BÖLÜM 23. ÖZDEVİNİM, BÖL VE YÖNET

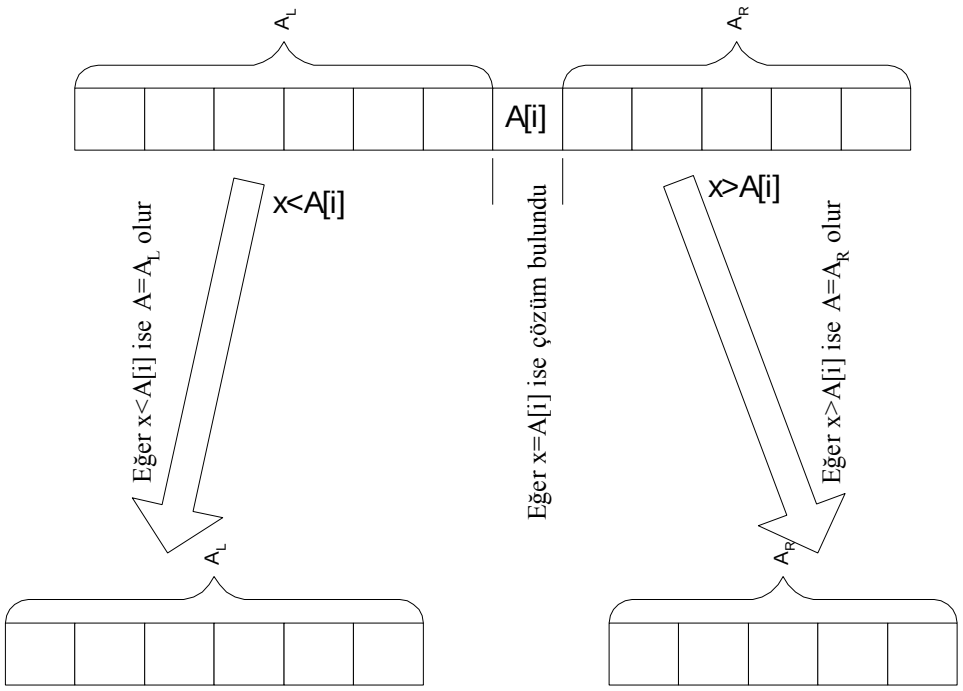
Bu yöntem tarihte Avrupa devletleri tarafından Osmanlı İmparatorluğuna uygulanmak istenmiştir. Çünkü küçük parçaları yönetmek daha kolaydır ve birde yapı küçüldükçe komplekslik azalır ve çözüme ulaşmak daha da kolaylaşır.

Algoritma tasarım yöntemi olarak böl ve yönet her problemin çözümünde kullanılamaz. Eğer problem küçük problemlere parçalandığında ilk problemin taşımış olduğu özellikler hala sağlanıyorsa, diğer bir deyişle küçük problemlerin kendi içlerinde çözülmeleri bütün çözümün bir parçasını oluşturuyorsa, bu durumda böl ve yönet mantığı ile verilen probleme çözüm üreten bir algoritma tasarımı yapılabilir demektir. Bu yönteme tümden gelim tasarımı da (top-down design) denilebilir, çünkü tümden küçük parçalara gidilmektedir. Bu mantığın daha iyi anlaşılabilmesi için bazı örnekler ele alınacaktır.

Örnek 23.1

Bir sıralı dizi içerisinde verilen bir elemanın var olup olmadığını test etmek ve eğer varsa, pozisyonunu test etme problemi böl ve yönet mantığı ile çözülebilir. Sıralı dizi A dizisi olsun ve A dizisi içerisinde var olup olmadığı kontrol edilecek olan eleman da x olsun. İlk olarak x ile A dizisinin ortadaki elemanı karşılaştırılır, eğer x bu elemandan küçük ise, x' in sağda kalan elemanlardan herhangi birine eşit olması mümkün değildir ve bundan dolayı soldaki elemanlar incelenmelidir. Bu sebepten, dizi ortadan iki parçaya bölünür ve sol parça yeni bir dizi olarak ele alınır ve bu dizi içerisinde x elemanı aranır.

$A=\{a_1,a_2,...,a_n\}$ şeklinde olsun. İlk adımdan sonra dizi A_L ve A_R şeklinde iki parçaya bölünür. Eğer $x<A[j]$ ise, x elemanı A_L dizisi içindeki elemanlardan birine eşit olabilir. Bundan dolayı algoritma bu dizi parçası ile çağırılır. Eğer $x=A[i]$ ise zaten çözüm bulunmuştur ve başka işlem yapmaya gerek yoktur. Eğer $x>A[i]$ ise, x elemanı A_R dizi parçası içindeki elemanlardan birine eşit olabileceğinden dolayı algoritma bu dizi parçası üzerinde özyinelemeli olarak çağırılır. Diziyi bölme şekli aşağıdaki gibi olur.



Şekil 23.1. *İkili arama algoritmasının bir kez çalışmasının şekli.*

Şekil 23.1' den de anlaşılacağı gibi eğer x=A[i] ise, çözüm bulundu demektir ve işleme daha fazla devam edilmez. Eğer x<A[i] ise, dizi sıralı olduğundan x elemanının A[i] ve onun sağında olması mümkün değildir ve bundan dolayı $A \leftarrow A_L$ yapılır ve bundan sonraki adım yeni A değeri üzerinde çalışır. Eğer x>A[i] ise, x elemanının A[i] ve A_L içinde olması mümkün değildir ve bundan dolayı x elemanı A_R içinde aranır. Bunun için de $A \leftarrow A_R$ yapılır ve bir sonraki adım yeni A üzerinde yapılır.

İlk olarak Şekil 23.1' de verilen yapının ne anlama geldiğini ve hem de algoritmanın mantığının anlaşılması için bir örnek üzerinde algoritmanın çalışması anlatılabilir. A={0,2,3,12,14,20,21,45,78,90} olmak üzere n=10 olur ve başlangıçta i=1 olur. x=78 için algoritmanın çalışması aşağıdaki gibi olur.

Algoritma 23.1. İkiliArama(A,x,i,n)

▷ A parametresi içinde sıralı olan diziyi ve x parametresi de A dizisi içerisinde aranacak olan elemandır. n dizideki eleman sayısını ve i parametresi eğer x dizide varsa, pozisyonunu verecektir.

```
1-  eğer n=0 ise
2-      hata mesajı
3-  eğer n=1 ise
4-     eğer x=A[i] ise
5-        i pozisyonu verir
6-     değilse hata mesajı ver
7-  eğer değilse
8-      j←i+n/2
9-      eğer A[j]≤x ise
10-       İkiliArama(A,x,j,n-n/2)
11-    değilse
12-       İkiliArama(A,x,i,n/2)
```

İlk adım: n>1 olduğundan j=6 olur ve x ile A[6]=20 karşılaştırılır ve x>A[6] olduğundan algoritma özyinelemeli olarak kendisini A dizisinin A[7..10] parçası üzerinde tekrar çağırır. Yani, İkiliArama(A,x,6,5) şeklinde kendisini çağırır.

İkinci Adım: Yine n>1 olduğundan x ile A[8] karşılaştırılır ve j=8 olur. x>A[8] olduğundan algoritma kendisini İkiliArama(A,X,8,2) şeklinde tekrar çağırır.

Üçüncü Adım: j=9 olur ve x=A[9] olduğundan çözüm bulunmuş olur.

Bu işlem başarılı bir arama için böyle olur. Eğer başarısız bir arama yapılırsa, algoritmanın çalışması aşağıdaki gibi olur (x=4).

İlk Adım: n>1 olduğundan j=6 olur ve x ile A[6]=20 karşılaştırılır. x<A[6] olduğundan algoritma kendisini özyinelemeli olarak çağırır ve İkiliArama(A,x,1,5) şeklinde kendisini çağırır.

İkinci Adım: n>1 olduğundan x ile A[3]=3 karşılaştırılır. x>A[3] olduğundan ve j=3 olduğundan algoritma kendisini İkiliArama(A,x,3,3) şeklinde çağırır.

Üçüncü Adım: n>1 olduğundan x ile A[4]=12 karşılaştırılır ve x<A[4] olduğundan algoritma kendisini İkiliArama(A,x,4,2) şeklinde çağırır.

Dördüncü Adım: n>1 olduğundan x ile A[5]=14 karşılaştırılır ve x<A[5] olduğundan algoritma kendisini İkiliArama(A,x,5,1) şeklinde çağırır.

Beşinci Adım: n=1 ve x≠A[5] olduğundan hata mesajı verilir. Bunun anlamı aranan elemanın dizide olmadığıdır.

Bu algoritmanın performansı, zaman bağıntısı tarafından belirlenir. Algoritma özyinelemeli olduğundan zaman bağıntısı da tekrarlı bağıntı olur ve aşağıdaki gibi ifade edilebilir.

$$T(n)=\begin{cases} O(1) & n \le 1 \\ T(n/2)+O(1) & n>1 \end{cases}$$

Bu tekrarlı bağıntı çözülecek olursa,

$$\begin{aligned} T(n) &= T(n/2)+1 \\ &= T(n/4)+2 \\ &= T(n/8)+3 \\ &\cdots\cdots\cdots \\ &= T(n/2^k)+k \end{aligned}$$

elde edilir. $n \approx 2^k$ olduğundan $T(n) \approx T(1)+k$ olur. Buradan $T(n)=T(1)+O(\lg n)=O(\lg n)$ olur.

Örnek 23.2

Böl ve yönet mantığı ile Fibonacci sayıları da hesaplanabilir. Fibonacci sayılarını veren bağıntı aşağıdaki gibidir.

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

Fibonacci sayılarının tanımı özyinelemeli olduğundan Fibonacci sayılarını hesaplayacak olan algoritmada özyinelemeli olur. Fibonacci sayıları için aşağıdaki önemli eşitlikler vardır ve bu eşitliklerle Fibonacci sayılarının ayrıştırılma işlemi yapılabilmektedir. Ayrıştırılma işleminin yapılabilmesi, bu sayıların böl ve yönet mantığı ile hesaplanabileceklerini gösterir.

$$F_{2k-1} = F_k^2 + F_{k-1}^2 \text{ ve } F_{2k} = F_k^2 + 2F_k F_{k-1} \qquad k \in Z^+.$$

Fibonacci sayılarının tanım bağıntısı tekrar yazılabilir

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{\lceil n/2 \rceil}^2 + F_{\lceil n/2 \rceil - 1}^2 & n \geq 2 \text{ ve n tektir} \\ F_{\lceil n/2 \rceil}^2 + 2F_{\lceil n/2 \rceil} F_{\lceil n/2 \rceil - 1} & n \geq 2 \text{ ve n çifttir.} \end{cases}$$

Algoritma 23.2' de verilen n sayısına göre Fibonacci sayısını hesaplayan algoritma görülmektedir. n>1 olarak verildiğinde F_n değerini hesaplamak için $F_{\lceil n/2 \rceil}$ ve $F_{\lceil n/2 \rceil - 1}$ sayıları hesaplanır ve elde edilen sonuçlar birleştirilir. Algoritma 23.2' nin zaman bağıntısını hesaplamak için T(n) bağıntısının monoton artan bir bağıntı olduğu yani $\forall n \geq 1$ için $T(n) \geq T(n-1)$ olduğu kabul edilsin. Böylece $T(\lceil n/2 \rceil) \geq T(\lceil n/2 \rceil - 1)$ olur. Programda n sayısı 2' nin kuvveti olarak düşünüldüğünden dolayı elde edilecek olan T(n) değeri bu algoritmanın zaman bağıntısının üst sınırını teşkil eder ve

Algoritma 23.2. Fibonacci(n)

▹n parametresi kaçıncı terimin hesaplanacağını belirtmektedir.

```
1-  eğer n=0 veya n=1 ise
2-      sonuç←n
3-  değilse
4-      a←Fibonacci((n+1)/2)
5-      b←Fibonacci((n+1)/2-1)
6-      eğer (n mod 2=0) ise
7-          sonuç←a*(a+2*b)
8-      değilse
9-          sonuç←a*a+b*b
```

$$T(n)=\begin{cases} O(1) & n \le 1 \\ 2T(n/2)+O(1) & n > 1 \end{cases}$$

olur. Bu tekrarlı bağıntı iteratif olarak çözülürse,

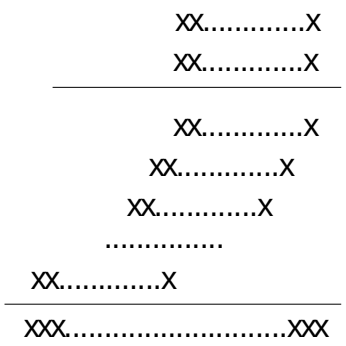
$$\begin{aligned} T(n) &= 2T(n/2)+1 \\ &= 4T(n/4)+1+2 \\ &= 8T(n/8)+1+2+4 \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ &= 2^k T(n/2^k) + \sum_{i=0}^{n-1} 2^i \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ &= nT(1)+n-1 \text{ eğer } n=2^k \text{ olarak kabul edildiyse.} \end{aligned}$$

Böylece $T(n)=2n-1=O(n)$ olur.

Örnek 23.3

İki tane n-bit sayının çarpımı düşünülsün ve eğer bu iki sayı X ve Y olarak gösterilirse, çarpımları XY şeklinde gösterilsin. N-bitlik iki tane sayının çarpımında n^2 tane bit çarpımı yapılır ve $(n-1)^2$ tane de

toplama yapılır. Bu çarpımda eğer böl ve yönet mantığı kullanılırsa, algoritmanın performansı daha da artacaktır.



X ve Y sayıları için X=a b ve Y=c d şeklinde eşitlikler tanımlanmış olsun. Bu durumda a, b, c ve d sayıları (n/2)-bitlik sayılardır.

$X=a2^{n/2}+b$ ve $Y=c2^{n/2}+d$

Seklinde X ve Y sayılarının eşitleri yazıldıktan sonra

$XY=(a2^{n/2}+b)(c2^{n/2}+d)$
$=ac2^{n}+(ad+bc)2^{n/2}+bd$

olur. Bu çarpımda problem boyutları n/2-bit olan 4 tane alt probleme indirgenmiş olur. Bu durumda

$XY=(2^{n}+2^{n/2})ac+2^{n/2}(a-b)(d-c)+(2^{n/2}+1)bd$

olur. Bu durumda problem boyutu n/2-bit olan 3 tane alt probleme bölünmüş olur. Bu durumda zaman bağıntısı

$T(n)\leq 3T(n/2)+kn$

olur. Master teoremi kullanılarak çözüm yapıldığında $T(n)=O\left(n^{\log_2 3}\right)=O\left(n^{1.59}\right)$ elde edilir. Dikkat edilirse, $O(n^2)$ yerine $O(n^{1.59})$ değeri elde edildi. Örnek olarak X=61438521 ve Y=94736407

sayılarının çarpımı XY=5820464730934047 olur. Anlatılan algoritmanın uygulanması için

X_L=6143 ve X_R=8521
Y_L=9473 ve Y_R=6407

olarak seçilsin. $X=X_L.10^4+X_R$ ve $Y=Y_L.10^4+Y_R$ olur. Buradan

$$XY=Y_LX_L.10^8+(X_LY_R+X_RY_L)10^4+X_RY_R$$

olur. Y_LX_L ,X_LY_R, X_RY_L ve X_RY_R olan dört tane çarpma var ve her birindeki sayılar n/2 dijit içermektedir. Çarpımların 10^8 ve 10^4 ile çarpılması işlemi sadece sona sıfır ekleme işlemidir. Toplamların mertebesi O(n) olur ve bu durumda algoritmanın mertebesi

$$T(n)=4T(n/2)+O(n)$$

olur ve $T(n)=O(n^2)$ olur. Bu mertebenin daha iyi olması için özyineleme adım sayısının azaltılması gerekir.

$$X_LY_R+X_RY_L=(X_L-X_R)(Y_R-Y_L)+X_LY_L+X_RY_R$$

eşitliği kullanılabilir. Bu şekilde çarpma sayısı 3' e indirgenir ve toplama sayısı da 4' de çıkar. Bu durumda mertebe

$$T(n)=3T(n/2)+O(n)$$

olur ve $T(n)=O\left(n^{\log_2 3}\right)=O(n^{1.59})$ elde edilir. $D_1=X_L-X_R$ için O(n) ve $D_2=Y_R-Y_L$ için O(n) gerekir. X_LY_L için T(n/2) ve X_RY_R için T(n/2) gerekir. D_1D_2 için de T(n/2) kadar zaman gerekir. $D_3= D_1D_2+ X_LY_L+ X_RY_R$ için O(n) gerekir. $D_3.10^4$ terimi ve $X_LY_L.10^8$ terimi için (her biri için) O(n) gerekir. Bunun sonucunda $X_LY_L.10^8+ D_3.10^4+X_RY_R$ için de O(n) zaman gerekecektir.

Örnek 23.4

Bir kümenin maksimum ve minimum elemanlarının belirlenmesi de böl ve yönet yöntemi ile yapılabilir. Küme S olsun ve ilk olarak n-1 tane karşılaştırma yapılarak maksimum eleman belirlenir ve ondan

sonra n-2 tane karşılaştırma yapılarak minimum eleman tespit edilir. Bu durumda

$T(n)=2n-3=O(n)$

olur. Acaba burdaki karşılaştırma sayısı daha da azaltılabilir mi? Bu sorunun cevabı Algoritma 23.3'te verilmektedir.

Algoritma 23.3. Maksimum_Minimum(S)

▹S kümesi elemanları içermektedir ve elemanların dizilişi hakkında herhangi bir bilgi yoktur.

```
1-  eğer |S|=1 veya |S|=2 ise
2-      bir karşılaştırma ile sonuçlar belirlenir
3-  değilse
4-     S=S1∪S2
5-     (min1,max1)←Maksimum_Minimum(S1)
6-     (min2,max2)←Maksimum_Minimum(S2)
7-     eğer (min1≤min2) ise
8-        min←min1
9-      değilse
10-       min←min2
11-    eğer (max1≤max2) ise
12-       max←max2
13-     değilse
14-       max←max1
```

Eğer n sayısı lgn sayısının katı ise,

$$T(n)=\frac{3}{2}n-2$$

olur, çünkü

$$T(n)=\begin{cases}0 & n=1\\ 1 & n=2\\ T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+2 & n\geq 3\end{cases}$$

olduğundan T(n) değeri daha önce bulunan değerden daha küçük olduğundan böl ve yönet mantığı bu problemde bize avantaj sağlamıştır.

Örnek 23.4

nxn boyutlarında iki matrisin çarpımı yapılmak isteniyor. Klasik matris çarpımı yöntemi ile çarpım yapılırsa, çarpma sayısı $O(n^3)$ ve toplama sayısı $O(n^3)$ olur. Bu algoritmanın performansını artırmak için böl ve yönet yöntemine başvurulabilir. Bunun için (matrisler A ve B olmak üzere)

$$A=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix},\ B=\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\ \text{ve}\ C=\begin{bmatrix}c_{11} & c_{12}\\ c_{21} & c_{22}\end{bmatrix}$$

şeklinde alt matrislere bölme işlemi yapılır. A ve B matrislerinin bölmelenmiş şekilleri kullanılarak yapılacak olan çarpmada C matrisinin parçalarının eşitlikleri tespit edilebilir.

$c_{11}=a_{11}b_{11}+a_{12}b_{21}$
$c_{12}=a_{11}b_{12}+a_{12}b_{22}$
$c_{21}=a_{21}b_{11}+a_{22}b_{21}$
$c_{22}=a_{21}b_{12}+a_{22}b_{22}$

Bu çarpımların daha rahat yapılabilmesi için aşağıdaki eşitlikler yazılabilir.

$P_1=(a_{11}+a_{22})(b_{11}+b_{22})$
$P_2=(a_{21}+a_{22})b_{11}$
$P_3=a_{11}(b_{12}-b_{22})$
$P_4=a_{22}(b_{21}-b_{11})$
$P_5=(a_{11}+a_{12})b_{22}$
$P_6=(a_{21}-a_{11})(b_{11}+b_{12})$
$P_7=(a_{12}-a_{22})(b_{21}+b_{22})$

Bu yeni terimler kullanılarak C matrisinin parçaları belirlenebilir.

$c_{11}=P_1+P_4-P_5+P_7$
$c_{12}=P_3+P_5$
$c_{21}=P_2+P_4$
$c_{22}=P_1+P_3-P_2+P_6$

Matrisler bu şekilde parçalandıktan sonra, bu parçalanmış matris şekilleri alınarak matris çarpımı yapıldığında 7 tane çarpma ve 18 tane toplama işlemi yapılmaktadır. Bu durumda tekrarı bağıntı

$$T(n)=7T(n/2)+\Theta(n^2)= \Theta(n^{lg7})$$

şeklinde olur.

Böl ve yönet mantığı ile tasarlanmış bir algoritmanın zaman bağıntısı

$$T(n)=aT(n/b)+ \Theta(n^k)$$

olur ($a\geq 1$ ve $b>1$). Bu denklemin çözümü

$$T(n)=\begin{cases} O\left(n^{\log_b a}\right) & a > b^k \\ O\left(n^k \lg n\right) & a = b^k \\ O(n^k) & a < b \end{cases}$$

şeklinde olur. $n=b^m$ olsun ve $b^{m-1}=n/b$ ve $n^k=(b^m)^k=b^{mk}=b^{km}=(b^k)^m$. $T(1)=1$ olduğu kabul edilsin ve $\Theta(n^k)$' daki sabit faktör ihmal edilsin.

$$T(b^m)=aT(b^{m-1})+(b^k)^m$$

elde edilir ve her iki taraf a^m değerine bölünürse,

$$\frac{T(b^m)}{a^m}=\frac{T(b^{m-1})}{a^{m-1}}+\left(\frac{b^k}{n}\right)^m$$

elde edilir. Diğer değerler için

$$\frac{T(b^{m-1})}{a^{m-1}} = \frac{T(b^{m-2})}{a^{m-2}} + \left(\frac{b^k}{n}\right)^{m-1}$$

$$\frac{T(b^{m-2})}{a^{m-2}} = \frac{T(b^{m-3})}{a^{m-3}} + \left(\frac{b^k}{n}\right)^{m-2}$$

.....

$$\frac{T(b^1)}{a^1} = \frac{T(b^0)}{a^0} + \left(\frac{b^k}{n}\right)^1$$

elde edilir. Toplama yapıldığında

$$\frac{T(b^m)}{a^m} = 1 + \sum_{i=1}^{m}\left(\frac{b^k}{a}\right)^i = \sum_{i=1}^{m}\left(\frac{b^k}{a}\right)^i$$

elde edilir ve bunun sonucunda

$$T(n) = T(b^m) = a^m \sum_{i=0}^{m}\left(\frac{b^k}{a}\right)^i$$

olur. Eğer $a>b^k$ ise, toplam bir geometrik seri olur ve oranı 1' den küçük olan bir seridir. Bu durumda serinin toplamı sabit olur ve böylece

$$T(n) = O(a^m) = O(a^{\log_b n}) = O(n^{\log_b a})$$

olur. Eğer $a=b^k$ ise, toplam sembolü içindeki her terim 1 olduğundan toplam $1+\log_b n$ terimlerini içerir ve $a=b^k$ eşitliğinden $\log_b a=k$ olur. Zaman bağıntısı

$$T(n) = O\left(a^m \log_b n\right) = O\left(a^{\log_b n} \log_b n\right) = O\left(n^k \log_b n\right) = O\left(n^k \log n\right)$$

Son olarak $a<b^k$ ise, toplam sembolü içindeki terimlerin değeri 1' den büyük olur.

$$T(n) = a^m \frac{\left(b^k / a\right)^{m+1} - 1}{\left(b^k / a\right) - 1} = O\left(a^m \left(b^k / a\right)^m\right) = O\left(b^{km}\right) = O(n^k)$$

olur.

Teorem 23.1

$a\geq 1$, $b>1$ ve $p\geq 0$ için $T(n)=aT(n/b)+\Theta(n^k \lg^p n)$ denkleminin çözümü

$$T(n) = \begin{cases} O\left(n^{\log_b a}\right) & a > b^k \\ O\left(n^{\log_b a} \lg^{p+1} n\right) & a = b^k \\ O\left(n^k \lg^p n\right) & a < b^k \end{cases}$$

olur.

Teorem 23.2

Eğer $\sum_{i=1}^{k} \alpha_i < 1$ ise, $T(n) = \sum_{i=1}^{k} T(\alpha_i n) + O(n)$ denkleminin çözümü $T(n)=O(n)$ olur.

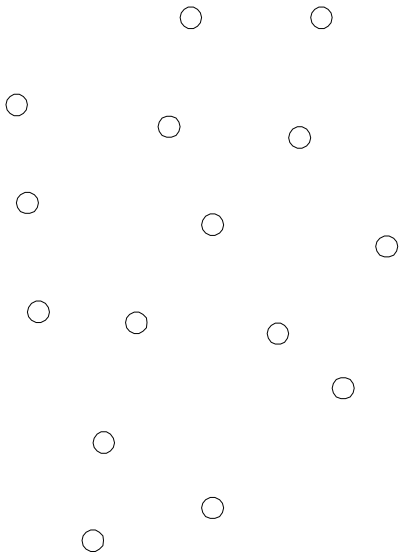
Örnek 23.5

Giriş verisi bir düzlemdeki P noktalar listesi olan bir algoritma ile en yakın noktalar tespit edilmeye çalışılacaktır. $P_1=(x_1,y_1)$ ve $P_2=(x_2,y_2)$ ise, P_1 ve P_2 noktaları arasındaki en kısa mesafe

$$\left[(x_2 - x_1) + (y_2 - y_1)\right]^{1/2}$$

olur. Amaç birbirine en yakın nokta çiftlerini bulmaktır ve iki noktanın aynı olma ihtimali vardır.

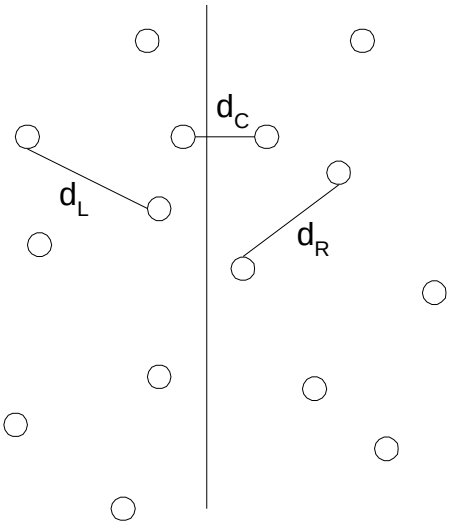
Eğer düzlemde n tane nokta varsa, bu düzlemde n(n-1)/2 tane nokta çifti vardır. Basit bir algoritma ile bu mesafelerin hepsi kontrol edilebilir ve bu algoritmanın mertebesi $O(n^2)$ olur. Acaba bu problemi çözen ve mertebesi daha iyi olan algoritma var mıdır ?



Şekil 23.2. *P kümesinin elemanları düzlemde görünmektedirler.*

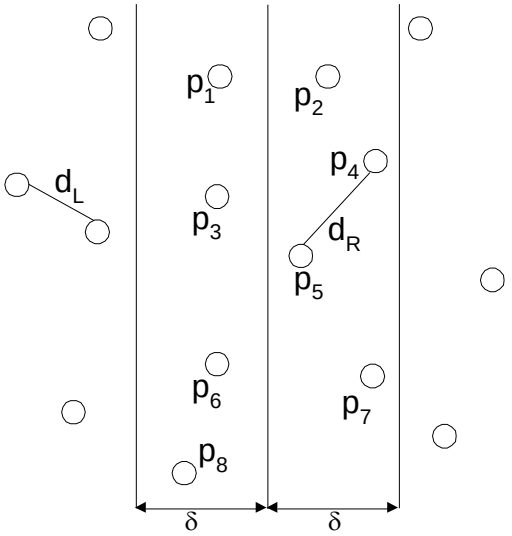
Birbirine en yakın nokta çiftlerini tespit etmek için, ilk olarak noktalar x eksenine göre sıralanır ve mertebeye O(nlgn) değeri eklenir.

Noktalar eksenine göre sıralandıktan sonra P kümesi P_R ve P_L kümelerine bölünür (Şekil 23.3). En kısa mesafelerin her iki ucu da P_L içinde ise, bu uzaklık d_L ile gösterilir; P_R içinde ise, d_R ile gösterilir, bir ucu P_L ve diğer ucu da P_R içinde ise, uzaklık d_C ile gösterilir. d_L ve d_R özyinelemeli olarak hesaplanabilirler, ve d_C'yi hesaplamak problemdir. Amaç mertebesi O(nlgn) olan bir algoritma geliştirmektir ve d_C' yi hesaplamak için ek olarak O(n) olur. Daha önceden bilinmektedir ki özyinelemeli bir algoritmada kendini iki kez çağırma var ve her çağırmada verinin yarısı kullanılır ve yapılan sabit işlemlerin mertebesi O(n) ise, bu algoritmanın mertebesi O(nlgn) olur.



Şekil 23.3. *P kümesi, P_R ve P_L kümelerine ayrıştırılmıştır.*

$\delta=\min(d_L,d_R)$ olsun ve eğer dC değeri δ' yı daha iyi yapıyorsa, bu durumda d_C hesaplanmalıdır. Eğer d_C böyle bir uzaklık ise, d_C' yi tanımlayan noktalar δ bölme doğrusu içinde olmalıdır; bu alana şerit denilsin . Bu bölgeye giren nokta sayısı sınırlı olur ve $\delta=d_R$ kabul edilir (Şekil 23.4).



Şekil 23.4. *δ göre şerit ve şerit içinde kalan noktalar.*

Algoritma 23.4. EnYakinNoktalar(P)

▷P parametresi noktaları içeren bir kümedir.

```
1- i←0,…, (Şerit içinmdeki nokta sayısı-1)
2-     j←i+1,..., (Şerit içinmdeki nokta sayısı-1)
3-         eğer uzaklık(pi,pj) <δ ise
4-             δ=uzaklık(pi,pj)
```

Algoritma 23.5. EnYakinNoktalar(P)

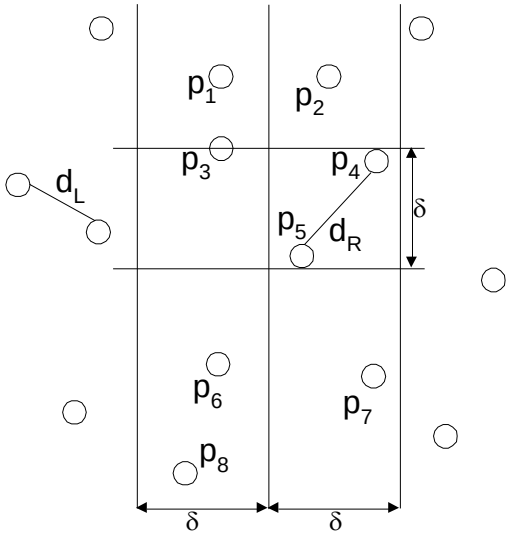
▷P parametresi noktaları içeren bir kümedir.

```
1- i←0,…, (Şerit içinmdeki nokta sayısı-1)
2-     j←i+1,..., (Şerit içinmdeki nokta sayısı-1)
3-         eğer pi ve pj koordinatları δ daha fazla farklı iseler
4-             bitir
5-         değilse
6-             eğer uzaklık(pi,pj)< δ ise
7-                 δ= uzaklık(pi,pj)
```

Algoritma 23.4' te herhangi bir özellik gözetmeksizin $\min(\delta,d_C)$ hesabı yapılmaktadır ve Algoritma 23.5' te ise noktalar y eksenine göre sıralıdır ve $\min(\delta,d_C)$ hesabı biraz daha özenle yapılmaktadır, çünkü bazı şartlar göz önünde tutulmaktadır.

d_C' yi hesaplamak için iki tane strateji vardır. Geniş bir noktalar kümesi içinde şerit içindeki nokta sayısı çok az olur. Ortalama olarak şerit içindeki nokta sayısı $O\left(\sqrt{n}\right)$ olur (n değeri P kümesindeki nokta sayısı). Böylece bu noktalar O(n) zamanda hesaplanmaya tabi tutulurlar (Algoritma 23.4). En kötü durum olarak bütün noktalar şerit içinde olurlarsa, lineer zamanda hesaplama yapılamaz. Bunu geliştirmek için eğer $d_C \le \delta$ ise, iki noktanın y koordinatları arasındaki

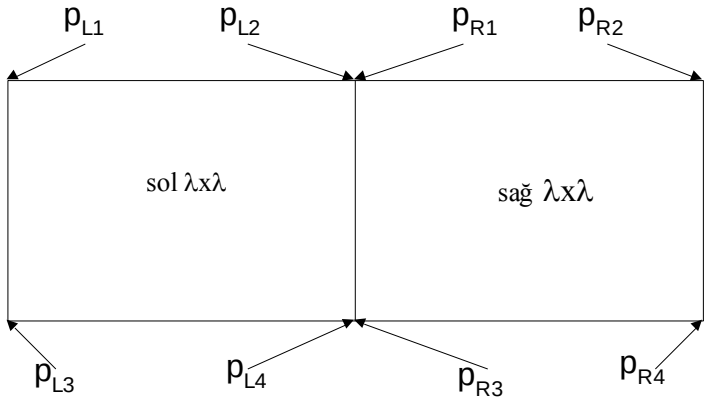
fark δ' dan küçük olur. Şerit içindeki noktalar y eksenine göre sıralı olduğu kabul edilsin. Böylece p_i ve p_j noktalarının y koordinatları arasındaki fark δ' dan büyükse, p_{i+1} noktasına geçilir (Algoritma 23.5).



Şekil 23.5. *Sadece p_4 ve p_5 düşünülür.*

En kötü durumda bir p_i noktası için 7 tane p_j noktası hesaba katılır. Yapılacak hesap, kenarları δ olan kare içindeki (sağda ve solda) noktaları hesaba katmaktır. p_{L1} ve p_{R1} noktalarının koordinatları aynı olmasına rağmen farklı noktalar olabilirler (Şekil 23.6). Bundan dolayı kenarları λ ve 2λ olan dikdörtgenin içindeki noktalar incelenir.

δxδ bölgesindeki noktaların hesabı O(n) zamanda yapılır ve bunun sonucunda elde edilecek çözümün O(nlgn) olacağı az da olsa görünmektedir. İlk etapta y eksenine göre sıralanmış noktalar vardır ve bunun sonucunda O(nlgn) işlem yapılır. Her özyinelemeli çağırmada bu işlem yapılırsa, mertebe $O(nlg^2n)$ olur.



Şekil 23.6. *Bölgelerdeki noktaların hesabı.*

P noktalar kümesi y eksenine göre sıralandıktan sonra Q listesi elde edilir ve x eksenine göre sıralanınca P kümesinin sıralı hali elde edilir. P kümesi p_L ve p_R şeklinde iki parçaya bölündükten sonra Q kümesi q_L ve q_R şeklinde bölünür (Q kümesi bölünürken p_L ve p_R kümeleri referans olarak kullanılırlar).

Örnek 23.6

Böl ve yönet mantığı ile çözülecek problemlerden biri de verilen bir dizinin elemanlarının sıralanması işlemidir. Bu örnekte böl ve yönet yöntemini kullanan sıralama algoritmalarından HızlıSırala (Quicksort) algoritması üzerinde durulacaktır. Bu algoritmanın en kötü durum zaman bağıntısının mertebesi $\Theta(n^2)$' dir. En kötü durumu böyle olmasına rağmen çok rağbet gören bir algoritmadır, ortalamada etkili bir algoritmadır ve ortalama durum mertebesi $\Theta(nlgn)$' dir.

HızlıSırala algoritması iki önemli bölümden oluşmaktadır ve bu bölümler aşağıdaki gibi listelenebilirler.

Bölme:A[p..r] dizisi iki tane boş olmayan alt diziye bölünür ve bu alt diziler A[p..q] ve A[q+1..r] şeklindedir, öyleki A[p..q] içindeki elemanların hepsi A[q+1..r] dizisindeki elemanlardan küçüktür.

Yönet: Özyinelemeli olarak HızlıSırala algoritması çağırılarak A[p..q] ve A[q+1..r] alt dizileri sıralanmaya çalışılır.

Birleştir: Alt diziler sıralı olduklarından, birleştirme safhasında herhangi bir işlem yapmaya gerek yoktur.

Bu işlemleri gerçekleştiren algoritma Algoritma 23.6' da görüldüğü gibidir.

Algoritma 23.6. HızlıSırala(A,p,r)

▷P parametresi sıralanacak diziyi, p bu dizinin ilk elemanın endeksini ve r ise son elemanın endeksini gösterir.

```
1-  Eğer p<r ise
2-      q←Böl(A,p,r)
3-      HızlıSırala(A,p,q)
4-      HızlıSırala(A,q+1,r)
```

Algoritma 23.7. Böl(A,p,r)

▷P parametresi sıralanacak diziyi, p bu dizinin ilk elemanın endeksini ve r ise son elemanın endeksini gösterir.

```
1-  x←A[p]
2-  i←p-1
3-  j←r+1
4-  Sonsuz döngü
5-      A[j]≤x oluncaya kadar devam et
6-          j←j-1
7-      A[i]≥x oluncaya kadar devam et
8-          i←i-1
9-      eğer i<j ise
10-         A[i]↔A[j]
11-     değilse
12-         işlemi bitir ve j' yi döndür.
```

HızlıSırala algoritmasında dizi ilk olarak iki parçaya bölünüyor ve solda kalan parça içindeki elemanlar sağda kalan parça içindeki elemanlardan küçüktürler. Parçaların içindeki elemanların bu şekilde organize edilebilmesi için bir referans lazımdır ve bu referans dizinin ilk elemanıdır ve elemana pivot adı verilir. Bu pivota göre dizi iki parçaya bölünür. Bu algoritmanın yapmış olduğu işlemler aşağıdaki gibi özetlenebilir.

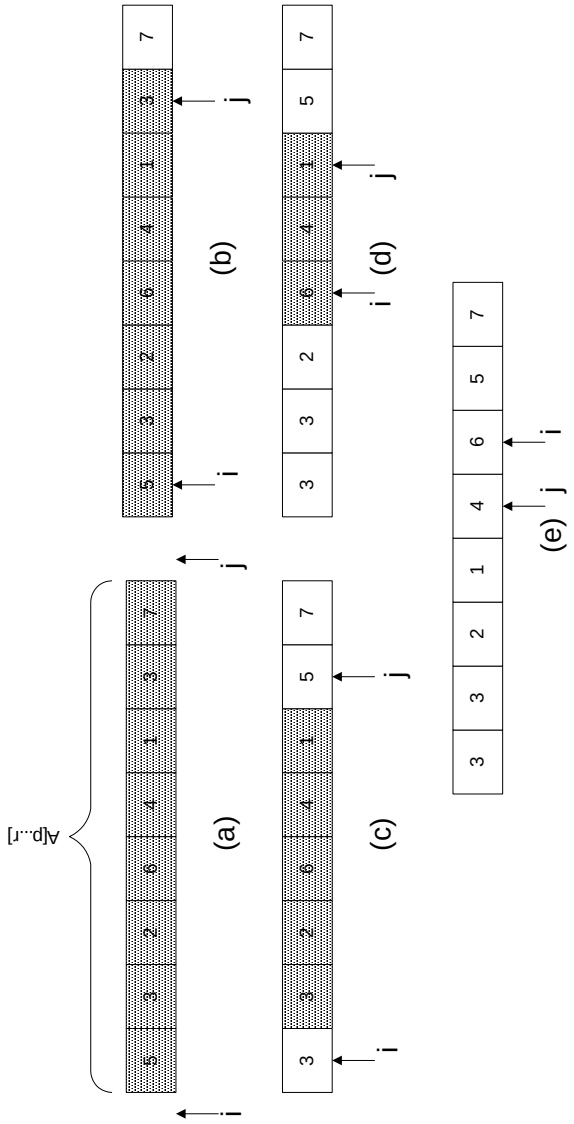
1- A dizisi içerisinde bir eleman seçilir, genellikle dizinin ilk elemanı seçilir ve buna pivot denilir.
2- Seçilen pivot diziden yokmuş gibi geriye kalan elemanlardan pivota eşit veya büyük olanlar dizi içerisinde sağ tarafa doğru kaydırılırlar ve küçük olanlar da sola doğru kaydırılırlar. Bu şekilde A[p..q] ve A[q+1..r] şeklinde iki tane dizi elde edilir ve bu alt diziler sıralı değildir.
3- A dizisi tekrar düzenlenir ve

   A=A[p..q]||x||A[q+2..r]

   şeklini alır. || operatörün sağında kalan ve solunda kalan diziler birbirine eklenir. Pivot eğer dizi sıralı olsaydı olacağı yere gider.
4- Özyinelemeli olarak A[p..q] ve A[q+1..r] dizileri sıralanır.
   Böl algoritması her seferinde kendisine parametre olarak gelen dizinin iki parçaya bölünmesi için ilk elemanını pivot olarak seçer ve ondan sonra iki tane değişken kullanarak biri sağdan sola ve diğeri de soldan sağa hareket edecek şekilde değerleri değiştirilir. Sağdan sola giden endeks ilk pivottan küçük eleman gördüğü yerde durur ve ondan sonra soldan sağa giden endekste ilk pivottan büyük elemanı gördüğü yerde durur. Bu iki eleman yer değiştirir ve bu işleme bu iki endeks birbirine ulaşıncaya kadar veya soldan sağa giden endeks sağdan sola gelen endeksin sağına geçinceye kadar devam eder. Sağdan sola gelen endeks bölme noktası olarak döndürülür.

HızlıSırala algoritmasının performansının incelenebilmesi için bu algoritmanın mertebesi hesaplanır ve mertebe hesabında üç durum

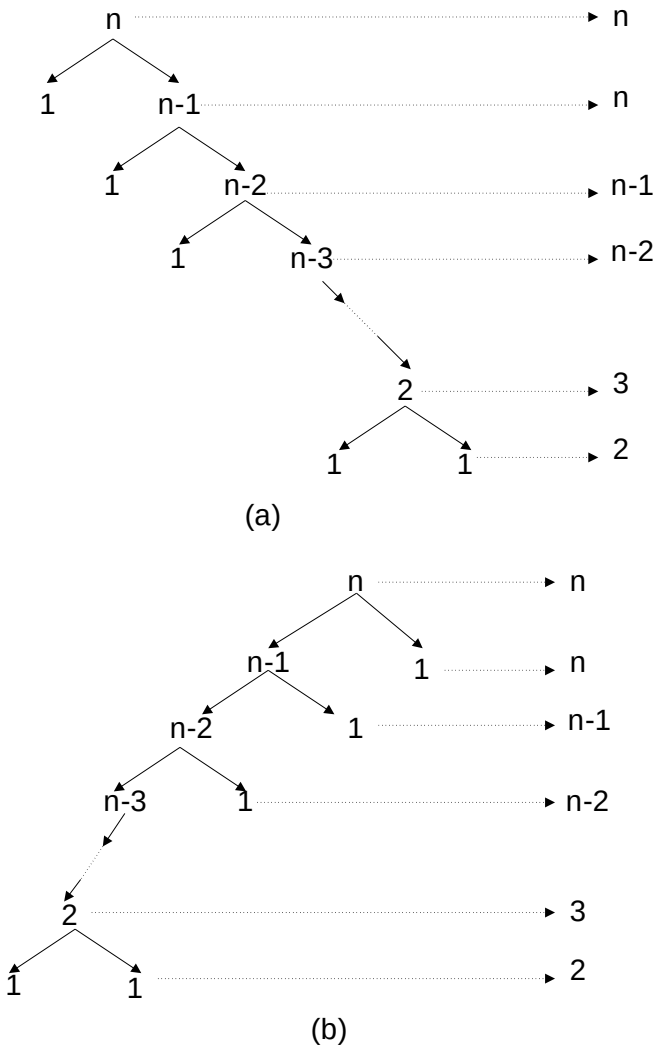
vardır : en iyi durum, en kötü durum ve ortalama durum. HızlıSırala algoritmasının analizi iki farklı şekilde yapılabilir.



Şekil 23.7. *Böl algoritmasının dizi üzerinde bir kez çalışmasının şekilsel ifadesi.*

I.YOL

En Kötü Durum: Eğer HızlıSırala algoritmasına gelen A dizisi eğer sıralı ise, en kötü durum oluşur. Çünkü ilk eleman pivot olarak seçildiğinden, dizi iki parçaya bölündüğünde parçaların birinde bir eleman varken, diğerinde ise n-1 tane eleman olur. Bu durumlar Şekil 23.8' de görülmektedir.



Şekil 23.8. *HızlıSırala algoritmasının en kötü durumları. (a) Dizi küçükten büyüğe doğru sıralı, (b) Dizi büyükten küçüğe doğru sıralıdır.*

Bu iki durum göz önüne alınarak algoritmanın zaman bağıntısı

$$T(n)=T(n-1)+T(1)+\Theta(n)$$

olur ve $T(1)= \Theta(1)$ olarak kabul edilir. Bu bağıntı iteratif olarak çözülecek olursa,

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= \sum_{k=1}^{n} \Theta(k) \\ &= \Theta\left(\sum_{k=1}^{n} k \right) \\ &= \Theta(n^2) \end{aligned}$$

sonucu elde edilir. Bu durum bir sıralama algoritması için iyi bir sonuç değildir.

En İyi Durum: HızlıSırala algoritmasının en iyi durumda algoritmanın mertebe bağıntısı

$$T(n)=2T(n/2)+\Theta(n)$$

olur. Master teoremi kullanılarak bu durumda $T(n)= \Theta(n\lg n)$ olduğu rahatlıkla gösterilebilir.

HızlıSırala algoritmasının mertebesini elde etmek için oluşabilecek durumların hepsinin göz önüne alınması ve kaç durum incelendiyse, bunun ortalaması gerekir. A gibi bir dizi sıralanırken n-1 tane farklı durum oluşabilir. Bundan dolayı

$$T(n) = \max_{1 \le q \le n-1} \left(T(q) + T(n-q) \right) + \Theta(n)$$

zaman bağıntısı elde edilir. T(n) bağıntısının cn^2 gibi bir terimden küçük olduğu kabul edilebilir ve c bir reel sabittir. Bu kabul yerine yazıldığında

$$T(n) \le \max_{1 \le q \le n-1} \left(cq^2 + c(n-q)^2 \right) + \Theta(n)$$

$$= c. \max_{1 \le q \le n-1} (q^2 + (n-q)^2) + \Theta(n)$$

elde edilir. $q^2+(n-q)^2$ terimi maksimum değerine $1 \le q \le n-1$ aralığının uç noktaları üzerinde kavuşur, çünkü terimin q' ye göre ikinci türevi pozitiftir. Bunun anlamı $\max_{1 \le q \le n-1} q^2+(n-q)^2 \le 1+(n-1)^2 = n^2-2(n-1)$ olur. Algoritmanın performansının tam olarak anlaşılması için ortalama durum mertebesinin elde edilmesi gerekir ve n tane durum olduğundan

$$T(n) = \frac{1}{n} \left(T(1) + T(n-1) + \sum_{q=1}^{n-1} \left(T(q) + T(n-q) \right) \right) + \Theta(n)$$

bağıntısı elde edilir. q değişkeninin değerinin dağılımı düzenlidir (uniform). $T(1)=\Theta(1)$ ve $T(n-1)=\Theta(n^2)$ olur ve en kötü durum analizinden

$$\frac{1}{n} \left(T(1) + T(n-1) \right) = \frac{1}{n} \left(\Theta(1) + O(n^2) \right)$$

$$= O(n)$$

elde edilir. Buradan $\Theta(n)$ terimi $\frac{1}{n} \left(T(1) + T(n-1) \right)$ terimini absorbe eder ve bağıntı tekrar yazılacak olursa,

$$T(n) = \frac{1}{n} \sum_{q=1}^{n-1} \left(T(q) + T(n-q) \right) + \Theta(n)$$

bağıntısı elde edilir. HızlıSırala algoritmasında A[p..q] ve A[q+1..r] dizilerinin içerebileceği eleman sayıları

A[p..q]	A[q+1..r]
1	n-1
2	n-2
3	n-3
.	.
.	.
.	.
n-2	2
n-1	1

şeklinde olur. Dikkat edilirse, A dizisi sıralanırken T(1)+T(n-1), T(2)+T(n-2), ... ,T(k)+T(n-k) toplamlarının her biri iki kez oluşur. Bundan dolayı T(n) bağıntısı

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n)$$

şeklini alır. a>0 ve b>0 sabitleri için çözüm T(n)≤anlgn+b şeklinde bu bağıntının bir çözümünün olduğu kabul edilsin. a ve b sabitleri öyle büyük seçilebilirler ki anlgn+b değeri T(1)' den büyük olur. Bu bilgiler ışığında tekrarlı bağıntı

$$\begin{aligned} T(n) &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n) \\ &\le \frac{2}{n}\sum_{k=1}^{n-1} \left(ak \lg k + b\right) + \Theta(n) \\ &= \frac{2a}{n}\sum_{k=1}^{n-1} k \lg k + \frac{2b}{n}(n-1) + \Theta(n) \end{aligned}$$

şeklini alır. Burada toplam sembolünün üst sınırı

$$\sum_{k=1}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

şeklinde olur. Bu sınır kullanıldığında

$$T(n) \le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \frac{2b}{n}(n-1) + \Theta(n)$$

$$\le an \lg n - \frac{a}{4}n + 2b + \Theta(n)$$

$$= an \lg n + b + \left(\Theta(n) + b - \frac{a}{4}n\right)$$

$$\le an \lg n + b$$

olduğu kesinleşir. Daha sıkı bir üst sınır elde etmek için $\sum_{k=1}^{n-1} k \lg k$ teriminin üst sınırının belirlenmesi gerekir. Her terim en fazla nlgn olduğundan

$$\sum_{k=1}^{n-1} k \lg k \le n^2 \lg n$$

olur. Bu da yeterli değildir ve bunun yerine $\sum_{k=1}^{n-1} k \lg k$ toplamı iki terim şeklinde yazılabilir.

$$\sum_{k=1}^{n-1} k \lg k = \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k$$

Sağdaki ilk toplamın üst sınırı lg(n/2)=lgn-1 olur. Buradan ikinci terimin de lgn tarafından sınırlandığı ortaya çıkar.

$$\sum_{k=1}^{n-1} k \lg k \le (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

$$= \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k$$

$$\le \frac{1}{2} n(n-1) \lg n - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2}$$

$$\le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

II.YOL

HızlıSırala algoritmasının analizi için T(1)=T(0)=0 olur. T(n) bağıntısı

$$T(n) = \frac{1}{n} \sum_{q=1}^{n} \left(T(q) + T(n-q)\right) + cn$$

$$= \frac{1}{n}\left(\sum_{q=1}^{n} T(q) + \sum_{j=0}^{n-1} T(j)\right) + cn$$

$$= \frac{1}{n}\left(\sum_{q=1}^{n-1} T(q) + \sum_{j=1}^{n-1} T(j) + T(n)\right) + cn$$

$$= \frac{1}{n}\left(2\sum_{q=1}^{n-1} T(q) + T(n)\right) + cn$$

şeklini alır veya

$$(n-1)T(n) = 2\sum_{q=1}^{n-1} T(q) + cn^2, \quad 1 < c < 2$$

olur. n yerine (n+1) yazıldığında

$$nT(n+1) = 2\sum_{q=1}^{n-1} T(q) + c(n+1)^2$$

$$(n-1)T(n) = 2\sum_{q=1}^{n-1} T(q) + cn^2$$

elde edilir. Bu iki eşitlik taraf tarafa çıkarıldığında

nT(n+1)-(n-1)T(n)=2T(n)+c(2n+1) veya
nT(n+1)=(n+1)T(n)+c(2n+1)

tekrarlı bağıntısı elde edilir. Bu bağıntı tekrar düzenlendiğinde

$$\frac{T(n+1)}{n+1} = \frac{T(n)}{n} + c\frac{2n+1}{n(n+1)}$$

bağıntısı elde edilir ve iteratif olarak bu bağıntı çözüldüğünde

$$\frac{T(n+1)}{n+1} = \frac{T(n)}{n} + c\left[\frac{1}{n+1} + \frac{1}{n}\right]$$
$$\frac{T(n)}{n} = \frac{T(n-1)}{n-1} + c\left[\frac{1}{n} + \frac{1}{n-1}\right]$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\frac{T(2)}{2} = \frac{T(1)}{1} + c\left[\frac{1}{2} + \frac{1}{1}\right]$$

eşitlikleri elde edilir ve bu eşitliklerden

$$\frac{T(n+1)}{n+1} = c\left[\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{2}\right) + \left(\frac{1}{n} + \frac{1}{n-1} + \ldots + 1\right)\right]$$
$$= c\left[H_{n+1} - 1 + H_{n+1} - \frac{1}{n+1}\right]$$
$$= 2cH_{n+1} - c\frac{n+2}{n+1}$$

eşitliği elde edilir. Böylece

$T(n)=scnH_n-c(n+1)\approx 2cnlnn-c(n+1)=O(nlnn)$

olur. HızlıSırala algoritmasının zaman bağıntısı

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n)$$

üzerinde değişiklik yapılarak, ayrık uzaydan sürekli uzaya geçiş yapıldığında yeni bağıntı

$$T(n) = \frac{2}{x}\int_{0}^{x} T(y)dy + x$$

şeklini alır ve bu bağıntı x ile çarpıldıktan sonra x' e göre türev alındığında

$$xT'(x)-T(x)=2x$$

olur. Bu bağıntıda eşitliğin her iki tarafı x^2' ye bölündüğünde

$$\frac{T'(x)}{x} - \frac{T(x)}{x^2} = \frac{2}{x}$$

olur ve düzenlendiğinde

$$\frac{d}{dx}\left(\frac{T(x)}{x}\right) = \frac{2}{x}$$

olur. Eşitliğin her iki tarafı entegre edildiğinde $T(x)/x=2\log x+c$ sonucu elde edilir. Buradan da $T(n)=\Theta(n\lg n)$ olur.

Örnek 23.7

Diğer bir problemde S gibi bir kümede k. en küçük veya k. en büyük elemanı bulma problemidir. Bu problem, eğer $k=\lceil n/2 \rceil$ ise, işlem S kümesinin medyanını bulma işlemine dönüşür.

Elemanlar sıralandıktan sonra v pivot elemanı olduğundan n sayısı pivot değerinden büyük olduğu kabul edilsin. Geriye kalan elemanlar S_1 ve S_2 kümelerine atılsınlar. S_1 kümesindeki elemanların v' den büyük olmadığı bilinmektedir ve S_2 kümesindeki elemanlar da v' den küçük olmayan elemanlar olsunlar. Eğer $k \leq |S_1|$ ise, k. en küçük eleman özyinelemeli olarak S_1 içerisinde aranır. Eğer $k=|S_1|+1$ ise, pivot k. en küçük elemandır. Diğer durumda k. en küçük eleman S_2 kümesi içerisinde $(k-|S_1|-1)$. en küçük elemandır. Bu problem ile HızlıSırala arasındaki fark; HızlıSırala her seferinde iki tane alt problem kullanırken, bu problemde sadece bir tane alt problem kullanılır ve diğeri atılır.

Lineer bir algoritma elde etmek için kullanılan alt problemin bütün problemin bir oranı olması gerekir. Bütün problemin birkaç eleman eksiği olmamalıdır. Bunun için pivotun doğru olarak belirlenmesi gerekir ve bunun için ekstra biraz zaman harcanması gerekir.

HızlıSırala algoritmasında iyi bir pivot seçmek için üç tane eleman seçilir ve bunların medyanı pivot olarak seçilir. Daha iyi bir pivotu belirlemek için O(n/logn) tane eleman seçilir ve heapsort ile bu elemanlar O(n) zamanda sıralanır. En kötü durumda bunun çalışmaması için seçilen O(n/logn) sayının hepsi en küçük olan sayılar olabilir. Bu durumda pivot [n-O(n/logn)]. eleman olur ve bu n sayısının bir sabit oranı olmaz.

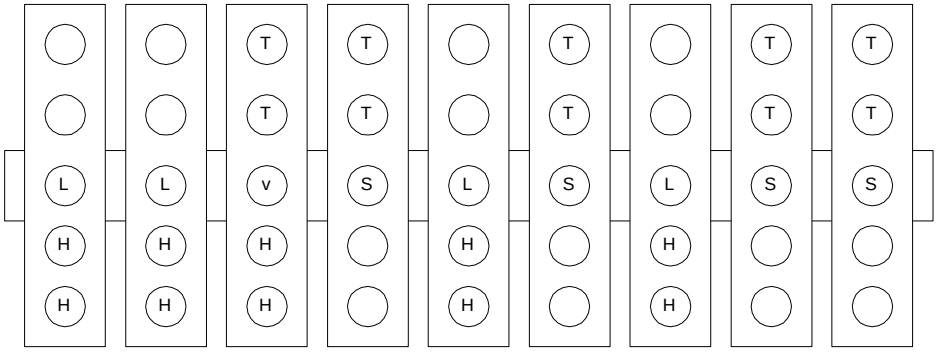
Daha etkili bir en kötü durum belirlemek için gelişigüzel elemanların medyanını bulmak yerine medyan örneklerinin medyanı bulunarak işlem yapılabilir. Bu işlemi gerçekleştiren algoritma Algoritma 23.8' de görülmektedir.

Bu şekilde elde edilen pivot ile O(n) seçme işlemi yapılır. n sayısı 5' e bölünebiliyorsa, ekstra eleman kalmaz. n/5 sayısı tek olduğu kabul edildiğinde M tek sayıda eleman içerir. n=10k+5 olduğu kabul edilsin ve bütün elemanların farklı olduğu kabul edilsin (Şekil 23.9).

Algoritma 23.8. HızlıSeç(S)

▹P parametresi sıralanacak diziyi, p bu dizinin ilk elemanın endeksini ve r ise son elemanın endeksini gösterir.

1- n elemanı $\lfloor n/5 \rfloor$ gruba ayrılır ve her grubun beş tane elemanı olsun (bazı durumlarda ekstra 4 eleman ihmal edilebilir)
2- Her grubun medyanı bulunur. Bunun sonucunda $\lfloor n/5 \rfloor$ medyandan oluşan M listesi oluşturulur.
3- M listesinin medyanı bulunur. Bu değer pivot v olarak döndürülür.



Şekil 23.9. *Beş elemanlı sıralı diziler.*

9 medyanın medyanı v olmak üzere v' den küçük olanlar S ile ve büyük olanlar ise L ile gösterildi (Şekil 23.9). S' li gruplarda medyandan 2 eleman küçük ve 2 eleman büyük olur. Büyük medyanlardan büyük olanlar H ve küçük medyanlardan küçük olanlar T ile gösterilsin. L ve H tipindeki elemanlar v' den büyük ve T ve S tipindeki elemanlar v'den küçük elemanlardır. n=10k+5 için L tipinde k tane, S tipinde k tane, H tipinde 2k+2 tane ve T tipinde 2k+2 tane eleman vardır. Böylece 3k+2 tane eleman v' den büyük ve 3k+2 tane eleman v' den küçüktür. Bu durumda özyinelemede en fazla 7k+2<0.7n eleman vardır. Pivotu elde etmek için sınırlı zaman

kullanılır. Grupların sıralanması sabit zamanda yapılır ve bu işlem $\lfloor n/5 \rfloor$ kez yapılır. Bu durumda mertebe O(n) olur ve amaç $\lfloor n/5 \rfloor$ grubun medyanını hesaplamaktır. Bunu yapmak için bu grup sıralanır ve ortadaki eleman medyan olarak verilir. Bu işlem

$$O(\lfloor n/5 \rfloor \log \lfloor n/5 \rfloor) = O(n \lg n)$$

olur. Bu durumda, bu çalışmaz. Çözümü yapmak için seçme algoritması $\lfloor n/5 \rfloor$ eleman üzerinde özyinelemeli olarak çalıştırılır.

Teorem 23.3
Beş elemanlı parçaların medyanlarının medyanını kullanan HızlıSeç algoritmasının çalışma zamanı O(n) olur.

İspat
Algoritmanın boyutları 0.7n ve 0.2n olan iki tane özyinelemeli çağırması vardır ve bunun yanında ekstradan yapılan lineer işlem vardır. Eğer $\sum_{i=1}^{k} \alpha_i < 1$ ise,

$$T(n) = \sum_{i=1}^{k} T(\alpha_i n) + On)$$

denkleminin çözümü T(n)=O(n) olur. Bu şekilde ispat tamamlanmış olur ♦

Bin tane sayı içerisinde yüzüncü en küçük sayıyı bulmak ile bu kümenin 100 elemanlık alt kümesi içerisinde onuncu en küçük elemanı bulmak ile birbirine benzer.

n elemanlı bir kümeden s tane elemanı olan S kümesi seçilmiş olsun. Algoritma tarafından kullanılan karşılaştırma sayısını minimize eden sayı δ olsun. $v_1=(ks/n-\delta)$. en küçük ve $v_2=(ks/n+\delta)$. en küçük olan eleman olsun (S kümesi içinde). Açık olarak S kümesi içinde k. en küçük eleman v_1 ile v_2 arasına düşer, böylece 2δ tane elemanı olan bir küme içinde eleman arama problemi kalır. Çok küçük bir olasılıkla eleman v_1 ve v_2 düşmeyebilir.

Bu algoritmanın mertebesini belirlemek için $S=n^{2/3}\log^{1/3}n$ ve $\delta=n^{1/3}\log^{2/3}n$ şeklinde seçildiğinde karşılaştırma sayısı

$$n+k+O(n^{2/3}\log^{1/3}n)$$

olur. Eğer $k>n/2$ ise, simetrik problem düşünülebilir ve (n-k). en büyük eleman aranır.

Analiz işlemi çok kolaydır. Son terim v_1 ve v_2 elemanlarını belirlemek için yapılan iki seçme işlemini temsil eder. Ortalama bölme zamanı

$$n+(n+k+O(n\delta/s))$$

olur ve ikinci terim S içerisinde beklenen v_2' nin rankını temsil eder. Eğer k. en küçük eleman S içerisinde ise, mertebe O(s) olur; eğer değilse, mertebe O(n) olur.