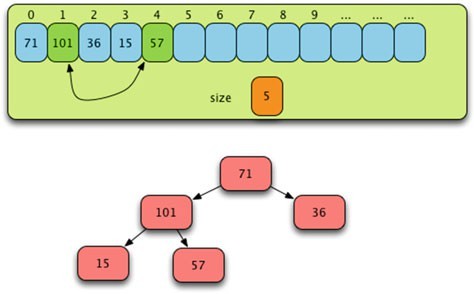
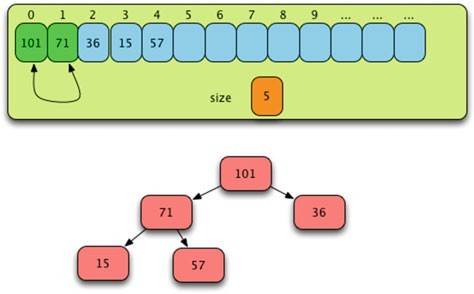
ERBİL BAYÇORAL 2024481015

* 1. Yığın Oluşturma 219



**Şekil 9.5** Yığın Oluşturma İkinci Bölüm



**Şekil 9.6** Yığın Oluşturma Üçüncü Bölüm

Doğru yere.

*parentIndex*= *(*1− 1*)*//2= 0

101 ile 71'i karşılaştırır ve iki öğeyi değiştiririz. Bu, eleme işleminin son yinelemesidir çünkü 101 artık yığının köküne (yani indeks 0'a) ulaşmıştır. İki değeri değiştirdikten sonra Şekil [9.6'](#_bookmark0)daki yığını elde ederiz.

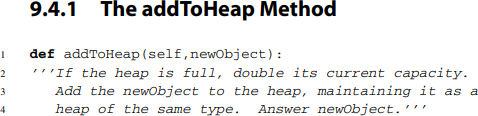
# Heapsort Algoritması Sürüm 1

Yığınların iki temel işlemi vardır. Bir yığına bir değer ekleyebilirsiniz. Ayrıca, yığın üstteki en büyük yığın ise, yığından en büyük değeri silebilir ve geri alabilirsiniz. Bu iki işlemi veya bunların varyasyonlarını kullanarak, bir değerler listesi ile bir yığın oluşturarak ve ardından değerleri tek tek kaldırarak bir sıralama algoritması tasarlayabiliriz.

220 9

Yığınlar

azalan sıra. Algoritmanın bu iki bölümünü aşama I ve aşama II olarak adlandıracağız. Aşama I'i uygulamak için Heap sınıfımızda yeni bir yönteme ihtiyacımız olacak.



Bu yeni yöntem, yeni öğeyi yığın içindeki son hedefine götürmek için *siftUpFrom* özel yöntemini kullanabilir. Aşama I'in 1. sürümü *addToHeap*'i *n* kez çağırır. Bu da O(n log n) karmaşıklığı ile sonuçlanır. Aşama I'in belirli adımları şunları içerir:

1. Gerekirse yığının kapasitesini iki katına çıkarın.
2. data[size]= newObject
3. siftUpFrom(size)
4. size+=1.

Gördüğünüz gibi, *siftUpFrom* yığının her elemanı için bir kez olmak üzere *n* kez

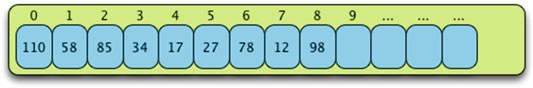
çağrılacaktır. *siftUpFrom* her çağrıldığında, yığın 1 eleman büyümüş olacaktır. Şekil

[9.7](#_bookmark1)'deki yığını 9. geçişten hemen önce düşünün.

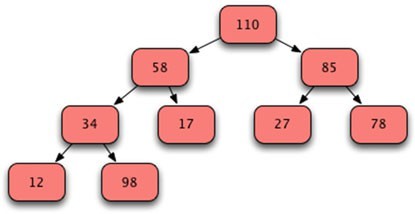
98'i yığın içindeki doğru konumuna kadar elemek üzereyiz. Kavramsal olarak Şekil

[9.8'](#_bookmark2)de gösterilen yığın resmine sahibiz.

98'i doğru konuma taşımak için Tablo [9.1](#_bookmark3)'de gösterildiği gibi alt indekslerden üst indeksi hesaplamalıyız.



**Şekil 9.7** Yığına 98 Ekleme

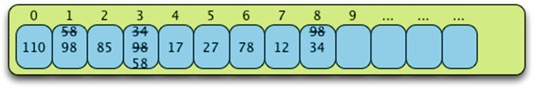


**Şekil 9.8** Yığına 98 Eklenirken Kavramsal Görünüm

9.4 Heapsort Algoritması Sürüm 1 221

**Tablo 9.1** Çocuk ve Ebeveyn Endeksleri

|  |  |
| --- | --- |
| childIndex | parentIndex= (childIndex− 1)//2 |
| 8 | 3 (swap) |
| 3 | 1 (swap) |
| 1 | 0 (stop) |



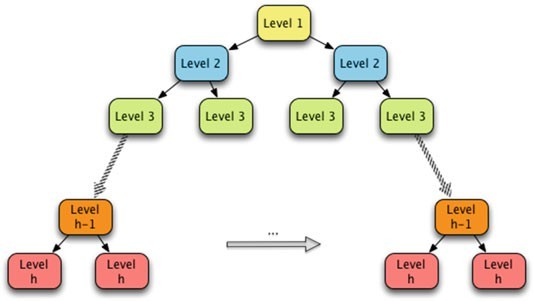
**Şekil 9.9** 98 Doğru Konuma Taşındıktan Sonra Yığın

Yığının listesinde 98'in elenmesi, son konumuna ulaşmadan önce iki takasla sonuçlanır. Şekil [9.9](#_bookmark4), 98'in 3. dizindeki 34 ile değiştirildiğini göstermektedir. Daha sonra dizin 1'deki 58 ile tekrar değiştirilir. Bu noktada daha fazla takas yapılmaz çünkü 101 98'den büyüktür. 98, yığın içindeki uygun konumuna ulaşmıştır.

# Versiyon 1 Aşama I'in Analizi

Aşama 1'in 1. versiyonunda benimsenen yaklaşım, göreceğimiz gibi yavaştır. Mükemmel bir tam ikili ağaç düşünün. Şekil [9.10'](#_bookmark5)da gösterildiği gibi *h* seviyeli, tüm seviyeleri tamamen dolu olan bir ağaç.

Tablo [9.2'](#_bookmark6)de gösterildiği gibi seviye sayısı ile yığındaki öğe sayısı arasındaki ilişkiyi göz önünde bulundurun.

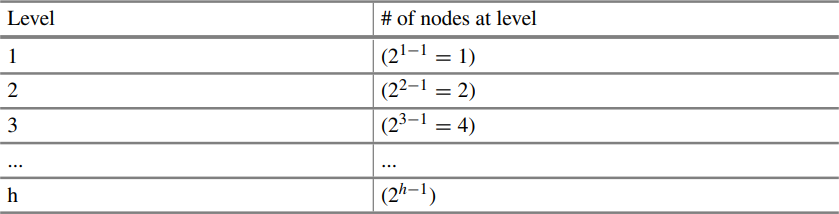


**Şekil 9.10** Mükemmel Bir İkili Ağaç

222 9

Yığınlar

**Tablo 9.2** Yığın boyutuna karşı yığın seviyeleri



*n* öğe bulunan bir yığın için *n* değeri, yığının ağacındaki her seviyedeki tüm düğümler toplanarak hesaplanabilir. Tartışmamızı basitleştirmek için yığının tam bir ikili ağaç olduğunu varsayacağız.

Bu bir geometrik dizinin toplamıdır. Bir geometrik dizinin toplamı aşağıdaki gibi hesaplanabilir.

Bu formülü yukarıdaki denklemimize uygularsak, *h* seviyeli tam bir ikili ağaçtaki (yani tam bir ikili yığın) düğüm sayısı aşağıdaki formülle verilir.

*h*

Bu da *n*+ 1= 2*h* anlamına gelir. Bu denklemi *h* için çözebiliriz. Bunu yaparak şunları elde ederiz



Yukarıdaki parantezler tavan operatörüdür ve basitçe bir sonraki en yüksek tamsayıya yuvarlamamız gerektiği anlamına gelir. Yukarı yuvarlama, her yığın ağacının tamamen dolu olmadığını dikkate alır, bu nedenle yukarı yuvarlamadığımız takdirde bize *h* için bir tamsayı vermeyecek bazı *n* değerleri olabilir. Aşağıdaki eşitsizlik, heapsort algoritmasının I. aşamasının hesaplama karmaşıklığını belirlemede faydalı 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 süzülme en 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.

