Bilgisayar Biliminde kelime yığın, birkaç farklı bağlamda kullanılır. Bir yığın bazen dinamik (yani çalışma zamanı) bellek tahsisi için kullanılan bir bellek alanına atıfta bulunur. Başka bir anlamı ve bu bölümün konusu, kavramsal olarak tam bir ikili ağaç olan bir veri yapısıdır. Yığınlar öncelik kuyruklarını, yığın sıralama algoritmasını ve bazı graf algoritmalarını uygulamak için kullanılır. Yığınlar, ağaçtaki öğelerin bir düzenini koruyan ikili arama ağaçları gibi birazdır. Ancak, bir yığın öğelerinin tam bir düzenini korumaz. Bu, bir yığının nasıl kullanılabileceğine dair bazı sonuçlara sahiptir. Bölüm Hedefleri Bu bölümün sonunda aşağıdaki soruları cevaplayabileceksiniz:

• Bir yığın nedir ve nasıl kullanılır?

• Bir yığından öğe eklemenin ve silmenin hesaplama karmaşıklığı nedir?

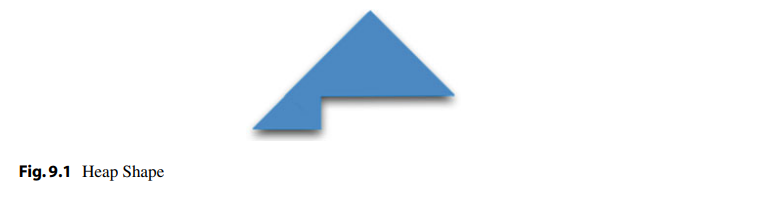
• Bir yığından öğeleri aramak için bir yığın kullanır mısınız yoksa kullanmaz mısınız?

• Ne zaman bir yığın kullanırsınız?

• Yığın sıralama algoritmasında, en büyük-en-üstte yığının avantajlı olmasının nedeni nedir?

**Ana Fikirler**

Yığınları anlamak için bir tanımla başlayacağız. En büyük-en-üstte bir yığın, her düğümün ≥ tüm çocuklarından büyük veya eşit olduğu tam sıralı bir ağaçtır (eğer varsa). Bir örnek, bu tanımı açıklamaya yardımcı olacaktır. Kavramsal olarak, bir yığın, tüm seviyelerde dolu olan ancak muhtemelen en alt seviye hariç (bu seviye soldan sağa doldurulur) bir ağaçtır. Genel şekli Fig. şeklinde alır.

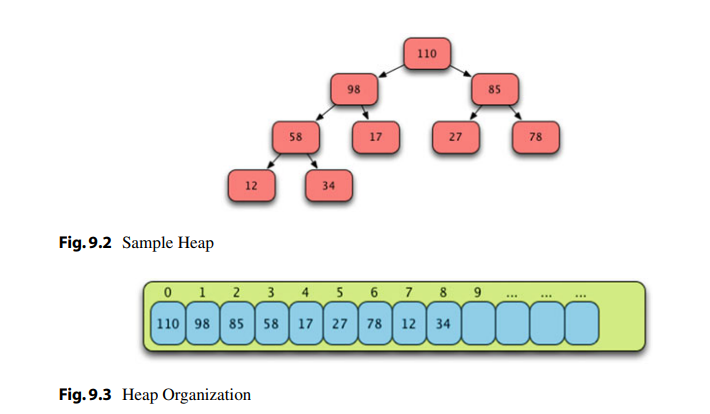


Kavramsal olarak, bir yığın bir ağaçtır, ancak yığınlar genellikle ağaçlar olarak depolanmazlar. Bir tam ağaç, en alt seviye hariç tüm seviyelerde dolu olan bir ağaçtır ve en alt seviye soldan sağa doldurulur. Yığınlar tam ağaçlar olduğu için bir dizi içinde depolanabilirler. Bir örnek, yığınları ve tam özelliği daha iyi anlamanıza yardımcı olacaktır. Bir dizi içinde kök düğümünün endeks 0'da depolandığı en üstte en büyük bir yığını ele alalım. Kavramsal olarak, Şekil 9.2, tamsayılar içeren bir yığındır. Bu kavramsal versiyonda veriler, kök düğümden başlayarak ağacı seviye seviye dolaşarak bir dizi içinde depolanır. Şekil 9.2'deki kavramsal yığın, Şekil 9.3'te düzenlenmiş olarak bir dizide depolanacaktır.

**Bir yığının sergilediği iki özellik vardır:**

• Yığın Yapısı Özelliği: Yığının öğeleri tam sıralı bir ağaç oluşturur.

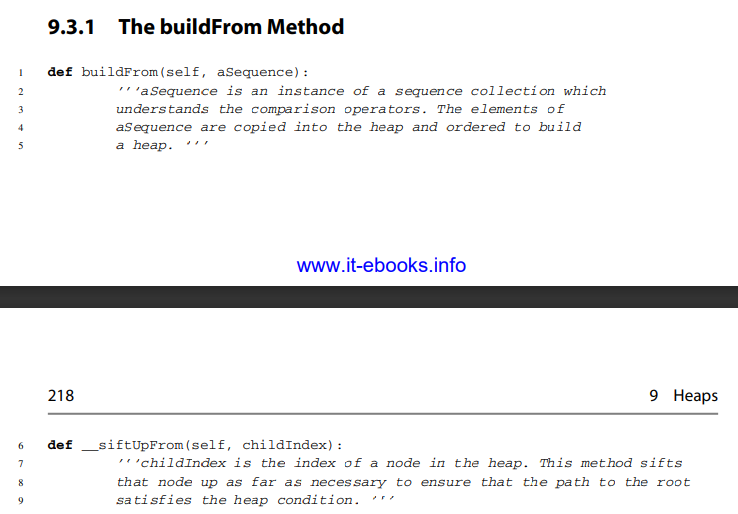
• Yığın Sıra Özelliği: Her üst düğüm ≥ tüm çocukları (tüm torunları dahil) şeklinde ifade edilir.

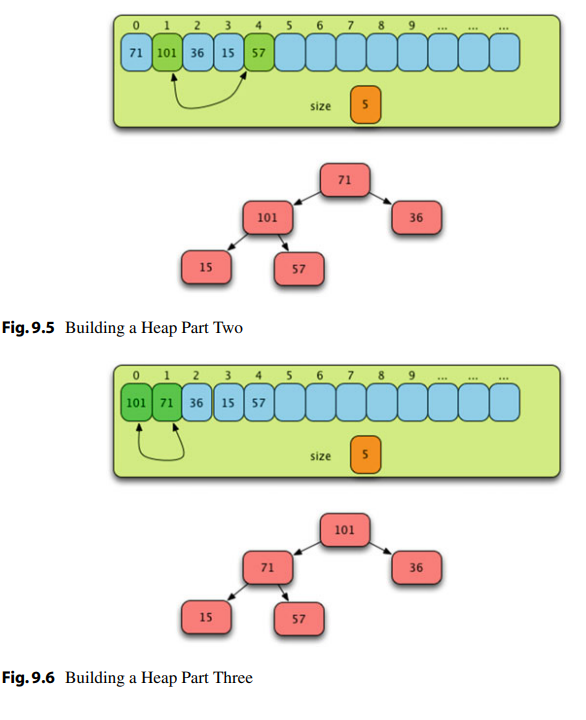
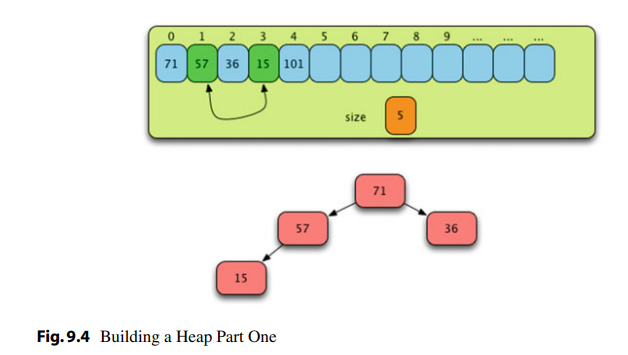
Şekil 9.2'deki yığın bu iki özelliği de korur. Şekil 9.3'teki bu yığının dizi uygulaması da bu özellikleri korur. Özelliklerin dizi uygulamasında nasıl korunduğunu görmek için konumu hesaplayabilmemiz gerekmektedir.

Çocukların ve ebeveynlerin endeksleri hesaplanabilir. Dizinin herhangi bir elemanının çocukları, ebeveynin endeksinden hesaplanabilir. solÇocukEndeksi = 2 ∗ ebeveynEndeksi + 1 sağÇocukEndeksi = 2 ∗ ebeveynEndeksi + 2 Bu formülleri Şekil 9.3 üzerinde kullanarak kök düğümün (yani endeks 0) çocuklarının 98 (endeks 1'de) ve 85 (endeks 2'de) olduğunu görebiliriz. Benzer şekilde, 85'in çocukları 27 ve 78 değerleri olan endeks 5 ve 6'da bulunur, ki bunların kavramsal modeldeki çocuklarla aynı olduğunu doğrulayabiliriz. Tabii ki, her düğümün bir çocuğu veya hatta iki çocuğu olmayabilir. Hesaplanan solÇocukEndeksi veya sağÇocukEndeksi, yığında bulunan değerlerin sayısından büyük veya eşitse, o düğüm bir yaprak düğümdür. Ayrıca, diğer yöne gitmek de mümkündür. Bir çocuğun endeksini verildiğinde, ebeveynin nerede olduğunu bulabiliriz. ebeveynEndeksi = (çocukEndeksi − 1)//2 Önceki formüldeki // tamsayı bölümünü temsil eder. Bu, sonucun her zaman bir tamsayı olacağı anlamına gelir. Eğer kesirli bir kısım varsa, bir sonraki daha düşük tamsayıya yuvarlarız. Bu nedenle, Şekil 9.3'teki 34'ün ebeveyninin endeksi aşağıdaki gibi hesaplanır: ebeveynEndeksi = (8 − 1)//2 = 3 Dizideki kavramsal modele danıştığımızda, 3 endeksindeki değer olan 58'in gerçekten 34'ün ebeveyni olduğunu görüyoruz. Bir yığında her düğümün bir ebeveyninin olmadığı da belirtilmelidir. Özellikle, endeks 0'daki kök düğümünün bir ebeveyni yoktur. Bir yığında diğer tüm düğümlerin ebeveynleri vardır.

**9.3 Yığın Oluşturma**

Bir yığının nasıl göründüğünü gördükten sonra, bir yığın oluşturmayı inceleyeceğiz. Yığınlar en üstte en büyük veya en üstte en küçük olarak oluşturulabilir. En üstte en büyük bir yığın oluşturacağız. Bir Yığın sınıfı, bir yığını oluşturmak için gerekli olan veri ve yöntemleri kapsayacaktır. Yığın nesneleri, yığında şu anda depolanan öğelerin bir listesini ve sayısını içerir. Bu sayıyı yığının boyutu olarak adlandıracağız. Verileri kapsamak için, bir değer dizisi alacak ve ondan bir yığın oluşturacak bir yönteme ihtiyacımız olacak. Bu yönteme buildFrom diyeceğiz. Bir özel yöntem de faydalı olacaktır. buildFrom yöntemi, her ardışık öğeyi dizinin içindeki doğru konumuna yerleştirmek için \_siftUpFrom yöntemini çağıracaktır.

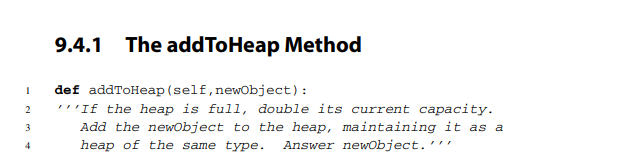


buildFrom yöntemine geçirilen değer dizisi yığına kopyalanır. Daha sonra, listedeki her ardışık değer, yığında son konumuna kadar yukarı süzülür. Değerlerin [71, 15, 36, 57, 101] listesi göz önüne alındığında, her aşamada oluşan yığın gösterilerek bu süreci izleyeceğiz. İlk olarak, liste burada verilen sırayla yığına kopyalanır. Daha sonra, her ardışık öğede siftUpFrom çağrılır. Başlamak için, siftUpFrom, bu durumda 57 olan ikinci öğede çağrılır. Yığının köküne siftUpFrom çağrılsaydı hiçbir etkisi olmazdı. Normalde, düğümün ebeveyn endeksi hesaplanır. Ebeveyn zaten diğer çocuktan (varsa) büyük olacaktır. Mevcut çocuk endeksindeki değer, ebeveyn endeksindeki değerden büyükse, ikisi değiştirilir ve süreç tekrarlanır. Bu süreç, gerekli olan kadar çok kez tekrarlanır: ya kök düğüme ulaşılıncaya kadar (yani listedeki endeks 0) ya da yeni düğüm yığın özelliğini korumak için uygun konumda olana kadar. Bazı hareketin ilk gerçekleştiği zaman, 57 yığına eklendiğinde olur. 57, son konumuna ulaşmak için 15 ile değiştirilir ve Şekil 9.4'teki yığını elde ederiz. Listenin ilk dört öğesi artık bir yığın oluşturuyor. Ancak, 101 henüz son konumunda değil. Yığında onu son konumuna getirmek için yukarı süzmeliyiz. Yığının kavramsal görünümüne (yani ağaca) bakarak, 101'in 57 içeren düğümün bir çocuğu olduğunu görebilirsiniz. Bu açıkça yığın özelliğine aykırıdır. Bu nedenle, 101 ve 57, Şekil 9.5'te gösterildiği gibi 101'i süzmek için değiştirilir. Kavramsal modeli incelemeden, 101 içeren düğümün ebeveynini yine de hesaplayabilirsiniz. 9.5'in ilk kısmında, 101 endeks 4'teyken, ebeveyn endeksi aşağıdaki gibi hesaplanmıştır. ebeveynEndeksi = (4 − 1)//2 = 1 Bu nedenle, 101, yukarıdaki ilk bölümdeki 57 ile karşılaştırılır. Sonra, ikinci bölümde 101'in 57'den büyük olduğu için değişim yapılır. Ancak, 101 hala değil.

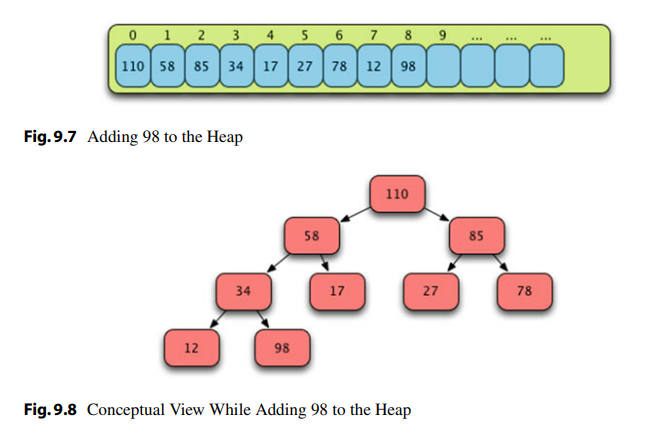
101'i 71 ile karşılaştırıyoruz ve iki öğeyi değiştiriyoruz. Bu, 101'in şimdi yığının köküne (yani endeks 0) ulaştığı son yukarı süzülme iterasyonudur. İki değeri değiştirdikten sonra Şekil 9.6'daki yığına ulaşırız.

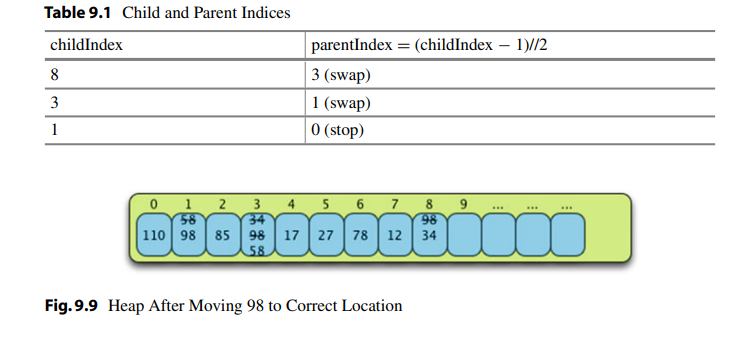
9.4 Yığın Sıralama Algoritması Versiyon 1

Yığınların iki temel işlemi vardır. Bir değeri bir yığına ekleyebilirsiniz. Ayrıca, bir yığının en büyük değerini silebilir ve alabilirsiniz, eğer yığın en üstte en büyük bir yığın ise. Bu iki işlemi veya bunların varyasyonlarını kullanarak, bir değerler listesiyle bir yığın oluşturarak ve ardından değerleri birer birer çıkararak bir sıralama algoritması tasarlayabiliriz.



Bu yeni yöntem, \_\_siftUpFrom özel yöntemini, yeni öğeyi yığın içindeki nihai konumuna ulaştırmak için kullanabilir. Faz I'in Versiyon 1'i addToHeap'i n kez çağırır. Bu, O(n log n) karmaşıklığına yol açar. Faz I'in belirli adımları şunları içerir: Yığının kapasitesini gerektiğinde iki katına çıkarın. data[size] = newObject \_\_siftUpFrom(size) size+=1. Görebileceğiniz gibi, \_\_siftUpFrom, yığının her bir öğesi için bir kez olmak üzere n kez çağrılacaktır. \_\_siftUpFrom her çağrıldığında, yığın 1 öğe ile büyümüş olacaktır. Pass #9'dan hemen önceki Şekil 9.7'deki yığına bir göz atın. 98'i yığın içindeki doğru konumuna yukarı süzmek üzereyiz. Kavramsal olarak, Şekil 9.8'de gösterilen yığın resmine sahibiz. 98'i doğru konuma taşımak için, Tablo 9.1'de gösterildiği gibi çocuk endekslerinden ebeveyn endeksini hesaplamamız gerekmektedir



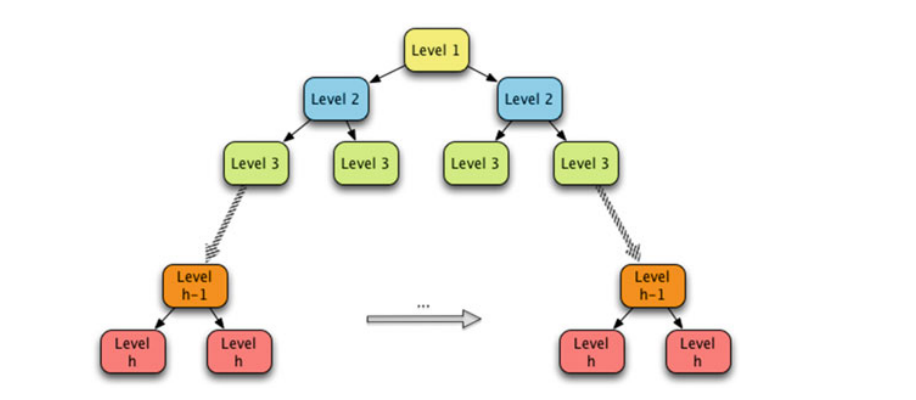


Yığın listesinde 98'in yukarı süzülmesi, son konumuna ulaşmadan önce iki değişimle sonuçlanır. Şekil 9.9, 98'in endeks 3'teki 34 ile değiştirildiğini gösterir. Ardından, endeks 1'deki 58 ile tekrar değiştirilir. Bu noktada, 101 98'den büyük olduğu için daha fazla değişim yapılmaz. 98, yığın içindeki uygun konumuna ulaşmıştır.

**9.5 Faz 1'in Versiyon 1'inin Analizi**

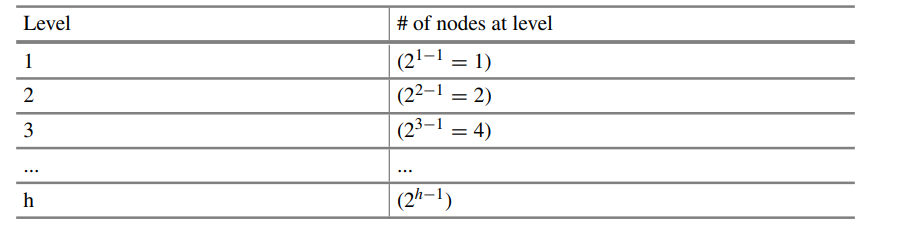
Faz I'in versiyon 1'de alınan yaklaşım yavaş olduğu gibi, göreceğiz. Mükemmel tam ikili ağacı düşünün. Tüm seviyelerde tamamen dolu olan ve Fig. 9.10'da gösterildiği gibi h seviyeye sahip olan bir ağacı düşünün.

Tablo 9.2'de gösterildiği gibi, seviye sayısı ile yığında bulunan öğelerin sayısı arasındaki ilişkiyi düşünün.

**Tablo 9.10** Mükemmel bir ikili ağaç.

**222 9 Yığınlar**

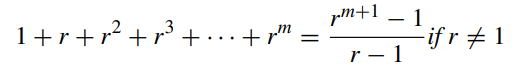
**Tablo 9.2** Yığın seviyelerine karşı Yığın boyutu.



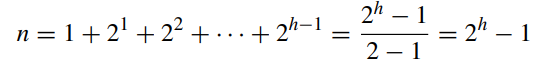
Yığında n öğe bulunan bir yığın için, n değeri yığının ağacındaki her seviyedeki tüm düğümlerin toplanmasıyla hesaplanabilir. Argümanımızı basitleştirmek için yığının dolu bir ikili ağaç olduğunu varsayacağız.



Bu, bir geometrik dizinin toplamıdır. Bir geometrik dizinin toplamı şu şekilde hesaplanabilir.



Yukarıdaki denklemimize bu formülü uygulayarak h seviyeli tam bir ikili ağaçta (yani tam bir ikili yığın) bulunan düğümlerin sayısı aşağıdaki formülle verilir.

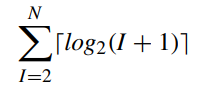


Bu, n + 1 = 2h demektir. Bu denklemi h için çözebiliriz. Bunu yaparsak:



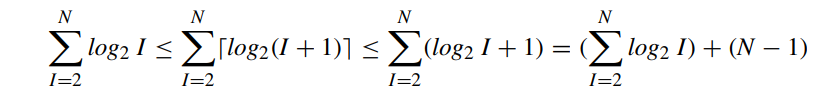
Yukarıdaki parantezler tavan operatörüdür ve basitçe en yakın daha büyük tamsayıya yuvarlamamız gerektiği anlamına gelir. Yuvarlamak, her yığın ağacının tamamen dolu olmadığını dikkate alır, bu nedenle yuvarlamadığımızda h için bize bir tamsayı vermeyen bazı n değerleri olabilir. Aşağıdaki eşitsizlik, heapsort algoritmasının Faz 1'inin hesaplama karmaşıklığını belirlemede yararlı olacaktır.



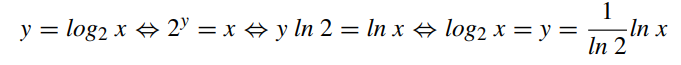
Şimdiye kadar tam bir ikili ağacın yüksekliğinin (yani seviye sayısı) ağaçtaki öğe sayısının +1'in tavanıyla eşit olduğunu belirleyebildik. Algoritmamızın Faz I'i her bir değeri listesinin sonuna ekler ve sonra yığındaki son konumuna yukarı süzülür. Yukarı süzülen fazla h seviyesinden geçeceği için ve yığın her seferinde bir öğe ile büyüdüğü için, aşağıdaki toplam, Faz I'de yapılması gereken işin bir üst sınırını tanımlar. 

**223 9.5 Sürüm 1 Aşamasının Analizi**

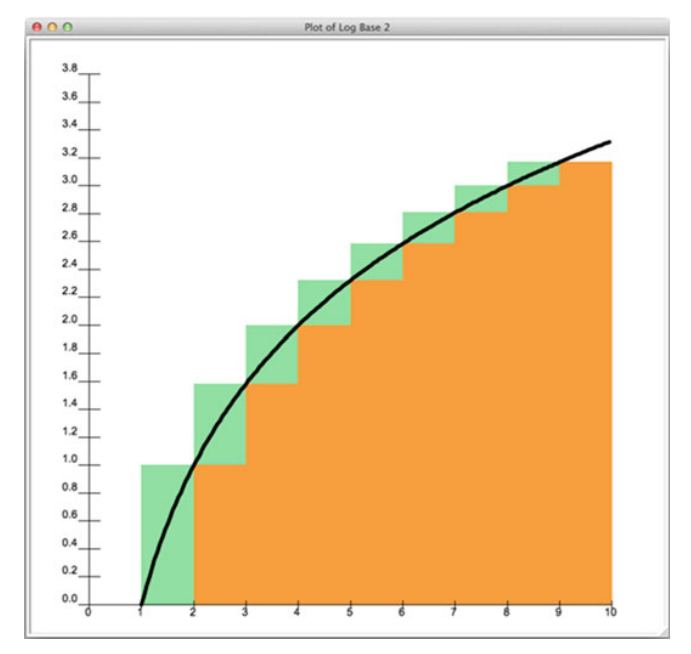
Ancak, yukarıdaki eşitsizliği uyguladığımızda aşağıdakileri elde ederiz. N - 1 terimi, 2'den N' e kadar olan son toplamdan gelir. Yukarıdaki eşitsizlikten, toplamın bir parçası olan N - 1 adet birlik vardır. Bunlar N - 1 olarak dışarı alınabilir.



Şimdi, toplamımız için bir alt ve üst sınırımız var. Aynı toplama, hem alt sınırda hem de üst sınırda görünür. Peki,ΣNI=2log2 I neye eşittir? Aşağıdaki eşitlikler, bu toplamın belirlenmesine yardımcı olacaktır.

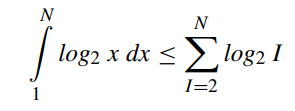


Yukarıdaki toplamın neye eşit olduğunu belirlemek için, toplamı üstten ve alttan sınırlayan birkaç eşitsizlik kurabiliriz. Şekil 9.11'de, toplama yeşil alan olarak görselleştirilebilir. Toplamdaki ilk terim, ilk yeşil dikdörtgeni sağlar, ikinci yeşil dikdörtgen toplamdaki ikinci terime karşılık gelir ve benzer şekilde devam eder. Şekildeki siyah çizgi, x'in logaritmasının tabanı 2'nin çizimidir. Açıkça, yeşil dikdörtgenler tarafından kaplanan alan, altındaki alandan daha büyüktür.

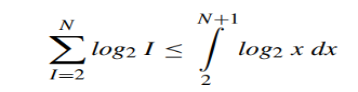
**Tablo 9.11** Log(n) grafiği.

**224 9 Yığınlar**

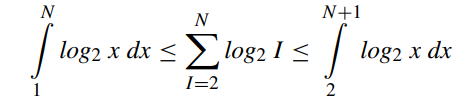
Logaritmanın eğrisinin altındaki alan, resimde 9 olmakla birlikte genelde N olan kesin integral alınarak bulunabilir. Bunu aşağıdaki eşitsizlikten elde ederiz



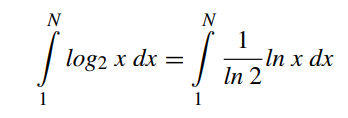
Şimdi, tüm yeşil alanı sağa doğru bir birim kaydırmayı düşünelim. Yukarıdaki şekilde bu, turuncu alan olarak gösterilmiştir. Turuncu ve yeşil alanlar tam olarak aynı büyüklüktedir; tek fark, turuncunun sağa kaydırılmış olmasıdır. Şimdi, log2 x eğrisinin grafiğine bakalım. Eğrinin altındaki alanın artık turuncu alandan açıkça daha büyük olduğu görülmektedir. Bu grafiğin N' e kadar uzandığını düşünürsek, belirli integralimize N+1’i de dahil etmemiz gerekir (çünkü turuncu alanı sağa kaydırdık). Bu nedenle aşağıdaki eşitsizliği elde ederiz.



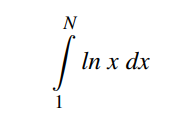
Bu iki eşitsizliği bir araya getirerek toplamımız için bir alt ve üst sınır elde ederiz.



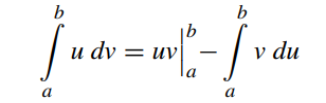
Doğal logaritmalar kullanarak entegrasyonu daha kolay hale getirmek için kesirli integrali aşağıdaki gibi yeniden yazacağız.



Integraldeki sabit terim dışarı çıkarılabilir. Bu nedenle aşağıdaki integrali inceleyeceğiz.

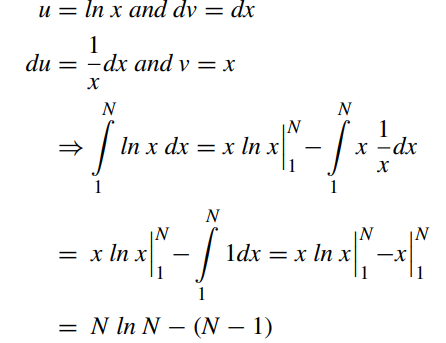


Yukarıda görünen kesin integralin sonucunu, parçalar halinde entegrasyon yaparak bulabiliriz. Parçalar halinde entegrasyon kuralı şöyledir.



**225 9.5 Sürüm 1 Aşama I’in Analizi**

Bu entegrasyonu uyguladığımızda aşağıdakileri elde ederiz..

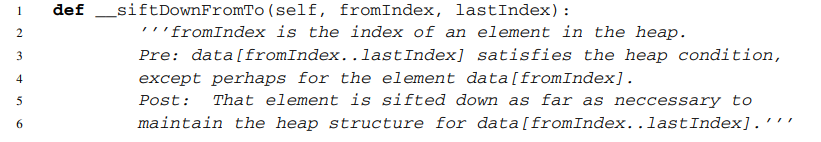


Alt sınırın N log N ile orantılı olduğunu kanıtlamış olduk. Benzer şekilde, üst sınırın da N log N ile orantılı olduğunu kanıtlayabilirdik. Bu nedenle, N öğeyi bir yığına eklerken **siftUpFrom** yöntemi kullanarak eklemenin getirdiği iş, θ (N log N) olur.Ancak daha iyisini yapabiliriz! Eğer yığında bulunan değerler doğru sıraya sahip olsaydı O(N) karmaşıklığına ulaşabilirdik. Farklı bir yaklaşım kullanarak, tüm durumlarda O(N) karmaşıklığına ulaşabiliriz.

**9.6 Yığınlar II**

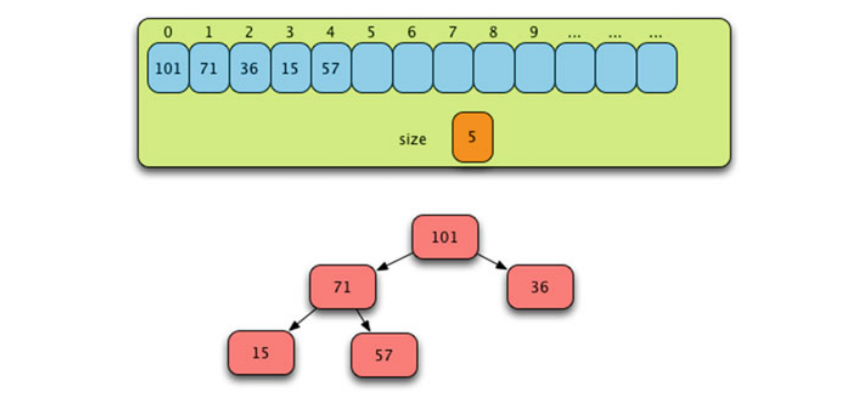
Daha sonra, Aşama I'in performansını nasıl iyileştirebileceğimizi inceleyeceğiz. Hatırlayacak olursak, yığın sıralama algoritmasının Aşama I'i, bir değerler listesinden bir yığın oluşturur. Aşama II ise, öğeleri yığından birer birer çıkarır ve bunları bir listeye yerleştirir. Alan tasarrufu sağlamak amacıyla, yığın için kullanılan aynı liste, döndürülecek değerlerin listesi olarak da kullanılabilir. Aşama II'nin her geçişi, listeden bir öğe alır ve onu hak ettiği yere yerleştirir, yığının boyutu birer birer azalır. Anahtar işlem **siftDownFromTo**  yöntemidir . (Şekil 9.12).

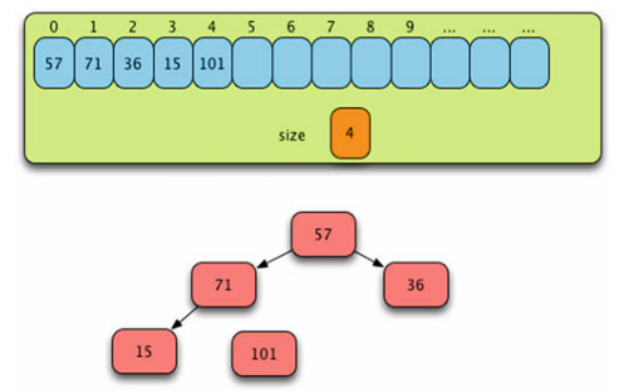
**9.6.1 siftDownFromTo Yöntemi**



Bu yöntemi görselleştirmek için, küçük bir yığın örneğimizi alalım ve ondan değerleri çıkarmaya başlayalım. Şekil 9.13'teki yığını düşünün, hem kavramsal görünüm hem de o yığının düzeni gösterilmiştir. 101 yığının en üstünde ve aynı zamanda en büyük değerdir.

**226 9 Yığınlar**

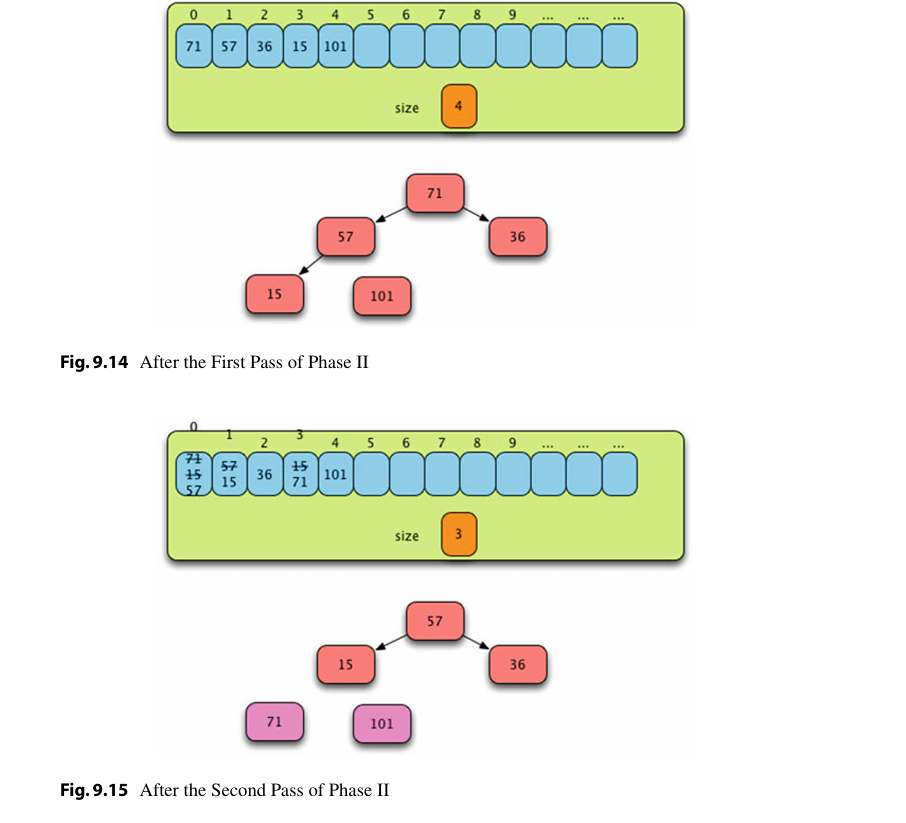
**ĞITablo 9.12 Aşama II’den önce.**



**Tablo 9. 13 İlk ve Son Değerin Değiştirilmesinden Sonra**

Eğer sıralanmışsa, 101 listenin sonuna gider. Yığındaki 5 öğe olduğundan, 57 ile 101'i takas ederiz. Bunu yaparak, 101 sıralı liste içinde nihai konumuna gelir. 57 ise yığındaki doğru konumunda değildir. Bu yüzden **siftDownFromTo** yöntemini çağırarak, 57'yi yığındaki 0. pozisyondan en fazla boyut-1 konumuna kadar kaydırırız. **SiftDownFromTo** yöntemi işlemi yapar ve 57'yi, iki çocuktan daha büyük olan 71 ile takas eder. 57, 15'ten büyük olduğu için daha fazla kaydırılmasına gerek yoktur. Bu durumda, Aşama II 'nin ilk geçişi sonrasında yığının görünümü Şekil 9.14'te gösterildiği gibi olur. Aşama II ' nin ikinci geçişi, 15 ile 71'i takas eder ve 71'i sıralı liste içindeki nihai konumuna taşır. Ardından 15'i, yığındaki hak ettiği konuma kaydırarak, Şekil 9.15'te gördüğünüz görüntüyü üretir.

**9.6 Aşama II'nin Analizi**   227

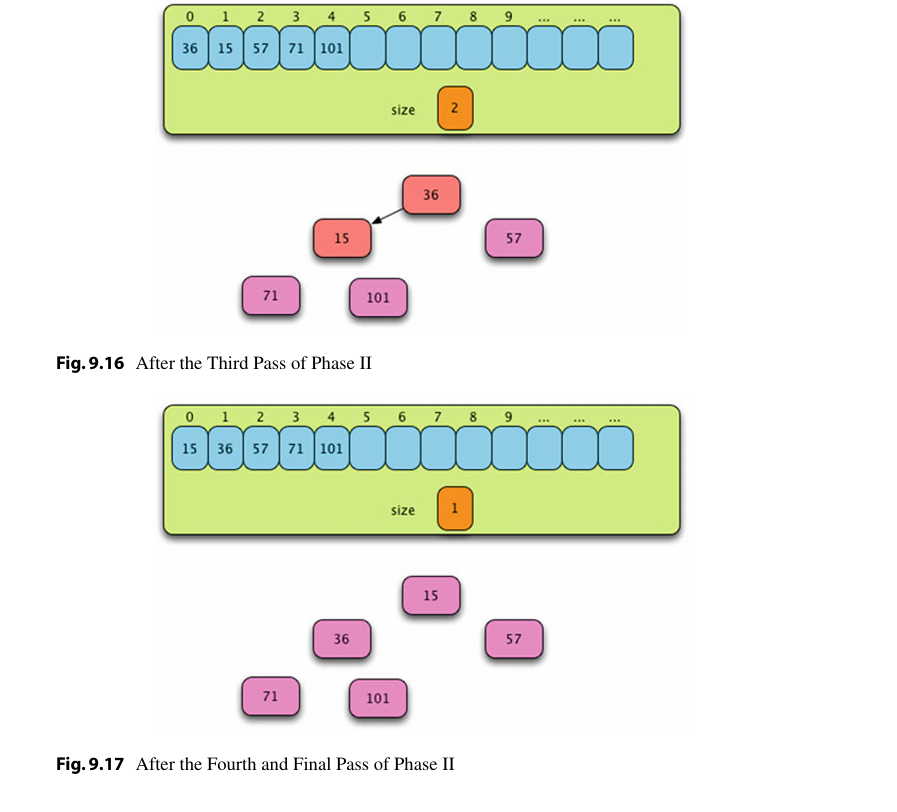


Aşama II'nin üçüncü geçişi sırasında 57 son konumuna yerleştirilir ve kendisine yer açmak için 36 ile değiştirilir. Her ne kadar \_\_siftDownFromTo çağrılsa da, 36 en üstte olduğu ve yığındaki en büyük değer olduğu için yığın içinde herhangi bir değer hareketi gerçekleşmez (Fig.9.16).

Dördüncü ve son geçiş sırasında 36, 15 ile değiştirilir. Bu sefer \_siftDownFromTo çağrısına gerek yoktur çünkü yığın takastan sonra sadece 1 boyutundadır. Büyüklüğü 1 olan bir yığın zaten sıralanmış ve doğru yerde olduğundan Liste artık Şekil 9.17'de gösterildiği gibi ek bir dizi kullanmadan yerinde sıralanır.

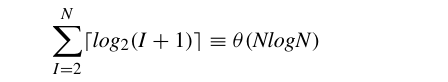
[www.it-ebooks.info](https://www.it-ebooks.info)

228 9 Yığınlar



**9.7 İkinci Aşama Analizi**

Faz II'nin çalışması, N−1 kez çağrılan \_\_siftDownFromTo yöntemine yapılan çağrılardadır. Her çağrı, ağaçtaki her seferinde bir öğe küçülen bir öğeyi elemelidir. Bu bölümün başlarında, ortalama ve en kötü durumdaki iş miktarının aşağıdakilerle orantılı olduğunu belirlemek için analizi yaptık



Aşama II'nin en iyi durumu, yığındaki tüm değerlerin aynı olmasını gerektirir. Bu durumda, hesaplama karmaşıklığı O (N) olacaktır, çünkü değerler asla düşmeyecektir.

[www.it-ebooks.info](https://www.it-ebooks.info)

**9.7 Aşama II'nin Analizi** 229

Bu en iyi durum senaryosu iyi bir noktayı ortaya koyuyor. Değerin ne kadar aşağı elendiğini sınırlayabilseydik, I. Aşamayı hızlandırabilirdik. Konumuz veya bir sonraki bölümümüz bu.

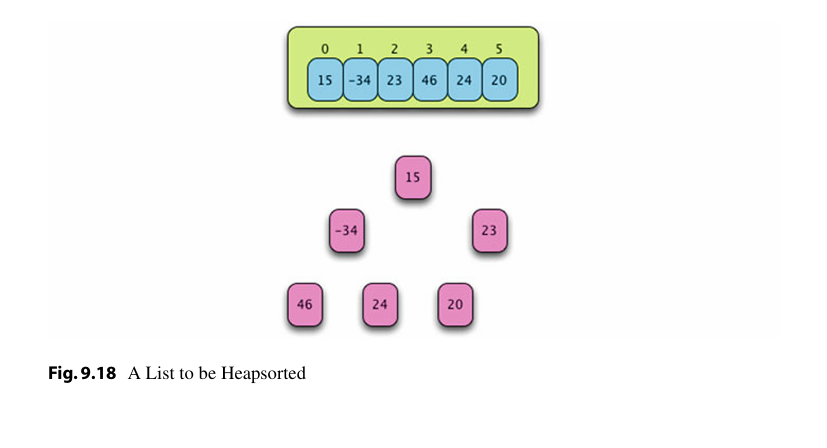
**9.8 Yığın Sıralama Algoritması Sürüm 2**

Birinci versiyonda, yığın algoritması Faz I ve Faz II sırasında O (N log N) karmaşıklığına ulaştı. İkinci versiyonda, yığın algoritmasının Faz I'ini O (N) karmaşıklığına kadar hızlandırabileceğiz. Bunu, yeni eklenen her değerin ne kadar elenmesi gerektiğini sınırlayarak yaparız. Fikir oldukça basit ama yine de güçlü bir teknik. Her öğeyi yığının en üstüne eklemek yerine, yığını veya yığınları aşağıdan yukarıya doğru oluşturacağız. Bu, yığınımızı oluşturmaya listenin başından değil sonundan başlayarak yaklaşacağımız anlamına gelir. Bir örnek bunu daha açık hale getirmeye yardımcı olacaktır. Şekil 9.18'deki yığın sıralamayı kullanarak sıralamak istediğimiz değerlerin listesini düşünün.

Listenin ilk elemanından başlamak yerine, listenin diğer ucundan başlayacağız. Göreceğimiz gibi son unsurla başlamaya gerek yoktur. Ağaçtaki bir düğümün üst öğesi olan bir düğüm seçmemiz gerekiyor. Nihai yığın ikili bir yığın olduğundan, sahip olduğumuz özellik, ağacın düğümlerinin yarısının yaprak düğümleri olması ve yığın içindeki herhangi bir düğümün üst öğesi olamamasıdır. İlk ana endeksi aşağıdaki gibi hesaplayabiliriz.

parentIndex = (size − 2)//2

Yukarıdaki boyut, sıralanacak listenin boyutudur. Listenin 0 ila -1 boyutunda indeksleri olduğundan, her durumda uygun üst İndeksi hesaplamak için iki çıkarmamız gerektiğini unutmayın. Bu durumda, bu üst Dizin 2'dir. Yığınlarımızı aşağıdan yukarıya doğru oluşturmaya başlamak için listede dizin 2 ile başlamamız gerekir. İndeks 2 ilk üst olacak ve gerektiği kadar aşağıya doğru eleyeceğiz.



[www.it-ebooks.info](https://www.it-ebooks.info)

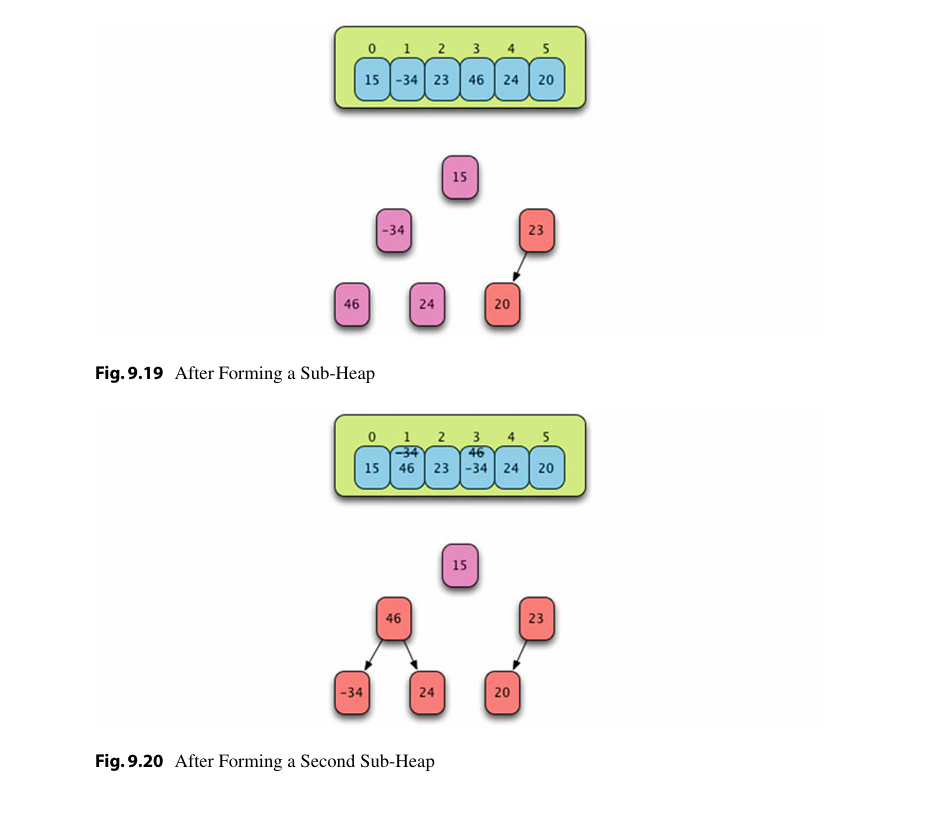
230 9 Yığınlar

childIndex1 = 2 ∗ parentIndex + 1 = 2 ∗ 2 +1 = 5

childIndex2 = 2 ∗ parentIndex + 2 = 2 ∗ 2 +2 = 6

Bu indekslerden ikincisi listenin son indeksinin ötesinde olduğundan, \_\_siftDownFromTo yöntemi alt İndeks 2'yi dikkate almayacaktır. 20 ve 23'ü göz önünde bulundurduktan sonra, bu iki düğümün aslında Şekil 9.19'da gösterildiği gibi bir yığın oluşturduğunu görüyoruz. Bunu aşağıdaki şekillerde bir okla birleştirerek göstereceğiz. Artık 5 yığınımız var, başladığımızdan bir tane daha az. Daha da önemlisi, üst ögeyi en fazla bir pozisyon aşağı kaydırmak zorunda kaldık.

Ardından, listede bir geri giderek dizin 1'e gidiyoruz. Bu düğümden başlayacağımızı belirterek \_\_siftDownFromTo'yu çağırıyoruz. Bunu yapmak, sift down yönteminin takas etmek için iki çocuktan büyük olanı seçmesine neden olur ve sonuç olarak -34, 46 ve 24 değerlerinden bir yığın oluşturur. Bu durum Şekil 9.20'de gösterilmektedir.

 [www.it-ebooks.info](https://www.it-ebooks.info)