Arama Algoritmaları

5.1 Ardışıl Arama

Arama, bir anahtara bağlı olarak verinin yapı üzerinde bulunup bulunmadığının araştırılmasıdır. Ardışıl aramada verilen anahtarın liste üzerinde varlığının araştırılması bir pozisyondan hemen takip eden pozisyona geçerek araştırma yapılır. Yani pozisyonlar müteakip olarak ardışıl (sequential search) bir şekilde araştırılır.

Ardışıl aramanın dizi için Java kodlaması aşağıdaki gibi verilebilir.

```
public int search(int nElems, int intArray, int target) {
        int pos = 0;
        while ( pos < nElems && intArray[pos] != target)
                pos++;
        if (intArray[pos] == target)
                return pos;                             // pos pozisyonunda buluntu
        else
                return –1;                              // bulunamadı
}
```

For döngüsü kullanılarak ise şu şekilde yazılabilir. Programda numElems elemanlı bir intArray dizisinde target verisinin dizideki yerini bulur. Eğer dizide bulunursa true, mevcut değilse false bilgisini geri döndürür. Girdi olarak bir dizi, eleman sayısı, aranan bilgiyi ve indis değişkenini alır.

```
bool sequential_search(int intArray[],int numElems, int target,  int position){
    for (int pos = 0; pos < numElems; pos++)
        if (intArray[pos] == target)
                return true;
    return false;
    };
```

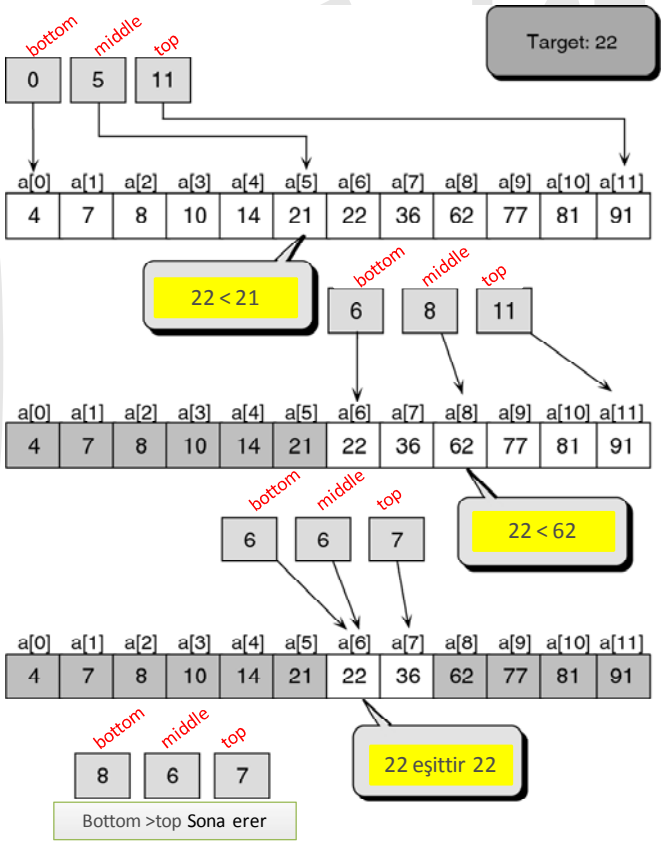
Bağlı liste için ardışıl arama mantığı Java kodlamasi ise şöyle yazılabilir.

```
private Node search(Node Head, int target) {
 //Eğer bulunursa bulunulan düğümü, bulunamaz ise null geri çevirir
        Node p=Head;
        while (p!=null&& !found){
                if(p.element=target)
                        found = true;
                else
                        p=p.next;
        }
        return p;
}
```

5.2 İkili Arama

İkili arama algoritmasi belirli şartlar altında oldukça hızlı çalışan bir algoritmadır. Ardışıl aramada en kötü durumda bütün pozisyonların incelenmesi gerekliliği vardır. Karmaşıklığı O(N) olarak ifade edilebilir.

İkili arama statik yapıda ve sıralı olması durumunda dizideki ortadaki anahtar ile karşılaştırmayı yapar ve küçük veya büyük olması ile de ilgilenir. Dizinin küçükten büyüğe doğru sıralı olduğu kabul edilirse, hedef olarak aranan veri karşılaştırılan değerden küçük ise arama alt yarıda büyük ise üst yarıda yapılır. İkili aramanın çalışması şekil 5.1'de verilmiştir. Burada aranan hedef 22'dir.



Şekil 5.1. İkili aramanın çalışma şeması.

Bu arama algoritması oldukça hızlı çalışır. Örneğin veri uzunluğu ikinin kuvveti olacak şekilde 1024 adet olduğu kabul edilirse, ardışıl arama en kötü durumda bütün pozisyonları bakar ve bulamaz. Buda en az 1024 karşılaştırma yapar. Oysaki ikili aramada 1024 veri için en kötü durumda 10 karşılaştırma ile aranan verinin dizide olmadığı belirlenebilir. Şöyle ki karşılaştırmalarda aranacak veri sayısı sırayla 512,256, 128, 64, 32, 16, 8, 4 ,2 düşürülerek 10 karşılaştırma ile bulunur. Bu algoritmanın analizi daha önceli bölümlerde yapılmış ve O(log(N) olarak bulunmuştu.

İkili arama algoritmasının kodlaması farklı şekillerde yapılabilir. Bunlardan ilki eşitliği tanıyan biçimi olarak isimlendirilen dizi ikiye ayrıldığında ortadaki veri anahtarı ile hedef anahtarın karşılaştırılması ve eğer aranan anahtar bu değilse küçük/büyük operatörünü devreye sokulmasına dayanır. Yani veri ortadaki anahtar değilse, aranan anahtar ortadakinden küçükse alt yarıda büyükse üst yarıda üst yarıda arama yapılmasıdır. Bu da eşitliğin tanınması ile üç yollu karşılaştırması biçiminin karşımıza çıkarır. Üç yoldan kasıt, eşit, küçük ve büyük olmasıdır.

```
int binarySearch2(int array[], int numElems, int target){
// Eşitliği tanıyan üç-yollu biçim
// Eğer bulunrsa indeks, bulunamaz ise -1 döndürür.
        int bottom=0,
        top = numElems-1,
        middle;
        bool found = false;
        while (!found && bottom <= top){            //Bulununcaya veya arama aralığı bitinceye kadar
                middle = (bottom + top)/2;
                if (array[middle] == target)        //Ortaya bak
                        found true
                else if (array[middle] < target)
                                bottom = middle +1  //Üst yarıda ara
                        else
                                 top = middle -1;   //Alt yarıda ara
        }
        if (found)
                return middle
        else
                return -1;
}
```

İkili aramada alt yarıda veya üst yarıda arama mantığının dizinin tamamında arama mantığından farkı yoktur. Bir problemin kendine benzer daha küçük alt parçalara ayrılabiliyorsa özyineli (recursive) yapılabileceği hatırlanırsa, ikili aramanın da özyineli olarak gerçekleştirilebileceği açıktır. Aşağıda üç yollu mantığın özyineli hale getirilmiş hali verilmiştir.

```
int binarySearch3(int intArray[], int bottom, int top,int target){
// Eşitliği tanıyan üç-yollu özyinelemeli biçim
// Eğer bulunrsa indeks, bulunamaz ise -1 döndürür.
        int middle;
        if (bottom < top){
                mid = (bottom + top)/2;
                if (intArray[middle] == target){
                        return middle;
                } else if (intArray[middle] < target)
                                return binarySearch3(intArray[], middle+1, top, target);
                        else
                                return binarySearch3(intArray[], bottom, middle-1, target);
                }
        else return -1;
}
```

İkili aramanın üç yollu biçiminde döngü içerisinde iki karşılaştırma yapılır. Birincisi eşit mi diye bakarken diğeri küçük/büyük olmasını değerlendirir. Bu ikili algoritmayı tek karşılaştırma ile yapmak mümkündür. Sadece küçük/büyük karşılaştırması yapılarak arama aralığı daraltılır ve arama aralığı kalmayıncaya kadar daraltma devam edilir. Arama aralığı kalmayınca döngüden çıkılır. İşte kritik nokta bu döngüden çıkma durumunda saklıdır. Veri bulundu da mi çıkıldı yoksa bulunmadan mı çıkıldı? Bu sorunun cevabı ise döngü sonunda indisteki veriye eşit mi karşılaştırması yapılarak karar verilir. Bu algoritmanın Java kodlaması ise aşağıdaki gibi verilebilir. Bu kodlama şeklinde verinin bulunduğu yer bottom indisi ile kontrol edilir.

```
int  binarySearch1(int array[], int numElems, int target){
// Eşitliği tanımayan iki-yollu biçim
// Eğer bulunursa indeks, bulunamaz ise -1 döndürür.
        int bottom=0,
        top = numElems-1,
        middle;
        while (bottom < top) {
                mid = (bottom + top)/2;
                if (array[middle] < target)             //Orta küçük mü?
                        bottom = middle + 1             //Üst yarıya bak
                else
                        top = middle;                   //Alt yarıya bak
        }
        if (top < bottom)
                return -1                               //Dizi mevcut değilse numElems=0
        else if (array[bottom] = = target)
                return
                        bottom;                         // Bulundu
         else
                return -1;                              // Bulunamadı
              }
}
```