

Algoritma ve Programlama II

Hafta 6

Sıralama (Sorting) ve Arama (Searching) Algoritmaları

Sıralama

- ▶ Bir grup veriyi azalan veya artan şekilde yerleştirme.
- ▶ Bilgisayar sistemleri için veri sıralama çok önemlidir.
- ▶ Sıralama işlemi, hem arama işlemlerini hem de bir grup veriyi listeleme işlemini hızlandırmaya yarar.
- ▶ En popüler sıralama algoritmaları:
 - Bubble sort (Kabarcık sıralaması)
 - Insertion sort (Araya ekleme sıralaması)
 - Selection sort (Seçim sıralaması)
 - Quick sort (Hızlı sıralama)



Bubble Sort (Kabarcık Sıralaması)

- ▶ Zaman karmaşıklığı $O(n^2)$
- ▶ Eğer n adet elemandan c adet eleman sıralı değilse zaman karmaşıklığı $O(c \cdot n)$
- ▶ Algoritmanın tasarımı kolaydır ancak algoritma verimli değildir.
- ▶ Az sayıda eleman üzerinde ya da çoğu elemanı zaten sıralanmış listeler üzerinde kullanılabilir.



Bubble Sort (Kabarcık Sıralaması)

Çalışma Mantığı

Bubble Sort, en büyük elemanı her turda listenin en sonuna "baloncuk gibi" taşıyarak çalışan bir sıralama algoritmasıdır.

- Dizi boyunca yan yana duran iki eleman karşılaştırılır.
- Eğer soldaki eleman büyükse sağdaki ile yer değiştirir.
- En büyük eleman en sona ulaştığında, tekrar baştan başlanır ve kalan elemanlar için aynı işlem uygulanır.
- Tüm dizi sıralanana kadar devam eder.



Bubble Sort (Kabarcık Sıralaması)

Sıralanacak dizi: [5, 3, 8, 4, 2]

1. Pass (Birinci Tur)

- (5, 3) karşılaştırılır, $5 > 3$ olduğundan yer değiştirir \rightarrow [3, 5, 8, 4, 2]
- (5, 8) karşılaştırılır, değişim olmaz \rightarrow [3, 5, 8, 4, 2]
- (8, 4) karşılaştırılır, $8 > 4$ olduğundan yer değiştirir \rightarrow [3, 5, 4, 8, 2]
- (8, 2) karşılaştırılır, $8 > 2$ olduğundan yer değiştirir \rightarrow [3, 5, 4, 2, 8]

2. Pass (İkinci Tur)

- (3, 5) karşılaştırılır, değişim yok \rightarrow [3, 5, 4, 2, 8]
- (5, 4) karşılaştırılır, $5 > 4$ olduğundan yer değiştirir \rightarrow [3, 4, 5, 2, 8]
- (5, 2) karşılaştırılır, $5 > 2$ olduğundan yer değiştirir \rightarrow [3, 4, 2, 5, 8]

3. Pass (Üçüncü Tur)

- (3, 4) karşılaştırılır, değişim yok \rightarrow [3, 4, 2, 5, 8]
- (4, 2) karşılaştırılır, $4 > 2$ olduğundan yer değiştirir \rightarrow [3, 2, 4, 5, 8]

4. Pass (Dördüncü Tur)

- (3, 2) karşılaştırılır, $3 > 2$ olduğundan yer değiştirir \rightarrow [2, 3, 4, 5, 8]

Dizi sıralandı! Sonuç: [2, 3, 4, 5, 8]



Bubble Sort (Kabarcık Sıralaması)

```
void bubbleSort(int dizi[], int n) {  
    int gecici;  
  
    for (int i = 0; i < n - 1; i++) { // Dış döngü: Diziyi n-1 kez sıralamaya çalışır  
        for (int k = 0; k < n - 1 - i; k++) { // İç döngü: Her turda en büyük elemanı sona taşı  
            if (dizi[k] > dizi[k + 1]) { // Yan yana elemanları karşılaştır  
                gecici = dizi[k];  
                dizi[k] = dizi[k + 1];  
                dizi[k + 1] = gecici; // Yer değiştirme işlemi  
            }  
        }  
    }  
}
```



Bubble Sort (Kabarcık Sıralaması)

```
1 #include <stdio.h>
2
3 void bubbleSort(int [],int);
4 int main(void)
5 {
6     int i=0,a[5];
7     printf("Sıralamak istediğin 5 sayı gir\n");
8     while(i<5){
9         scanf("%d",&a[i]);
10        i++;
11    }
12    i=0;
13    bubbleSort(a,5);
14
15    printf("Bubble sort işleminden sonra...\n");
16    while(i<5){
17        printf("%d ",a[i]);
18        i++;
19    }
20    return 0;
21 }
```

Insertion Sort (Araya Ekleme)

- ▶ Sıralı bir listeye eleman eklemek için uygundur.
- ▶ Sıralı bir listeye eleman eklemenin karmaşıklığı : $O(n)$
- ▶ Eğer liste veya dizi sıralı değilse karmaşıklığı : $O(n^2)$



• Insertion Sort (Araya Ekleme)

Çalışma Mantığı

Insertion Sort, bir kart destesi sıralamaya benzer şekilde çalışır:

- İlk eleman zaten sıralı kabul edilir.
- Sonraki her eleman, **öncesindeki sıralı kısmın içine uygun yere yerleştirilir.**
- Eleman yerleştirilirken, büyük olan elemanlar sağa kaydırılır.
- Bu işlem dizi tamamen sıralanana kadar devam eder.



• Insertion Sort (Araya Ekleme)

Sıralanacak dizi: [5, 3, 8, 4, 2]

1. Adım: (İlk eleman zaten sıralı kabul edilir)

- Sıralı kısım: [5] | 3, 8, 4, 2

2. Adım: (3 elemanı sıralı kısmın içine yerleştirilir)

- 3, 5'ten küçük olduğu için sola kaydırılır → [3, 5] | 8, 4, 2

3. Adım: (8 elemanı sıralı kısma eklenir)

- 8 zaten en büyük, değişiklik olmaz → [3, 5, 8] | 4, 2

4. Adım: (4 elemanı sıralı kısma eklenir)

- 4, 8'den küçük → 8 sağa kaydırılır
- 4, 5'ten küçük → 5 sağa kaydırılır
- 4 yerleştirilir → [3, 4, 5, 8] | 2

5. Adım: (2 elemanı sıralı kısma eklenir)

- 2, 8'den küçük → 8 sağa kaydırılır
- 2, 5'ten küçük → 5 sağa kaydırılır
- 2, 4'ten küçük → 4 sağa kaydırılır
- 2, 3'ten küçük → 3 sağa kaydırılır
- 2 yerleştirilir → [2, 3, 4, 5, 8]

Dizi sıralandı! Sonuç: [2, 3, 4, 5, 8]



Insertion Sort (Araya Ekleme)

```
void insertionSort(int dizi[], int n) {  
    int i, j, gecici;  
  
    for (i = 1; i < n; i++) { // İlk eleman zaten sıralı kabul edilir, bu yüzden i=1'den başlı  
        gecici = dizi[i]; // Şu anki elemanı gecici değişkene al  
        j = i - 1;  
  
        // Büyük olan elemanları bir adım sağa kaydır  
        while (j >= 0 && dizi[j] > gecici) {  
            dizi[j + 1] = dizi[j];  
            j--;  
        }  
  
        dizi[j + 1] = gecici; // Boşalan yere küçük elemanı yerleştir  
    }  
}
```



Insertion Sort (Araya Ekleme)



```
1 #include <stdio.h>
2
3 void insertionSort(int [],int);
4 int main(void)
5 {
6     int i=0,a[5];
7     printf("Siralamak istediğin 5 sayı gir\n");
8     while(i<5){
9         scanf("%d",&a[i]);
10        i++;
11    }
12    i=0;
13    insertionSort(a,5);
14
15    printf("Insertion sort isleminde sonra...\n");
16    while(i<5){
17        printf("%d ",a[i]);
18        i++;
19    }
20    return 0;
21 }
```



• Selection Sort (Seim Sıralaması)

alıřma Mantıėı

Selection Sort, her adımda dizinin sıralanmamıř kısmındaki en kk elemanı bulur ve bunu sıralı kısmın sonuna yerleřtirerek alıřır:

- İlk eleman sıralı kabul edilir, ancak yer deėiřimi gerekebilir.
- Dizinin geri kalan kısmındaki en kk eleman bulunur.
- Bu eleman, sıralanmamıř kısmın bařındaki eleman ile yer deėiřtirilir.
- İřlem dizi tamamen sıralanana kadar devam eder.



Selection Sort (Seim Sıralaması)

Sıralanacak dizi: [5, 3, 8, 4, 2]

1. Adım: (En küçük eleman bulunup ilk sıraya alınır)

- Dizide en küçük elemanı arıyoruz: 2
- 2, ilk eleman 5 ile yer deęiştirir.

Sıralı kısım: [2] | 3, 8, 4, 5

2. Adım: (En küçük eleman bulunup sıralı kısma eklenir)

- Geriye kalan dizide en küçük elemanı arıyoruz: 3
- 3, yerinde olduęu için deęişiklik yapılmaz.

Sıralı kısım: [2, 3] | 8, 4, 5

3. Adım: (En küçük eleman bulunup sıralı kısma eklenir)

- Geriye kalan dizide en küçük elemanı arıyoruz: 4
- 4, 8 ile yer deęiştirir.

Sıralı kısım: [2, 3, 4] | 8, 5

4. Adım: (En küçük eleman bulunup sıralı kısma eklenir)

- Geriye kalan dizide en küçük elemanı arıyoruz: 5
- 5, 8 ile yer deęiştirir.

Sıralı kısım: [2, 3, 4, 5] | 8

5. Adım: (Son eleman sıralı kısma eklenir)

- Son kalan eleman 8 zaten sıralıdır, deęişiklik gerekmez.

Sıralı kısım: [2, 3, 4, 5, 8]

Sonuç:

Dizi sıralandı!

Sonuç: [2, 3, 4, 5, 8]

• Selection Sort (Seçim Sıralaması)

```
void selectionSort(int dizi[], int n)
{
    int i, j;
    int index, enkucuk;

    // Dizinin her elemanı için işlemi tekrar et
    for (i = 0; i < n - 1; i++)
    {
        // Başlangıçta en küçük elemanı dizinin son elemanı olarak kabul et
        enkucuk = dizi[n - 1];
        index = n - 1;

        // i. indisten itibaren dizinin en küçük elemanını bul
        for (j = i; j < n - 1; j++)
        {
            // Eğer şu anki eleman mevcut en küçük elemandan küçükse, güncelle
            if (dizi[j] < enkucuk)
            {
                enkucuk = dizi[j]; // Yeni en küçük elemanı ata
                index = j; // En küçük elemanın indeksini güncelle
            }
        }

        // Bulunan en küçük elemanı i. indeks ile değiştir
        dizi[index] = dizi[i];
        dizi[i] = enkucuk;
    }
}
```



• Selection Sort (Seçim Sıralaması)

```
1 #include <stdio.h>
2
3 void selectionSort(int [],int);
4 int main(void)
5 {
6     int i=0,a[5];
7     printf("Sıralamak istediğin 5 sayı gir\n");
8     while(i<5){
9         scanf("%d",&a[i]);
10        i++;
11    }
12    i=0;
13    selectionSort(a,5);
14
15    printf("Selection sort işleminden sonra...\n");
16    while(i<5){
17        printf("%d ",a[i]);
18        i++;
19    }
20    return 0;
21 }
```


Quick Sort (Hızlı Sıralama)

Quick Sort, böl ve fethet (divide and conquer) yaklaşımını kullanarak çalışan, verimli ve yaygın kullanılan bir sıralama algoritmasıdır.

Çalışma Prensipleri:

1. **Pivot Seçimi:** Diziden bir eleman pivot olarak seçilir. (Genellikle ilk, son veya rastgele bir eleman)
2. **Bölme (Partitioning):** Dizideki elemanlar pivot etrafında düzenlenir:
 - Pivot'tan küçük olanlar sola, büyük olanlar sağa yerleştirilir.
3. **Tekrarlama (Recursion):**
 - Pivot'un solundaki ve sağındaki alt dizilere aynı işlem uygulanır.
 - Bu işlem dizinin tamamı sıralanana kadar devam eder.



Quick Sort (Hızlı Sıralama)

Sıralanacak dizi: [5, 3, 8, 4, 2]

1. Adım: Pivot Seçilir ve Diziyi İkiye Bölme

Pivot: 5 (İlk elemanı seçiyoruz)

Diziyi pivot etrafında bölelim:

- Pivot'tan küçük olanlar: [3, 4, 2]
- Pivot: 5
- Pivot'tan büyük olanlar: [8]

Yeni hali: [3, 4, 2] 5 [8]

2. Adım: Sol Alt Diziyeye Quick Sort Uygulama ([3, 4, 2])

Pivot: 3

Diziyi pivot etrafında bölelim:

- Pivot'tan küçük olanlar: [2]
- Pivot: 3
- Pivot'tan büyük olanlar: [4]

Yeni hali: [2] 3 [4]

Artık [2] ve [4] tek elemanlı olduğu için sıralama tamamlandı.

3. Adım: Sağ Alt Diziyeye Quick Sort Uygulama ([8])

- [8] tek elemanlı olduğu için işlem gerekmez.

Sonuç:

Bütün parçalar birleştirildiğinde:

[2, 3, 4] 5 [8] → [2, 3, 4, 5, 8]

Özet

1. Pivot seçilir.
2. Küçükler sola, büyükler sağa gider.
3. Sol ve sağ alt dizilere aynı işlem uygulanır.
4. Tüm alt diziler sıralandığında, sonuç birleşerek sıralı diziyi oluşturur.

Böylece Quick Sort, ortalama $O(n \log n)$ karmaşıklığı ile oldukça hızlı çalışır!



Quick Sort (Hızlı Sıralama)

```
// Quick Sort algoritması
void quickSort(int dizi[], int low, int high) {
    if (low < high) {
        int pi = partition(dizi, low, high); // Diziyi böl

        quickSort(dizi, low, pi - 1); // Sol alt diziyi sırala
        quickSort(dizi, pi + 1, high); // Sağ alt diziyi sırala
    }
}

// Diziyi pivot etrafında bölen fonksiyon
int partition(int dizi[], int low, int high) {
    int pivot = dizi[high]; // Pivot olarak son eleman seçildi
    int i = (low - 1); // Küçük elemanların yerleştirileceği indeks

    for (int j = low; j < high; j++) {
        if (dizi[j] < pivot) { // Eğer mevcut eleman pivot'tan küçükse
            i++;
            swap(&dizi[i], &dizi[j]); // Küçük elemanı sol tarafa al
        }
    }
    swap(&dizi[i + 1], &dizi[high]); // Pivot'u yerine yerleştir
    return (i + 1); // Pivot'un yeni konumunu döndür
}

// İki elemanın yerini değiştiren yardımcı fonksiyon
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Arama

- ▶ Bir dizi içerisinde belli bir elemanı bulma sürecine arama denir.
- ▶ İki arama tekniği vardır:
 - Doğrusal Arama (Linear Search)
 - İkili Arama (Binary Search)

Doğrusal Arama (Linear Search)

Doğrusal arama, dizinin başından sonuna kadar tek tek kontrol ederek aranan elemanı bulmaya çalışır.

Sıralı olmasına gerek yoktur.

Çalışma Prensipleri

1. Dizinin ilk elemanından başlanarak tüm elemanlar sırayla kontrol edilir.
2. Eğer eleman bulunursa indeksi döndürülür.
3. Sonuna kadar bulunamazsa, eleman olmadığı sonucuna varılır.



Doğrusal Arama (Linear Search)

Aranan sayı: 4

Verilen dizi: [5, 3, 8, 4, 2]

1. Adım: İlk eleman kontrol edilir

- $5 \neq 4 \rightarrow$ Devam et

2. Adım: İkinci eleman kontrol edilir

- $3 \neq 4 \rightarrow$ Devam et

3. Adım: Üçüncü eleman kontrol edilir

- $8 \neq 4 \rightarrow$ Devam et

4. Adım: Dördüncü eleman kontrol edilir

- $4 = 4 \rightarrow$ Eleman bulundu!

Sonuç: Eleman, 3. indeksde bulundu. |



Doğrusal Arama (Linear Search)

```
21 int linearSearch(int dizi[],int aranan,int n)
22 {
23     for (int i = 0; i < n; i++)
24     {
25         if (dizi[i] == aranan)
26             return i;
27     }
28     return -1;
29 }
```

Doğrusal Arama (Linear Search)

```
1 #include <stdio.h>
2
3 int linearSearch(int [],int,int);
4 int main(void)
5 {
6     int dizi[] = { 1, 3, 5, 7, 8, 10, 11 };
7     int sonuc, aranan,i;
8     for (i = 0; i < 7; i++)
9         printf("%d ",dizi[i]);
10
11     printf("Arananı giriniz:");
12     scanf("%d",&aranan);
13
14     sonuc = linearSearch(dizi, aranan,7);
15     if (sonuc == -1)
16         printf("\nAranan dizide yok\n");
17     else
18         printf(sonuc + ". sırada bulundu\n");
19 }
```


İkili Arama (Binary Search)

İkili arama, sadece sıralı dizilerde çalışan, çok daha hızlı bir arama algoritmasıdır. Her adımda diziyi ikiye bölerek çalışır.

Çalışma Prensibi

1. Dizi sıralı olmalıdır.
2. Ortadaki eleman kontrol edilir:
 - Eğer aranan sayı **ortadaki sayıdan küçükse**, sol yarıda aranır.
 - Eğer **büyükse**, sağ yarıda aranır.
 - Eğer **eşitse**, eleman bulundu!
3. Alt diziler için işlem **tekrarlanır**.



İkili Arama (Binary Search)



Aranan sayı: 4

Verilen sıralı dizi: [2, 3, 4, 5, 8]

1. Adım: Ortadaki eleman bulunur

- Orta = 4. indeks \rightarrow 4
- $4 = 4 \rightarrow$ Eleman bulundu!

Sonuç: Eleman, 2. indeksde bulundu.

Not: Eğer aranan sayı 6 olsaydı:

- Ortadaki eleman 4 olduğundan sağ yarıya bakılırdı.
- Yeni alt dizi: [5, 8]
- Orta = 5 $\rightarrow 5 < 6$ olduğundan sağ yarıya bakılırdı.
- Yeni alt dizi: [8]
- $8 \neq 6 \rightarrow$ Eleman bulunamadı.



İkili Arama (Binary Search)



```
#include <stdio.h>

// İkili arama fonksiyonu (recursive)
int binarySearch(int dizi[], int low, int high, int aranan) {
    if (low <= high) {
        int mid = low + (high - low) / 2; // Ortadaki eleman

        // Eğer ortadaki eleman aranan sayı ise döndür
        if (dizi[mid] == aranan)
            return mid;

        // Eğer aranan eleman ortadan küçükse sol tarafta ara
        if (dizi[mid] > aranan)
            return binarySearch(dizi, low, mid - 1, aranan);

        // Eğer aranan eleman ortadan büyükse sağ tarafta ara
        return binarySearch(dizi, mid + 1, high, aranan);
    }

    return -1; // Eleman bulunamazsa -1 döndür
}
```



İkili Arama (Binary Search)



```
// Diziyi ekrana yazdıran fonksiyon
void printArray(int dizi[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", dizi[i]);
    printf("\n");
}

// Ana program
int main() {
    int dizi[] = {2, 3, 4, 5, 8}; // Sıralı dizi
    int n = sizeof(dizi) / sizeof(dizi[0]);
    int aranan = 4; // Aranacak eleman

    printf("Dizi: ");
    printArray(dizi, n);

    int sonuc = binarySearch(dizi, 0, n - 1, aranan);

    if (sonuc != -1)
        printf("Eleman %d. indeksde bulundu.\n", sonuc);
    else
        printf("Eleman bulunamadı.\n");

    return 0;
}
```




Kaynaklar

- ▶ Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- ▶ Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- ▶ Paul J. Deitel, “C How to Program”, Harvey Deitel.
- ▶ “A book on C”, All Kelley, İra Pohl

Doç. Dr. Caner ÖZCAN, KBÜ Yazılım Mühendisliği www.canerozcan.net*

- ▶ Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- ▶ Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- ▶ Paul J. Deitel, “C How to Program”, Harvey Deitel.
- ▶ “A book on C”, All Kelley, İra Pohl

* Bu dersin slaytları genelde bu kaynaktan türetilmiştir.



Ders içerikleri ve duyurular için dersin web sitesine aşağıdaki adresten ulaşabilirsiniz.

hru-algpro.github.io

S o r u l a r

?



Dinlediğiniz için teşekkürler

