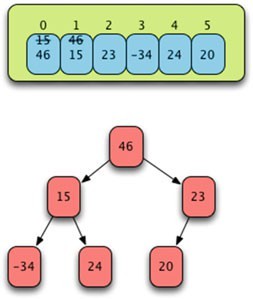
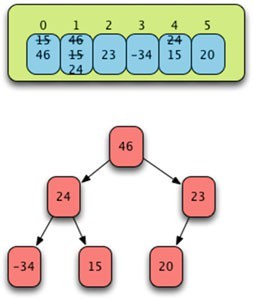
* 1. Heapsort Algoritması Sürüm 2 231

Son olarak, listede bir eleman daha geriye giderek 0 indeksine ulaşıyoruz. Bu kez yalnızca iki alt öğenin değerlerine bakmamız gerekiyor çünkü bunlar zaten kendi yığınlarındaki en büyük değerler . Listenin ilk elemanı üzerinde *siftDownFromTo* çağrıldığında 15, 46 ve 23 arasından en büyük değer seçilecek ve 15 değeri bu değerle değiştirilerek Şekil [9.21](#_bookmark0)'deki durum ortaya çıkacaktır.

Bu henüz tam bir yığın oluşturmaz. Hala 15'i tekrar aşağı taşımamız gerekiyor ve *siftDownFromTo*, Şekil [9.22](#_bookmark1)'de gösterildiği gibi 15'i yığının en altına taşımayı hallediyor.



**Şekil 9.21** 15'in Aşağı Elenmesi



**Şekil 9.22** Aşama I'in 2**.** Versiyonunu kullanan Nihai Yığın

232

# Heapsort Sürüm 2'nin Analizi

9

Yığınlar

Aşama II'nin, değerlerin bir yığında olduğu ve sıralanmış listeyi oluşturmak için her seferinde bir tane çıkarıldığı zaman olduğunu hatırlayın. Hapsort algoritmasının 2. Aşaması 1. Aşama ile aynıdır ve karmaşıklığı O(N log N)'dir.

Öte yandan Sürüm 2 Aşama I, sürüm 1'deki yukarıdan aşağıya yığın oluşturma yaklaşımından sürüm 2'de aşağıdan yukarıya yığın oluşturma yaklaşımına geçmiştir. Bu yeni aşama I'in karmaşıklığının O(N) olduğunu iddia ettik, burada *N* listedeki düğüm sayısıdır. Daha resmi olarak ifade edersek bu iddiaya sahibiz. (2*(h*)− 1) *düğüm içeren h yüksekliğinde mükemmel bir ikili ağaç için, maksimum karşılaştırma yollarının uzunluklarının toplamı* (2*(h) 1*− − h)'*dir*.

Yüksekliği *h'*ye kadar olan 1, 2, vb. yükseklikteki ikili yığınları düşünün.

Algoritmanın 2. versiyonu için verilen örnekten, *siftDownFromTo*'ya yapılan herhangi bir çağrı için maksimum yol uzunluğunun Tablo [9.3'](#_bookmark2)te gösterildiği gibi belirleneceği açık olmalıdır).

(2*h*− 1)'in nihai yığındaki düğümlerin yarısını (yaprak düğümleri) temsil ettiğine ve

yığındaki düğümlerin yarısı için maksimum yol uzunluğunun 0 olacağına dikkat edin. Bu gözlem, aşağıdan yukarıya doğru bir yığın oluşturmak için daha verimli bir algoritmaya yol açar. Tüm bu maksimum yol uzunluklarını toplayabilseydik, bu algoritmanın 2. versiyonunun 1. aşamasında yapılacak iş miktarı için bir üst sınıra sahip olurduk.

*S*= 1∗ *(h*− 1*)*+ 2∗ *(h*− 2*)*+ 22∗ *(h*− 3*)* + - - -+ 2*(h*)− 3∗ 2+ 2*(h*)−(2)∗ 1

*S* değeri, yapılacak işin bir üst sınırı, maksimum yol uzunluklarının toplamı olacaktır. Formülde küçük bir değişiklik yaparak bu toplamdaki terimlerin çoğunu ortadan kaldırabiliriz. S değeri 2S− S= S olarak hesaplanabilir. Bu formülü kullanarak değeri şu şekilde yazabiliriz

*S*= 2∗ *S*− *S*= 2∗ *(h*− 1)+ 2(2)∗ *(h*− 2*)*+ ---+ 2*(h*)−(2)∗ 2+ 2*(h*)− 1∗ 1

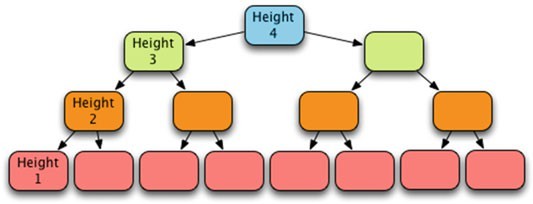
- [*(h*− 1*)*+ 2∗ *(h*− 2*)*+ 2*(*2)∗ *(h*− 3*)* + - - -+ 2*(h*)−*(*2)∗ 1]

Yukarıdaki denklemde terimleri sıralarsak (şu anda sıralandıkları gibi), benzer terimleri çıkarabiliriz. İlk benzer terimde h− 1− (h− 2) görüyoruz. Bu şu şekilde basitleşir

**Tablo 9.3** siftDownFromTo için maksimum yol uzunluğu

|  |  |  |
| --- | --- | --- |
| Seviye | Maksimum yol uzunluğu | # Seviyedeki düğüm sayısı |
| 1 | *h*− 1 | 1 |
| 2 | *h*− 2 | 2 |
| 3 | *h*− 3 | 4 |
| ... | ... | ... |
| *h*− 2 | 2 | 2*h 3*− |
| *h*− 1 | 1 | 2*h 2*− |
| *h* | 0 | 2*h 1*− |

9.9 Heapsort Sürüm 2'nin Analizi 233



**Şekil 9.23** Yüksekliği 4 Olan Bir İkili Yığın

h− h− 1+ 2 = 1. Benzer şekilde, diğer benzer terimler de sadeleşir ve böylece S için

aşağıdaki formül elde edilir.

*S*= 2∗ *S*− *S*= 2+ 2(2)+ + 2*(h*)− 2+ 2*(h*)− 1− *(h*− 1*)*

= 1+ 2+ 2(2)+ + 2*h*−(2)+ 2*h*− 1− *h*

= 2*h*− 1− *h*≡ *O(N)burada N*= 2*(h)*− 1 *düğüm.*

Yukarıdaki basitleştirmenin son adımında, geometrik bir dizinin toplamı olarak da bilinen, 2'nin ilk h− 1 kuvvetlerinin toplamına sahibiz. Bu toplam, 2'nin h yükseltilmiş haline eşittir, eksi bir. Bu, tümevarım yoluyla basit bir kanıtla ispatlanabilir. Böylece, aşama I'in 2. versiyonunun O(N) olduğunu kanıtlamış olduk. Aşama II hala O(N log N) olduğundan yığın sıralamanın genel karmaşıklığı O(N log N)'dir.

Yüksekliği 4 olan ikili bir yığın düşünün (Şekil [9.23](#_bookmark3)).

Böyle bir yığında, aşağı eleme yöntemi kullanıldığında, ilk eleme ağaçta bir seviye aşağı inebilen dört düğümün bulunduğu 2. yükseklikte gerçekleşir. Yükseklik 3'te iki seviye aşağı inebilen iki düğümümüz vardır. Son olarak, kök düğüm üç seviye aşağı inebilir. Aşağıdaki maksimum yol uzunlukları toplamına sahibiz.

1+ 1+ 1+ 1+ 2+ 2+=

=

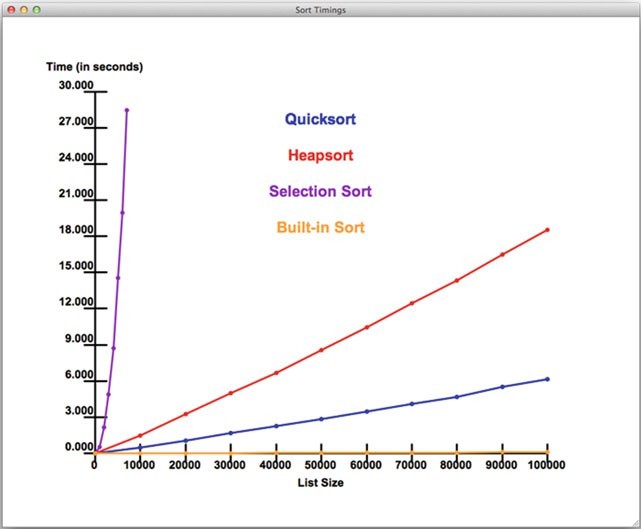
24− 1−=

2*h*− 1− *h*

# Diğer Sıralama Algoritmaları ile Karşılaştırma

Hapsort algoritması, quicksort algoritmasıyla aynı karmaşıklıkta olan O(N log N) zamanda çalışır. Önemli bir fark, tek tek değerlerin hareketidir. Quicksort'ta değerler her zaman son konumlarına doğru hareket ettirilir. Heapsort, değerleri önce bir yığın oluşturmak için hareket ettirir, ardından son konumlarına ulaşmak için tekrar hareket ettirir

234 9

Yığınlar

**Şekil 9.24** Çeşitli Sıralama Algoritmalarının Karşılaştırılması

sıralanmış bir liste içinde. Quicksort, aynı hesaplama karmaşıklığına sahip olmalarına rağmen heapsort'tan daha verimlidir.

Şekil [9.24](#_bookmark4) incelendiğinde, çok kısa listeler dışında kabul edilemez olan *θ* (N2) karmaşıklığıyla çalışan seçme sıralamasını görüyoruz. Quicksort algoritması beklendiği gibi heapsort algoritmasından daha olumlu davranmaktadır. C'de uygulanan quicksort olan yerleşik sıralama, C'de uygulanması nedeniyle en hızlı şekilde çalışır.

# Bölüm Özeti

Bu bölümde yığınlar ve yığın sıralama algoritması tanıtılmıştır. Bir yığın oluşturmak O(N) zaman karmaşıklığında verimli bir şekilde yapılabilir. Bir yığın, en üstteki elemanın sıralı bir değer koleksiyonunun en büyük ya da en küçük elemanı olacağını garanti eder. Bu prensibi kullanarak yığınları kullanarak birçok algoritma ve veri tipi uygulayabiliriz. algoritması bu bölümde yığınların kullanımına bir örnek olarak sunulmuştur. Yığınlar değer aramak için iyi değildir. Bir yığında bir değeri aramak O(N) zaman alacaktır ve bir değer için bir listenin doğrusal olarak aranmasından daha iyi olmayacaktır. Bunun nedeni, bir yığın içindeki öğelerin en büyük (veya en küçük) değerin en üstte olması dışında bir sıralamasının olmamasıdır. Bir değerin yığınınneresinde olduğunu belirleyemezsi niz.

En büyük değere eşit veya daha büyük olmadığı ve en büyük üst yığınınız olmadığı sürece, tüm yığını aramadan. Aynı şekilde, en küçük üstte yığınınız varsa ve bir değer arıyorsanız, aradığınız değer en küçük değere eşit veya daha küçük olmadığı sürece tüm değerlere bakmanız gerekir.

Genellikle yığınlar, bir kuyruğun elemanlarının bir tür öncelik değerine göre sıralandığı öncelik kuyruklarını uygulamak için kullanılır. Mevcut bir yığına O(log N) zamanında bir eleman eklenebilir. Bir yığından bir eleman da O(log N) zamanda çıkarılabilir. Bu, bir yığını öncelik kuyruğu uygulaması için mantıklı bir seçim haline getirir. Öncelik kuyrukları, mesaj geçirme çerçevelerinde ve özellikle bazı çizge algoritmalarında ve sezgisel arama algoritmalarında kullanışlıdır.

-- 9.13 Programming Problems

3-) **En küçük öğenin en üstte olduğu bir yığın (heap) uygulayın ve bunu bir öncelikli kuyruk (priority queue) uygulamak için kullanın.**  
Bir öncelikli kuyruğun enqueue (eleman ekleme) ve dequeue (eleman çıkarma) yöntemleri vardır.  
Öncelikli kuyruğa bir öğe eklerken, bir öncelik değeri belirtilmelidir.  
Kuyruğa eklenen öğeler, hem veri öğesini hem de önceliği içermelidir.  
Öncelikli kuyruk veri yapınızı test etmek için bir test programı yazın.

4-) **Önceki alıştırmadaki öncelikli kuyruğu kullanarak, 7. bölümdeki Dijkstra algoritmasını uygulayın.**  
Öncelikli kuyruk kullanarak yapılan Dijkstra algoritması daha verimlidir.  
Her öğenin önceliği, öncelikli kuyruğa eklenen her düğümün o ana kadar olan maliyetidir.  
Öncelikli kuyruktan öğe çıkardığımızda, otomatik olarak en düşük maliyetli düğümü aramaya gerek kalmadan elde ederiz.  
Bu sayede algoritmanın karmaşıklığı **O(|V| log |V|)** olur; bu, **O(|V|²)** olan standart uygulamaya göre daha verimlidir.

5-) **Heapsort algoritmasını (birinci veya ikinci versiyonunu) kullanarak, 7. bölümdeki Kruskal algoritmasını uygulayın.**  
Programınızı test etmek için, ders kitabı web sitesinde bulunan örnek grafik XML dosyalarından birini giriş verisi olarak kullanın.