Bilgisayar Bilimi Köşesi

Chris Stephenson ve Ali Nesin*
cs@cs.bilgi.edu.tr, anesin@bilgi.edu.tr

Sayıları Tepeleyerek Sıralamak

Sıralama. Bilgisayar programcılığının en önemli konularından biri sıralamadır. Sıralayamayan (ve çabuk sıralayamayan) programcı düşünülemez. Benim diyen programcı, örneğin, upuzun bir borsa listesini göz açıp kapayıncaya kadar sıralayabilmeli.

Sayıları sıralamak önemli. Ama sayıları sıralamakla iş bitmiyor ki... Sözcükleri de sıralamak gerekebilir. Örneğin bir sözlükte ya da bir telefon rehberinde... Hatta, önce sayıları sonra sözcükleri sıralamak gerekebilir. Örneğin, bir mezunlar derneğinde, mezunlar önce mezun oldukları yıla göre sıralanırlar, ardından her yılın mezunları soyadlarına göre sıralanırlar; aynı yıl mezun olmuş aynı soyadlı kişiler varsa bunlar da daha sonra önadlarına göre sıralanırlar.

Ama biz bu yazıda sadece sayıları sıralamayla ilgileneceğiz.

Hantal Bir Sıralama Yöntemi. Bilgisayar öğrencilerinin çoğunun ilk yazdıkları programlardan biri "Kabarcık Sıralama Programı"dır[1]. Bu program karışık bir liste halinde verilmiş bir sayı dizisini küçükten büyüğe doğru sıraya sokar ve $O(n^2)$ sürede çalışır.

Nitekim n sayıyı ikişer ikişer birbirleriyle karşılaştıracak olursak $\binom{n}{2}$ karşılaştırma yapmak zorundayız, ki bu da $n(n-1)/2 = O(n^2)$'dir[2]. Amacımız daha hızlı bir sıralama algoritması bulmak.

En kötü sıralama programı, verilen n sayının en küçüğünü $n-1$ soru sorarak (yani $n-1$ adımda) bulur. Sonra bu sayıyı atar ve geri kalan sayılara aynı işlemi uygular, ta ki hiç sayı kalmayana dek. Böylece bu program $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 \approx n^2$ adımda verilen n sayıyı sıralamış olur.

Büyük O

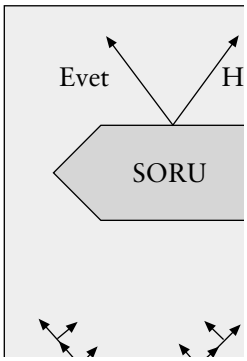
Büyük O (sıfır değil, o harfi, daha dogrusu Yunanca omikron harfi), bir fonksiyonun hangi hızla büyüdüğünü söyler. Örneğin $f(n) = 4n^2 + 2n - 108$ ise, büyük n'ler için, $n^2 \le f(n) \le 5n^2$ eşitsizlikleri, hatta $4n^2 \le f(n) \le 5n^2$ doğrudur. Eğer sabitleri yok sayarsak, $f(n)$ fonksiyonunun n^2 gibi büyüdüğünü söyleyebiliriz.

f ve g iki fonksiyon olsun. Eğer her $x \ge N$ için, $|f(x)| \le C\,|g(x)|$ eşitsizliğini sağlayan N ve C sabitleri varsa, o zaman $f = O(g)$ yazarız. Bu, bir anlamda f, g'den daha hızlı büyümüyor demektir.

Burada eşitlik simgesinin kullanımı doğru olmasa da gelenek böyledir.

Büyük O'ya **Landau simgesi** de denir. Alman Landau bir sayılar kuramcısıdır ve bu kavram ve O simgesi onun buluşudur.

Sıralama İçin Gereken En Az Süre. Sonuca daha hızlı ulaşan bir program yapabilir miyiz? Bu yazıda işte bu soruyu yanıtlayacağız. Ama önce sıralama yapacak bir programın en az $O(n\log n)$ kadar bir süre alması gerektiğini kanıtlayalım:



Teorem. *Elemanları karşılaştırma yöntemine dayalı bir sıralama programı, eğer sıralanacak eleman sayısı n ise,* $O(n\log n)$ *adımdan*[3] *(süreden) önce bitemez.*

* İstanbul Bilgi Üniversitesi

1 İngilizcesi "bubble sort".

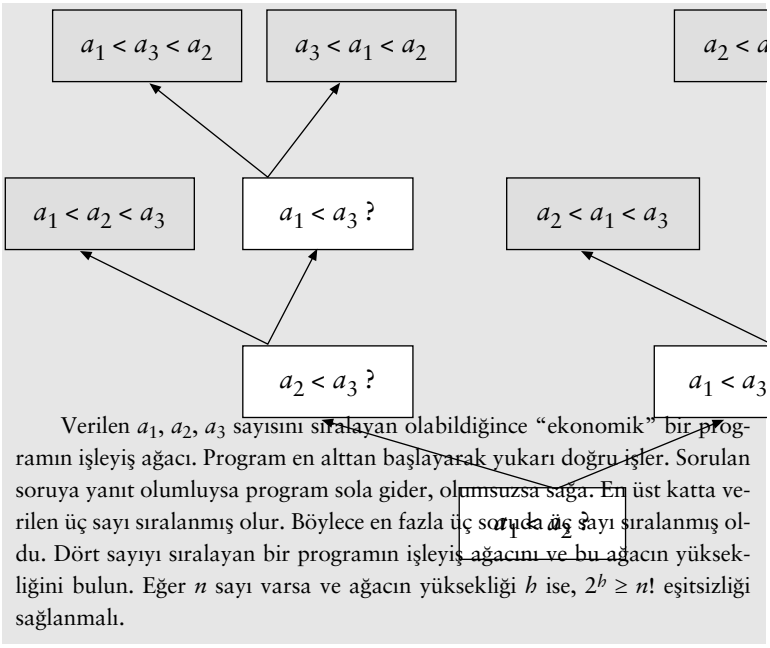
2 Bu "ilkel" programı merak edenler herhangi bir programlama kitabının ilk bölümlerine bakabilir. Ama gerek yok, birazdan çok daha iyisini göreceğiz.

3 Bu yazıda log, $\log_2$ anlamına gelecek. Ama $O(\log n)$ yazılımında, $\log_a x = \log_a b \times \log_b x$ eşitliğinden dolayı, log'un hangi tabanda alındığının hiçbir önemi yoktur.

Kanıt: n nesne $n!$ değişik biçimde sıralanabilir, bunu aklımızda tutalım. Demek ki program bu değişik $n!$ tane sıralamanın herbirini bulabilmeli. Programımızı işlettiğimizde sorduğumuz her soruyu, yandaki şekildeki gibi, aldığımız "evet" ya da "hayır" yanıtına göre, aşağıdan yukarı doğru yükselen bir ağacın ikiye ayrılan bir dalı olarak algılarsak, o zaman programımızın olası tüm işleyişlerini bir ağaç biçiminde görebiliriz. Bu ağacın en kökünden en tepesindeki dala kadar uzanan her yol programın başından sonuna kadar bir işleyişini ve olası bir sıralamayı verir. Demek ki ağaçta en az sıralama sayısı kadar, yani en az $n!$ tane en kökten en tepeye kadar uzanan "en uzun yol" olmalıdır. Programımızın işleyişini en tepesinde en az $n!$ tane yaprağı olan bir ikili ağaç[4] olarak ifade ettik. Dal oluşumu esnasında yapılan işin sabit olduğunu varsayarsak, programımızın maksimum çalışma süresi, ağacın "en uzun yolu"nun uzunluğu olacaktır. Maksimum yol uzunluğu h olan bir ikili ağacın en-uzun-yol sayısının 2^h'den az olacağı açıktır. Demek ki $2^h \geq n!$, yani $h \geq \log(n!)$. Bu, bize elemanların karşılaştırılması ilkesi üzerine kurulmuş sıralama programlarının bitmesi için gereken sürenin altsınırını verir. Daha hesaplar bitmedi ama; Stirling formülüne[5] de ihtiyacımız var. Stirling formülüne göre,

$$\lim_{n\to\infty} \frac{n!}{(n/e)^n \sqrt{2\pi n}} = 1,$$

yani $n! \approx (n/e)^n \sqrt{2\pi n}$ olduğundan, $\log(n!) = O(n\log n + (\log n)/2) = O(n\log n)$, dolayısıyla h en az $n\log n$ olabilir. □



Verilen a_1, a_2, a_3 sayısını sıralayan olabildiğince "ekonomik" bir programın işleyiş ağacı. Program en alttan başlayarak yukarı doğru işler. Sorulan soruya yanıt olumluysa program sola gider, olumsuzsa sağa. En üst katta verilen üç sayı sıralanmış olur. Böylece en fazla üç soruda üç sayı sıralanmış oldu. Dört sayıyı sıralayan bir programın işleyiş ağacını ve bu ağacın yüksekliğini bulun. Eğer n sayı varsa ve ağacın yüksekliği h ise, $2^h \geq n!$ eşitsizliği sağlanmalı.

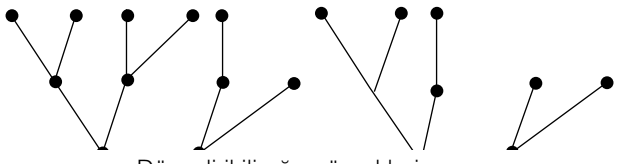
Bu teoremin eleman karşılaştırma yöntemiyle çalışan bir program için geçerli olduğunu unutmayalım. Başka yöntemlerle sıralama daha çabuk yapılabilir. Sayfa 92'deki karede bu yöntemlerden birini bulacaksınız.

Optimum Sürede Çalışan Bir Algoritma Vardır! Eleman karşılaştırma ilkesine dayanan sıralama algoritmalarından biri, bu $O(n\log n)$ optimum çalışma süresine ulaşır.

$O(n\log n)$ sürede çalışan bir algoritma bulmak için doğru soruları sormalıyız elbet, yanlış soru sorduğumuzda algoritmanın $O(n^2)$ sürede biteceğini yukarda gördük. Yazının devamında $O(n\log n)$ sürede çalışan bir sıralama algoritması göreceğiz.

Birçok farklı sıralama algoritması vardır. Bunların içinde **birleştirerek sıralama**[6] ve **tepeleyerek sıralama**[7] adı verilen iki algoritma bu en küçük "büyük O" değerine inebilmiştir. Biz burada tepeleyerek sıralayacağız.

Sıralamamız gereken sayılara a_1, ..., a_n diyelim. Algoritmamızın tam ne yaptığını anlatabilmek için bu diziyi kimileyin aşağıdaki gibi bir "düzenli ikili ağac"ın noktalarının adları olarak görebilmek yararlıdır.



Düzenli ikili ağaç örnekleri

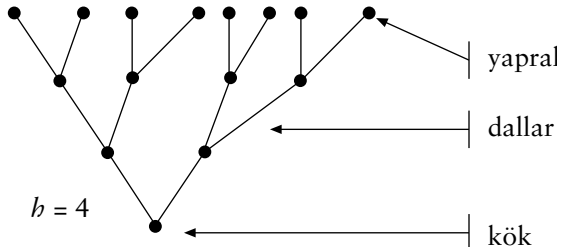
Düzenli İkili Ağaç. Aşağıdaki şekildeki gibi, belli bir h "yüksekliğinde" ve toplam $2^h - 1$ noktalı bir ikili ağaç alalım. Ağacın en altındaki noktaya kök diyelim. En tepedekiler dışında her nokta yukarıya

4 İngilizcesi "binary tree".

5 MD 2003-III, sayfa 33'te de sözünü ettiğimiz bu formülü bir başka sayımızda kanıtlayacağız.

6 İngilizcesi "merge sort".

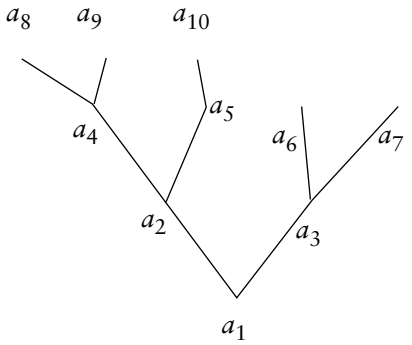
7 İngilizcesi "heap sort".



doğru sağa ve sola olmak üzere iki dala ayrılır. En tepede de tam 2^{h-1} tane dal vardır. Bu en tepedeki dalların ucundaki noktalara "yaprak" diyelim. h yüksekliğinde bir ağaçta 2^{h-1} tane yaprak vardır[8].

Şimdi böyle bir ağacın sağ tarafındaki yapraklarından istediğimiz kadarını silelim. Elde ettiğimiz ağaca **düzenli ikili ağaç**, ya da kısaca **DİA** diyelim. Bir DİA'nın yüksekliği h ve nokta sayısı n ise, $2^{h-1} \le n < 2^h$ yani $h - 1 \le \log n < h$ eşitsizlikleri geçerlidir elbette.

Sıralamamız gereken a_1, a_2, .., a_n dizisini bir DİA'nın noktalarının adları olarak görmek istiyoruz. En alttaki noktadan, yani kökten başlayıp, önce soldan sağa, sonra yukarıya doğru giderek, DİA'nın noktalarını dizimizin sayılarıyla adlandıralım. Aşağıdaki şekilde de görüldüğü üzere her a_i sayısının sol üstünde a_{2i} ve sağ üstünde a_{2i+1} sayıları vardır. Ayrıca her a_i sayısının altında $a_{[i/2]}$ sayısı vardır[9].



Verilen bir diziyi böylece bir DİA'nın noktalarının adları olarak görebiliriz.

Her noktası sayılarla adlandırılmış bir DİA'ya kısaca **ADİA** diyelim. Kolayca görüleceği üzere n noktalı bir ADİA'nın yapraklarının adları $a_{[n/2]+1}$, ..., a_n sayılarıdır. Bu birazdan gerekecek.

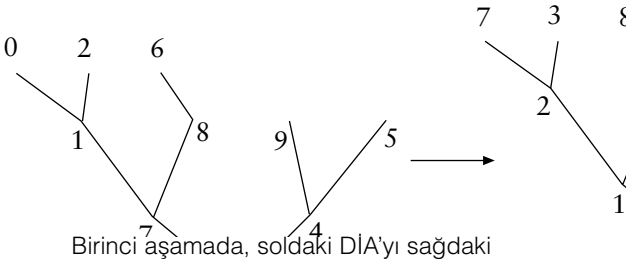
Algoritmamız birbirine çok benzeyen iki aşamadan oluşacak.

Algoritmanın Birinci Aşaması. Birinci aşamada, bir DİA'nın noktalarını, sıralamamız gereken a_1, ..., a_n sayılarıyla adlandırıp öyle bir ADİA elde edeceğiz ki, bu ADİA'nın her dalındaki sayılar aşağıdan yukarıya gidildikçe küçükten büyüğe doğru sıralanmış olacaklar, yani eğer iki sayı arasında kenar varsa, alttaki sayı her zaman üstteki sayıdan daha küçük olacak. Bu tür DİA'lara "sıralı adlandırılmış düzenli ikili ağaç" diyelim ve bunları **SADİA**[10] olarak kısaltalım. Örneğin yukardaki ADİA'nın bir SADİA olması için, $a_1 \le a_2 \le a_4 \le a_8$, $a_1 \le a_2 \le a_4 \le a_9$, $a_1 \le a_2 \le a_5 \le a_{10}$, $a_1 \le a_3 \le a_6$ ve $a_1 \le a_3 \le a_7$ eşitsizlikleri geçerli olmalıdır.

Sıralanacak sayılarımız

3, 7, 4, 1, 8, 9, 5, 0, 2, 6

olsun ve bize bu sırayla verilmiş olsunlar. Bu sayıları aşağıda soldaki şekildeki gibi bir ağaca yerleştirip bir DİA elde edelim. Bu birinci aşamada, O(n) sürede çalışan bir algoritmayla, soldaki DİA'dan hareket ederek sağdaki gibi bir SADİA elde etmek istiyoruz.



Birinci aşamada, soldaki DİA'yı sağdaki SADİA'ya çeviren bir algoritma bulacağız.

Bir SADİA'nın kökündeki sayı, sıralamamız gereken dizinin en küçük sayısıdır zorunlu olarak, başka türlü olamaz. Demek ki sıralanması gereken sayıları bir SADİA'ya dizerek, en küçük sayıyı O(n) sürede bulmuş olacağız. Dizinin en küçük sayısını O(n) sürede bulmak kolay, bunu herkes yapar, ama bunun dışında, SADİA sayesinde, sayılarımızı kısmi olarak da olsa sıralamış olacağız, en azından her daldaki sayıların aşağıdan yukarıya doğru küçükten büyüğe sıralanmış olduklarını bileceğiz ve bunu O(n) gibi kısa bir sürede anlamış olacağız. Nihai sıralama yolunda önemli bir adım...

Elde ettiğimiz SADİA'nın yüksekliğine h diyelim. $2^{h-1} \le n$ ve $h - 1 \le \log n$ eşitsizliklerini unutmayalım, çok gerekecek.

Şimdi bir DİA'yı bir SADİA'ya dönüştürmeyi yukardaki örnek üzerinden öğreneceğiz. DİA'nın

8 Bir ağacı ters çevirirsek tepeye benzer bir şekil elde ederiz. "Tepelemek" ya da İngilizcesiyle "heapify" buradan gelmektedir.

9 $[x]$, x sayısının tam kısmıdır.

10 İngilizcesi "binary heap", yani "ikili tepe".

en tepesinden ve en sağından (6'dan) başlayarak önce sola ve sonra aşağı doğru gidip SADİA'mızı yavaş yavaş (bir anlamda tümevarımla) elde edeceğiz. Okurun algoritmanın işleyişini ağacın üstünden izlemesinde sonsuz yarar vardır.

Sayılarımız 3, 7, 4, 1, 8, 9, 5, 0, 2, 6. On sayı olduğundan $n = 10$. En sağdan (yani $i = 10$'dan) başlayarak ve sola doğru giderek, i-inci sıradaki sayıyı $2i$ ve $(2i+1)$-inci sıradaki sayılarla (eğer o sıralarda sayı varsa) karşılaştıracağız. Uygulamada hep $i = [n/2]$'den başlanır elbet, örneğimizde $i = 5$'ten başlayacağız. Eğer i-inci sıradaki sayı $2i$ ve $(2i+1)$-inci sıradaki sayılardan daha küçükse sorun yok, bir sola kayıp aynı işlemi $(i-1)$-inci sıradaki sayıyla yapacağız. Eğer i-inci sıradaki sayı bu iki sayının birinden ya da her ikisinden de daha büyükse, o zaman i-inci sayıyla $2i$ ve $(2i+1)$-inci sıradaki sayıların en küçüğünün yerini değiştireceğiz. Arkasından bu işlemi i-inci sıradaki sayının gittiği yerle yapacağız ve buna devam edeceğiz, ta ki i-inci sıradaki sayı yerini buluncaya kadar. İşlem bittiğinde aynı işlemi $(i-1)$-inci sıradaki sayıya uygulayacağız.

$i > [n/2]$ ise $2i > n$ olduğundan, yukarda tarif ettiğimiz işlem aslında $i = [n/2] = 5$'ten başlar. Yani DİA'nın en tepesinde bulunan 9, 5, 0, 2, 6 sayıları için yapılacak bir işlem yoktur. Şimdi $i = 5$ olsun:

3, 7, 4, 1, **8**, 9, 5, 0, 2, 6

Beşinci sıradaki 8 sayısını onuncu ve on birinci sıralardaki sayılarla karşılaştıracağız. On birinci sırada sayı olmadığından beşinci sıradaki 8 sayısını onuncu sıralardaki 6'yla karşılaştıracağız. (Yukardaki ağaçta 8'le 6'nın yerlerine bakın.) $8 > 6$ olduğundan bu iki sayının yerlerini değiştirmeliyiz:

3, 7, 4, **1**, 6, 9, 5, 0, 2, 8

dizisini elde ettik. $i = 5$ ile işimiz bitti, çünkü 8 en sona gitti. Şimdi $i = 4$ olsun. Dördüncü sıradaki 1'i sekizinci ve dokuzuncu sıradaki sayılarla, yani 0 ve 2'yle karşılaştıracağız. $1 < 2$ olduğundan, 2'ye dokunmayalım, ama 1'le 0'ın yerini değiştirelim.

3, 7, **4**, 0, 6, 9, 5, 1, 2, 8

1, sekizinci sıraya gitti ve $8 > [n/2]$ olduğundan 1'le de işimiz bitti. Şimdi $i = 3$. Üçüncü sıradaki 4'ü altıncı sıradaki 9'la ve yedinci sıradaki 5'le karşılaştıracağız. 4, her ikisinden de küçük. Hiçbir değişiklik yapmamıza gerek yok. Şimdi $i = 2$.

3, **7**, 4, 0, 6, 9, 5, 1, 2, 8

İkinci sıradaki 7'yi dördüncü sıradaki 0'la ve beşinci sıradaki 6'yla karşılaştıracağız. 7, bu iki sayının her ikisinden de büyük. Olmadı... 7'yle bu iki sayıdan en küçüğü olan 0'ın yerlerini değiştirelim.

3, 0, 4, **7**, 6, 9, 5, 1, 2, 8

7 sayısı daha gerçek yerini bulmadı ama. Dördüncü sıradaki 7'yi sekizinci sıradaki 1 ve dokuzuncu sıradaki 2'yle karşılaştıralım. 7, her ikisinden de büyük, o zaman 7'yle 1'in yerini değiştirelim:

3, 0, 4, 1, 6, 9, 5, 7, 2, 8

Artık 7'yi karşılaştıracağımız sayı kalmadı. Şimdi $i = 1$ olsun. Birinci sıradaki 3'le ikinci sıradaki 0'ı ve üçüncü sıradaki 4'ü karşılaştıralım. $3 > 0$ eşitsizliği oyunbozanlık yapıyor, yerlerini değiştirelim:

0, **3**, 4, 1, 6, 9, 5, 7, 2, 8

3'le daha işimiz bitmedi. İkinci sıradaki 3'ü dördüncü sıradaki 1'le ve beşinci sıradaki 6'yla karşılaştıralım. $3 > 1$ eşitsizliğinden dolayı 3'le 1'in yerini değiştirelim:

0, 1, 4, **3**, 6, 9, 5, 7, 2, 8

3'le işimiz hâlâ bitmedi. Dördüncü sıradaki 3'ü sekizinci sıradaki 7 ve dokuzuncu sıradaki 2'yle karşılaştıralım. $3 < 7$ güzel de, $3 > 2$ güzel değil. 3'le 2'nin yerlerini değiştirelim:

0, 1, 4, 2, 6, 9, 5, 7, 3, 8.

İşte şimdi bir SADİA elde ettik, hem de yukardaki şekilde gösterdiğimiz SADİA'yı elde ettik.

Bu işlem ne kadar sürede biter?

i-inci sıradaki sayıyı hep $2i$ ve $2i+1$ sıradaki sayılarla karşılaştırıyoruz. Bir değiş tokuş yapılmışsa, i-inci sıradaki sayının gittiği yerle aynı işlemi yapıyoruz: Diyelim i-inci sıradaki sayı $2i$-inci sıradaki sayının yerine gitti. Şimdi bu sayıyı $4i$-inci ve $(4i+1)$-inci sıradaki sayılarla karşılaştıracağız. Kolayca görüleceği üzere, eğer $2^j i > n$ ise, bu işlem en fazla $j - 1$ adım sürer. Bu eşitsizliği sağlayan en küçük j tamsayısı $j = [\log(n/i)] + 1 \le \log(n/i) + 1$ eşitsizliğini sağlar. Demek ki i-inci sıradaki sayıyla işimiz en fazla $\log(n/i)$ adımda bitiyor. Madem ki i, n'den[11] 1'e kadar değişiyor, yukardaki algoritmanın işleyiş süresi en fazla

$$\sum_{i=1}^{n} \log(n/i)$$

dir. Biraz hesap yapalım.

$$\sum_{i=1}^{n} \log(n/i) = \log\left(\frac{n^n}{n!}\right) \approx \log\left(\frac{e^n}{\sqrt{2\pi n}}\right) = O(n).$$

(İkinci ilişkide daha önce de kullandığımız Stirling formülünden yararlandık.)

11 Aslında $[n/2]$'den, ama hiç önemi yok.

Algoritmanın İkinci Aşaması. Yukardaki yöntemle O(n) sürede bir SADİA elde ettik:

0, 1, 4, 2, 6, 9, 5, 7, 3, 8.

Bu SADİA'nın kökünü atıp, ki o kökteki sayı dizinin en küçük sayısıdır, onun yerine en tepe ve en sağdaki yaprağı dalıyla birlikte silip yaprağın sayısını köke koyalım:

8, 1, 4, 2, 6, 9, 5, 7, 3.

Bir ADİA elde ederiz. Bu ADİA'nın sayılarının yerlerini değiştirerek yeni bir SADİA elde edeceğiz. Birinci (i = 1) sayı olan 8'den başlayarak, aynen yukardaki gibi, 8'i ikinci ve üçüncü sayılarla karşılaştıracağız. 8, her ikisinden de büyük. Bu sayıların en küçüğü olan 1'le 8'in yerlerini değiştirelim.

1, 8, 4, 2, 6, 9, 5, 7, 3.

Şimdi 8 ikinci sırada. İkinci (i = 2) sıradaki 8'i dördüncü ve beşinci sıradaki sayılarla karşılaştıracağız. 8'le 2'nin yerlerini değiştirmek zorundayız:

1, 2, 4, 8, 6, 9, 5, 7, 3.

Şimdi 8 dördüncü (i = 4) sırada. 8'i sekizinci ve dokuzuncu sıradaki 7 ve 3'le karşılaştıralım. 3'le 8'in yerini değiştireceğiz.

1, 2, 4, 3, 6, 9, 5, 7, 8.

Burada duralım, çünkü yeni bir SADİA elde ettik. Kökteki sayı (1) bu yeni SADİA'nın en küçük, dizimizin de ikinci küçük sayısıdır. 1'i atıp, yerine en tepedeki 8'i koyalım:

8, 2, 4, 3, 6, 9, 5, 7.

Gene aynı işlemleri yapacağız, ta ki sayı kalmayana dek. Aşağıdaki kutuda bu işlemlerin sonucunu bulacaksınız.

İkinci aşamanın çalışma uzunluğunu hesaplayalım şimdi. Eğer dizide k sayı varsa, birinci sayıdan en tepeye kadar çıkmak için en fazla $\log(k)$ adım atarız. Demek ki ikinci algoritma $\log(1) + \log(2) + \ldots + \log(n-1) = \log((n-1)!) \le \log(n!) \approx \log((n/e)^n \sqrt{2\pi n}) = O(n \log n)$ sürede biter.

Birinci aşama O(n) sürede, ikinci aşama O($n$log n) sürede bitti. Demek ki tepeleyerek sıralama toplam algoritması O($n$log n) sürede biter.

2, **8**, 4, 3, 6, 9, 5, 7
2, 3, 4, **8**, 6, 9, 5, 7
2, 3, 4, 7, 6, 9, 5, **8**
8, 3, 4, 7, 6, 9, 5
3, **8**, 4, 7, 6, 9, 5
3, 6, 4, 7, **8**, 9, 5
5, 6, 4, 7, 8, 9
4, 6, **5**, 7, 8, 9
9, 6, 5, 7, 8
5, 6, **9**, 7, 8
8, 6, 9, 7
6, **8**, 9, 7
6, 7, 9, **8**
8, 7, 9
7, **8**, 9
9, 8
8, **9**
9

Program yazarken sık sık verileri (örneğin sayıları) makinada depolamak zorunda kalırız. Bu verileri makina daha sonra genellikle belli bir sırayla kullanacağından, depolanan verilerin bir "öncelik değeri" olur. Öncelik değeri en yüksek veri, ilk önce kullanılacak ve kullanıldıktan sonra gerekirse depodan atılacak ilk veridir. Kimileyin depoya öncelik değerli yeni bir veri eklemek zorunda kalabiliriz.

Önceliği en yüksek veriyi bulmak O(n) zaman alır elbet. Ve depo her değiştiğinde bu işlemi tekrar tekrar yapmak gerekir. Bu da programların çalışmasını yavaşlatır. Ama eğer verileri öncelik değerine göre bir SADİA'da depolarsak, o zaman ilk SADİA'yı yaratmak O(n) zaman alsa da, daha sonra yeni bir verinin eklenmesiyle ya da çıkarılmasıyla değişen depoyu tekrar bir SADİA'ya dönüştürmek yukarda gördüğümüz gibi O(log n) zaman alır.

Sonuç olarak, yarattığımız SADİA bir "veri yapısı"dır ve "öncelik değeri" olan verileri depolarken bize hız kazandırır; bu veri yapısı sayesinde önceliği en yüksek sayı O(n^2) yerine O(n log n) zamanda bulunur.

Böylece, geçen sayıda bahsettiğimiz Karmarkar Karp algoritmasının ihtiyacı olan veri yapısını da bulmuş olduk. ♣

İkili karşılaştırma yöntemine başvurmadan sayıları şöyle de sıralayabiliriz: Sıraya dizilecek

$a_1, a_2, \ldots, a_n$

doğal sayıları verilmiş olsun. Bunların en büyüğü N ise

$b_0, b_1, b_2, \ldots, b_N$

kutularını yaratalım. Bu b kutularının herbirinin değeri başlangıçta 0 olsun. Sonra a'ları teker teker gözden geçirelim. Her i değerine rastladığımızda b_i'nin değerini 1 artıralım. Bunu yapmak n birim süre alır. a'ların sonuna geldiğimizde her b_i bize a dizisinde kaç tane i olduğunu söyler. Örneğin sıraya dizilmesi gereken a sayıları,

3, 0, 0, 5, 7, 7, 3, 0, 2, 2, 3, 0

ise, başlangıçta herbiri 0 olan b'ler a'ların sonuna gelindiğinde şöyle olur:

4, 0, 2, 3, 0, 1, 0, 2.

Şimdi b'lere bakarak a'ları sıraya dizebiliriz:

0, 0, 0, 0, 2, 2, 3, 3, 3, 5, 7, 7.

Bu da N birim süre alır. Demek ki bu program n + N birim sürede biter.

Bu programın önemli bir sorunu var: n sayısı küçük ama N sayısı çok büyük olabilir. Örneğin, sözcükleri sıralayacaksak, 29 harften ve (diyelim) en fazla 15 harften oluşan sözcükleri sayıya dönüştürecek olursak, $N = 29^{15}$ olur, ki bu da çok çok çok büyük bir sayıdır.