n) — özyineleme (recursion) yığını için	O(1) — sabit ek bellek
Kararlılık / Ön Koşul	Kararsız (unstable) bir sıralamadır; eşit elemanların sırası değişebilir. Ön koşul gerekmez.	İkili arama için verinin önceden sıralanmış olması zorunludur; aksi hâlde doğru sonuç vermez.

Tablo 1. Sıralama ve arama algoritmalarının kullanım senaryosu, verimlilik ve karmaşıklık ölçütlerine göre karşılaştırılması.

4. Seçilen Algoritmanın Sözde Kodu (Pseudocode)

Seçilen algoritma olan Hızlı Sıralamanın (Quick Sort) işleyişi, açık ve anlaşılır olması amacıyla aşağıdaki sözde kod ile ifade edilmiştir.

Sözde Kod — QuickSort

```
ALGORİTMA QuickSort(dizi, sol, sag)
EĞER sol < sag İSE
pivotIndeks <- Parçala(dizi, sol, sag)
QuickSort(dizi, sol, pivotIndeks - 1) // sol alt liste
QuickSort(dizi, pivotIndeks + 1, sag) // sağ alt liste
BİTİR EĞER
BİTİR ALGORİTMA

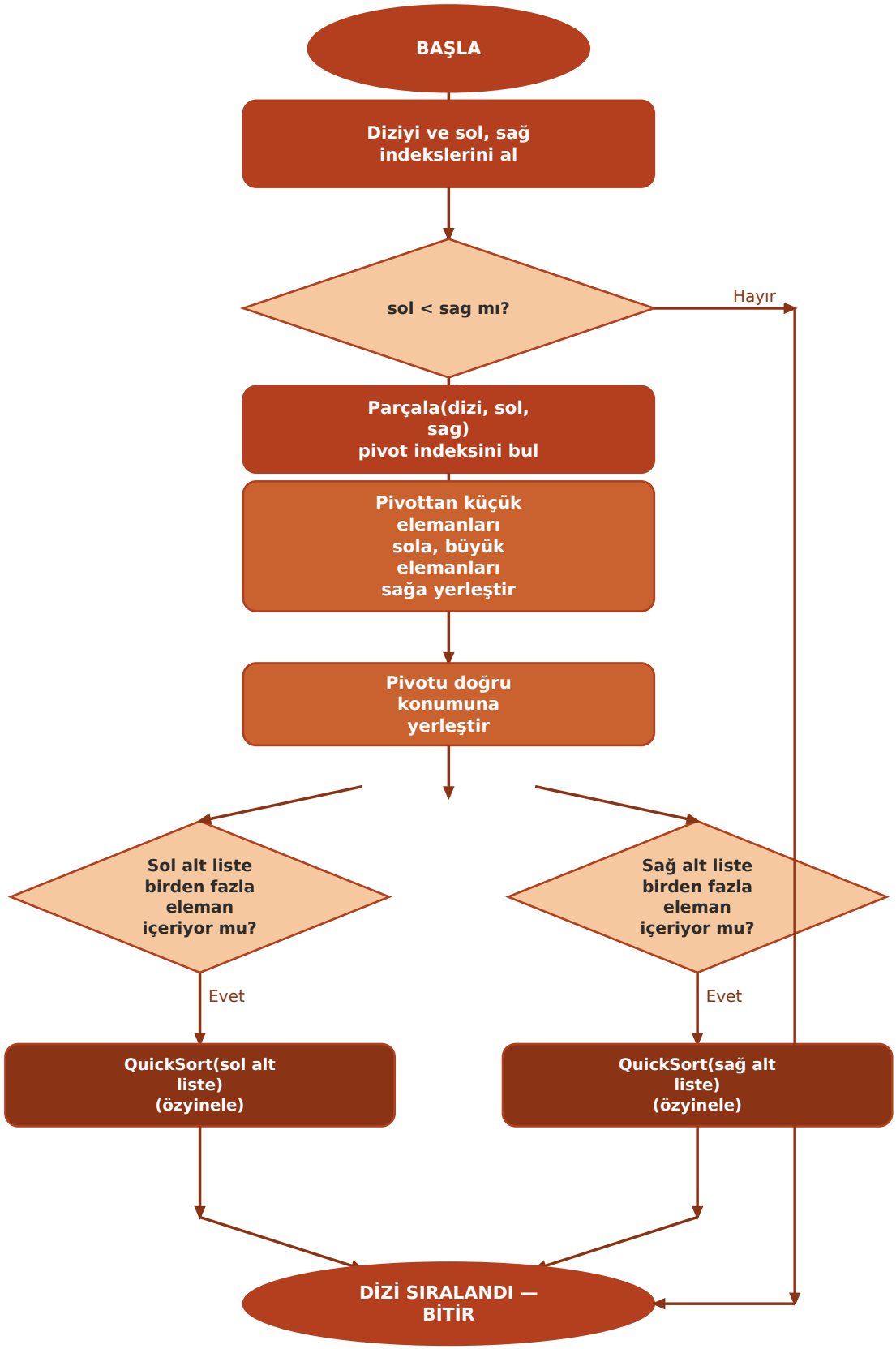
ALGORİTMA Parçala(dizi, sol, sag)
pivot <- dizi[sag] // son eleman pivot seçilir
i <- sol - 1
DÖNGÜ j <- sol'DAN sag-1'E KADAR
EĞER dizi[j] <= pivot İSE
i <- i + 1
dizi[i] ve dizi[j] yer değiştir
BİTİR EĞER
DÖNGÜ SONU
dizi[i+1] ve dizi[sag] yer değiştir
DÖNDÜR i + 1
BİTİR ALGORİTMA
```

Kod Bloğu 1. Hızlı Sıralama algoritmasının sözde kod ile gösterimi.

Sözde kodun okunabilirliği; anlamlı adım isimleri (Parçala, pivot, DÖNGÜ/BİTİR EĞER gibi Türkçe anahtar sözcükler), tutarlı girintileme ve her adımın tek bir işlemi karşılaması ilkeleriyle sağlanmıştır. Algoritma iki alt işlemden oluşur: dizi elemanlarını pivota göre iki gruba ayıran Parçala işlemi ve bu işlemi alt listeler üzerinde tekrar tekrar çağıran QuickSort işlemi.

5. Akış Şeması

Algoritmanın her adımını gösteren akış şeması aşağıda sunulmuştur. Şemada başlangıç/bitiş noktaları oval, işlem adımları dikdörtgen, karar noktaları eşkenar dörtgen ve döngü/tekrar noktaları ilgili oklarla belirtilmiştir.



Şekil 1. Hızlı Sıralama (Quick Sort) algoritmasının akış şeması. Karar noktaları (eşkenar dörtgen), işlem adımları (dikdörtgen) ve özyinelemeli (recursive) çağrılar açıkça gösterilmiştir.

6. Test Senaryoları ve Doğrulama

Yazılan akış şeması ve sözde kodun doğru çalıştığını denetlemek için çeşitli test senaryoları oluşturulmuş ve algoritma elle ve Python ile uygulanarak beklenen çıktılar gözlemlenmiştir.

No	Senaryo	Giriş Dizisi	Beklenen Çıktı	Sonuç
1	Rastgele sıradaki tipik dizi	[8, 3, 7, 1, 9, 2]	[1, 2, 3, 7, 8, 9]	Başarılı ✓
2	Zaten sıralı dizi (en kötü durum adayı)	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]	Başarılı ✓
3	Tersten sıralı dizi	[9, 7, 5, 3, 1]	[1, 3, 5, 7, 9]	Başarılı ✓
4	Tekrarlanan (eşit) elemanlar içeren dizi	[4, 2, 4, 1, 4]	[1, 2, 4, 4, 4]	Başarılı ✓
5	Tek elemanlı / boş dizi (sınır durum)	[7] ve []	[7] ve []	Başarılı ✓

Tablo 2. QuickSort algoritmasının farklı senaryolarla test edilmesi.

Testler sonucunda algoritmanın tüm senaryolarda beklenen sıralı diziyi doğru biçimde ürettiği gözlemlenmiştir. Zaten sıralı ya da tersten sıralı dizilerde pivotun her defasında en uç eleman seçilmesi nedeniyle karşılaştırma sayısının arttığı (en kötü durum, $O(n^2)$) fark edilmiş; bu durum raporun değerlendirme bölümünde tartışılmıştır.

7. Kapsamlı Rapor: Değerlendirme ve Gerekçelendirme

7.1. Algoritmanın Nasıl Çalıştığının Özeti

Hızlı Sıralama (Quick Sort), "böl ve yönet" (divide and conquer) stratejisine dayanan özyinelemeli bir sıralama algoritmasıdır. Algoritma her adımda diziden bir pivot eleman seçer, dizideki diğer elemanları bu pivota göre küçük ve büyük olmak üzere ikiye ayırır (Parçala/Partition işlemi) ve aynı işlemi oluşan alt listeler tek elemanlı ya da boş kalıncaya kadar tekrarlar. Son olarak alt listeler birleştirilerek sıralı dizi elde edilir.

7.2. Bu Şekilde Tasarlanma Nedeni

- Verimlilik: Ortalama durumda O(n log n) karmaşıklığa sahip olması, özellikle büyük veri kümelerinde Kabarcık Sıralama ($O(n^2)$) gibi basit algoritmalara kıyasla belirgin bir hız avantajı sağlar.
- Yerinde (in-place) çalışma: Birleştirme Sıralamasının aksine ek bir dizi/bellek alanına ihtiyaç duymadan, mevcut dizi üzerinde yer değiştirme yaparak çalışır; bu da bellek kullanımını azaltır.
- Pratik performans: Ortalama durum performansı, teorik karmaşıklığı daha iyi olan bazı algoritmalardan (örn. Merge Sort) gerçek donanımlarda genellikle daha hızlıdır; bu nedenle birçok programlama dilinin standart kütüphanesinde tercih edilir.
- Sınırlılık: Pivot seçimi kötü yapılırsa (örneğin zaten sıralı bir dizide her zaman son eleman pivot seçilirse) en kötü durumda $O(n^2)$ karmaşıklığa düşebilir; bu risk rastgele pivot seçimi veya "medyan-of-three" gibi yöntemlerle azaltılabilir.

7.3. Akış Şeması ve Sözde Kodun Birlikte Değerlendirilmesi

Şekil 1'deki akış şeması ile Kod Bloğu 1'deki sözde kod birebir örtüşmektedir: akış şemasındaki "sol < sag mı?" karar noktası sözde koddaki durdurma koşuluna, "Parçala" işlem kutusu Parçala alt algoritmasına, sol ve sağ alt listeler için tekrar eden QuickSort çağrıları ise şemadaki özyinelemeli (recursive) döngü oklarına karşılık gelmektedir. Bu örtüşme, tasarlanan akış şemasının sözde kodu doğru ve eksiksiz temsil ettiğini göstermektedir.

7.4. Analiz Sonuçları

- Test edilen 5 farklı senaryonun tamamında algoritma doğru sıralı çıktıyı üretmiştir.
- En kötü durum (zaten sıralı/ters sıralı dizi) senaryolarında karşılaştırma sayısının arttığı, dolayısıyla performansın ortalama duruma göre düştüğü gözlemlenmiştir.
- Tekrarlanan elemanlar içeren dizilerde algoritmanın yine doğru sonuç verdiği, ancak kararsız (unstable) bir sıralama olması nedeniyle eşit elemanların göreli sırasının değişebileceği teyit edilmiştir.
- Küçük veri kümelerinde ($n \leq 10$) Quick Sort ile Bubble Sort arasındaki performans farkının gözle görülür olmadığı; asıl farkın büyük veri kümelerinde ortaya çıktığı değerlendirilmiştir.

8. Sunum Özeti

Aşağıdaki maddeler, çalışmanın sınıf arkadaşlarına sunumunda kullanılacak temel başlıkları özetlemektedir.

8.1. Temel Prensipler

- Böl ve yönet (divide and conquer) yaklaşımı
- Pivot seçimi ve dizinin ikiye ayrılması (Parçala/Partition)
- Alt listeler üzerinde özyinelemeli (recursive) tekrar

8.2. Uygulama Süreci

- Algoritmanın doğal dille tanımlanması
- Sözde kodun yazılması
- Akış şemasının çizilmesi
- Beş farklı senaryo ile test edilmesi ve sonuçların doğrulanması

8.3. Analiz Sonuçları ve Kapanış

- Ortalama durumda $O(n \log n)$, en kötü durumda $O(n^2)$ zaman karmaşıklığı
- Büyük veri kümelerinde yüksek performans, küçük veri kümelerinde Bubble Sort'a kıyasla belirgin olmayan fark
- Tüm test senaryolarının başarıyla geçmesiyle akış şeması ve sözde kodun doğruluğu teyit edilmiştir

---

Bu belge; görev tanımı, karşılaştırma tablosu, sözde kod, akış şeması, test senaryoları, kapsamlı rapor ve sunum özetini tek bir dokümanda bir araya getirmektedir.