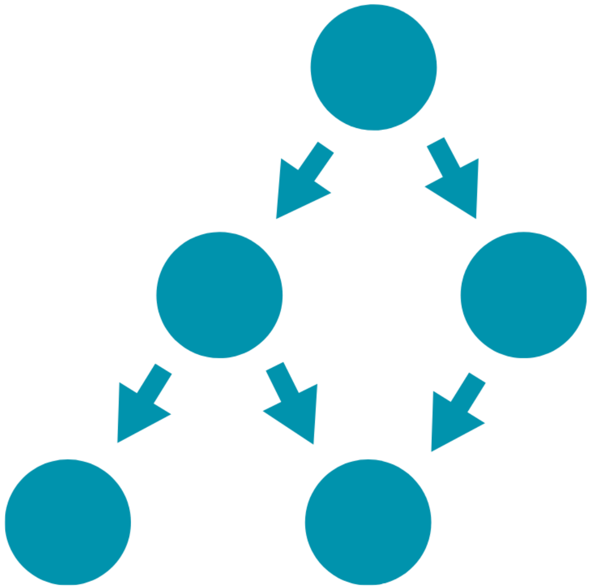


VERİ YAPILARILARI VE ALGORİTMALAR

Priority Queue and Heaps

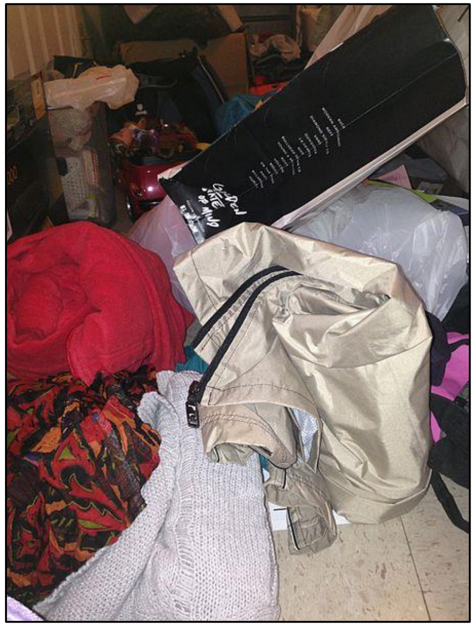
Giriş

1. Priority Queue
2. Priority Queue ADT
3. Priority Queue Applications
4. Priority Time Complexity
5. Heaps and Binary Heaps
6. Min-Heap and Max-Heap
7. Heapsort

Öncelikli Kuyruk

Prioirty Queue

- Hızlı ekleme ve hızlı çıkarma işlevinin her ikisini de isteriz.

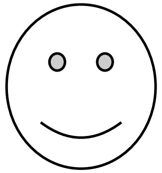


Hızlı ekleme

+

Hızlı çıkarma

=



Öncelikli Kuyruk

Prioirty Queue

- Bazen bir koleksiyonun elemanları arasından en küçük ya da en büyük elemanın seçilmesi istenebilir. Bunun için **`Prioirty Queue ADT`** kullanılabilir.

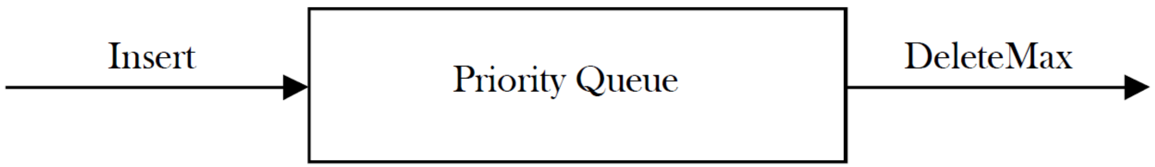
- Bir öncelikli kuyruk soyut veri türü **`Insert`**, **`DeleteMin`**, **`DeleteMax`** işlevlerini içeren veri yapısı olarak tanımlanabilir.

- Bu işlevler aslında **`EnQueue`** ve **`DeQueue`** işlevlerine karşılık gelir. Aradaki fark, öncelik sıralarında, öğelerin sıraya girme sırasının işlendikleri sırayla aynı olmayabilmesidir.

Öncelikli Kuyruk Soyut Veri Türü

Prioirty Queue ADT

- **Ana İşlevler**
  - Insert
  - deleteMin/deleteMax
  - GetMinimum
  - GetMaximum



Sorting (first **`insert`** all, then repeatedly **`deleteMin`**)

- **Yardımcı İşlevler**
  - Kth en küçük/büyük eleman
  - Size
  - Heapsort

Öncelikli Kuyruk Uygulamaları

- Sorting an array using heapsort algorithm
- Implementing priority queues
- Data compression : Huffman Coding algorithm
- Shortest path algorithms : Dijkstra's algorithm
- Minimum spanning tree algorithms: Prim’s algorithm
- Event-driven simulation: customers in a line
- Selection problem: Finding kth- smallest element
- Run multiple programs in the operating system
- Select print jobs in order of decreasing length
- «Greedy» algorithms

Uygulama

Implementation

- Sırasız dizi ile uygulanması durumunda:
  - Elemen ekleme O(1),
  - Silme işlevi O(n) karmaşıklığına göre çalışır.
- Sırasız liste uygulanması durumunda:
  - Ekleme işlevi O(1)
  - Silme işlevi O(n)
- Sıralı dizide:
  - Ekleme işlevi O(n)
  - Minimum silme işlevi O(1)

Uygulama

Implementation

- Sıralı bağlı listede
  - Ekleme O(n)
  - Minimum silme O(1)
- İkili arama ağacında: Ortalama durumda
  - Ekleme O(logn)
  - Silme O(logn)
- Dengeli ikili arama ağacında (balanced binary search treess): En kötü durumda;
  - Ekleme O(logn)
  - Silme O(logn)

Zaman Karmaşıklığı

Time-Complexity

Implementation	Insertion	Deletion (deleteMax)	Find Min
Unordered array	1	n	n
Unordered list	1	n	n
Ordered array	n	1	1
Ordered list	n	1	1
Binary Search Trees	$logn$ (average)	$logn$ (average)	$logn$ (average)
Balanced Binary Search Trees	$logn$	$logn$	$logn$
Binary Heaps	$logn$	$logn$	1

Heaps and Binary Heaps

- Bir heap (yığın) özel nitelikleriyle **bir ikili ağaç türüdür**.

- **Complete olmalıdır.**
  Buradaki temel kural bir yığının çocuklarından eşit ya da küçük veya eşit ya da büyük olma kuralıdır.

- Verilerin düzeni sıralama kuralına uymalıdır:
  Bu kural heap özelliği **`(heap property)`** olarak ifade edilir.

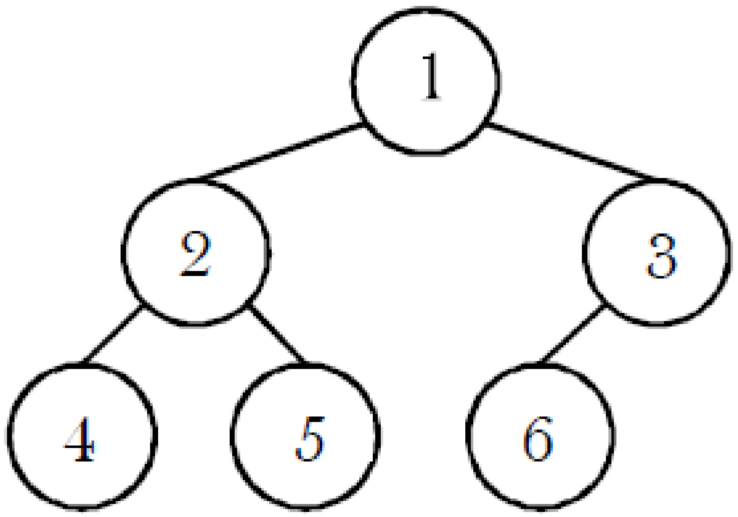
Heaps and Binary Heaps

- Heap yapısının bir diğer özelliği tüm yaprakların en az ***h*** ya da ***h-1*** seviyede olmasıdır.

- Örneğin tam ikili ağaçlarda (`complete binary tree`) bu kural ***h>0*** şeklinde dikkate alınır.

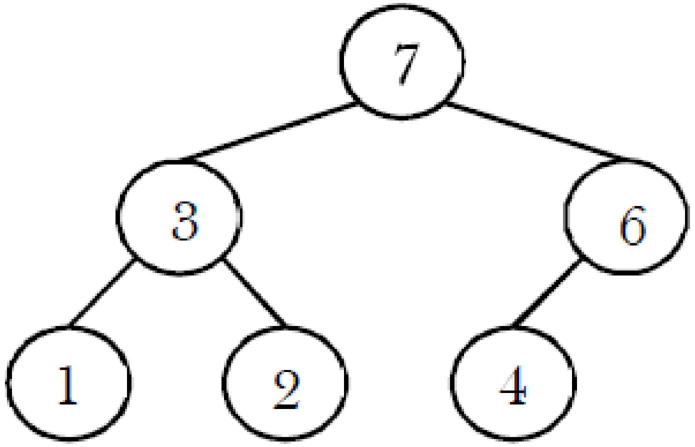
- Bunun anlamı şudur ki; <u>**heap bir tam ikili ağaç oluşturur**</u>**.**

Heaps and Binary Heaps

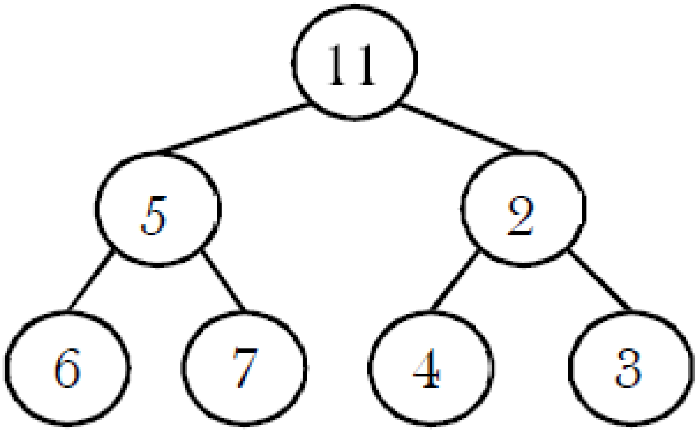


Binary heap örneği

Heaps and Binary Heaps



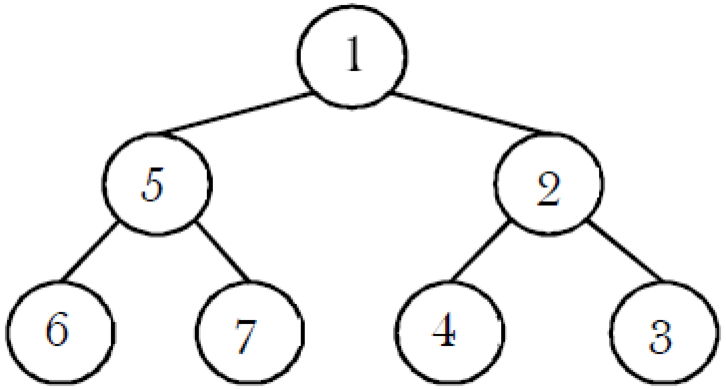
Binary heap örneği



Binary heap değil!

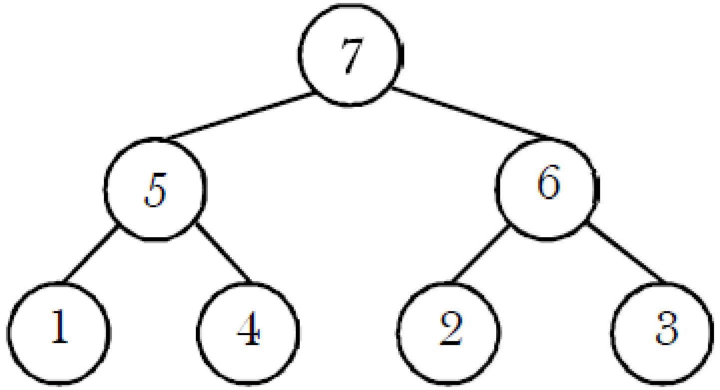
Heap Türleri

Min-Heap



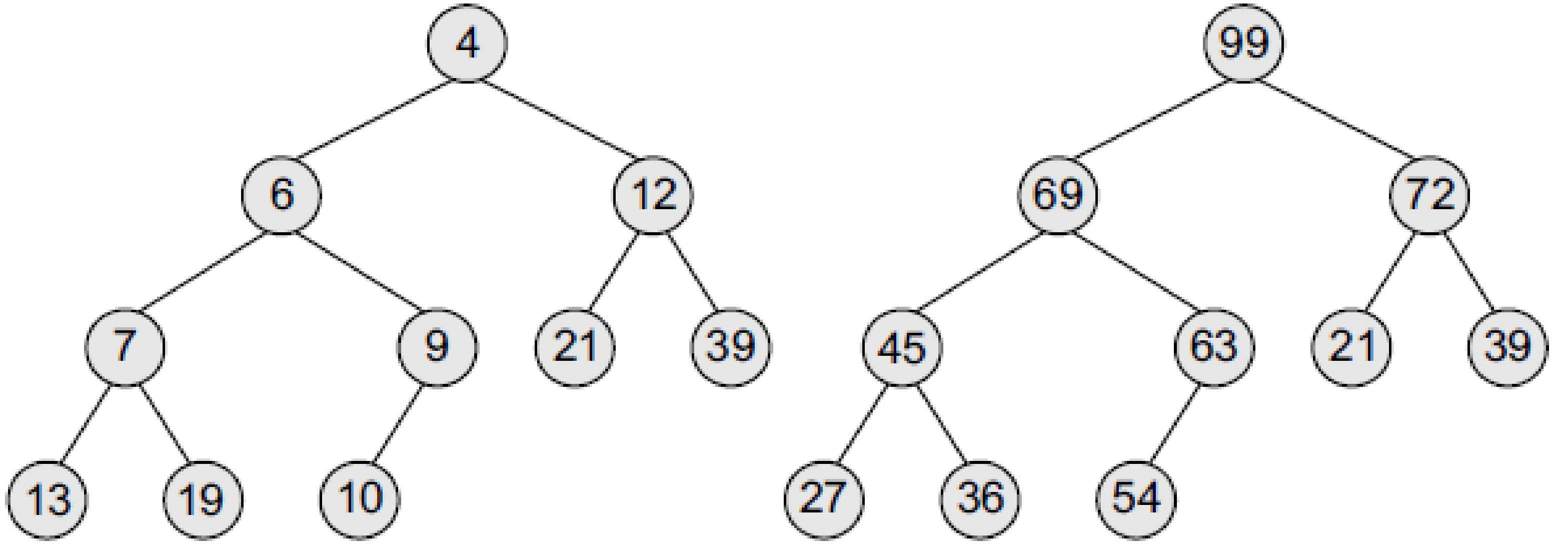
0	1	2	3	4	5	6
1	5	2	6	7	4	3

Max-Heap



0	1	2	3	4	5	6
7	5	6	1	4	2	3

Heap Türleri



Min heap Max heap

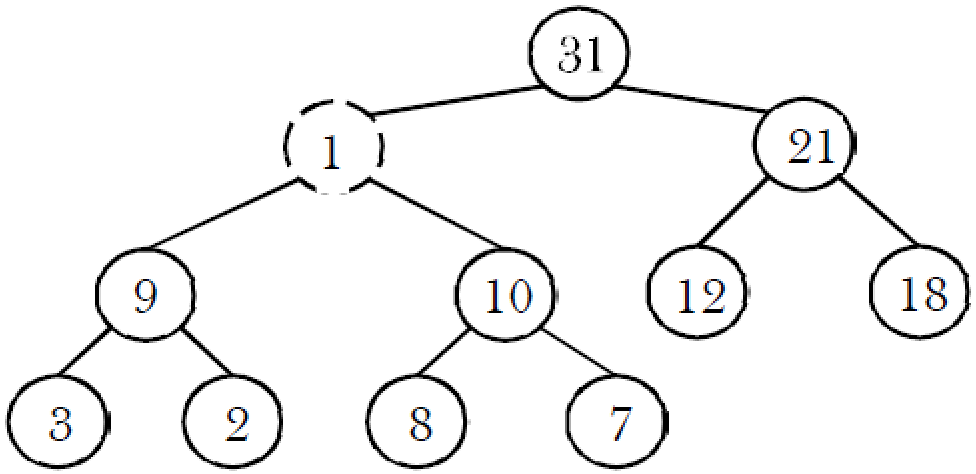
Heapifying

- Bir eleman heap eklendikten sonra, heap özelliği (**`heap property`**) bozulabilir.

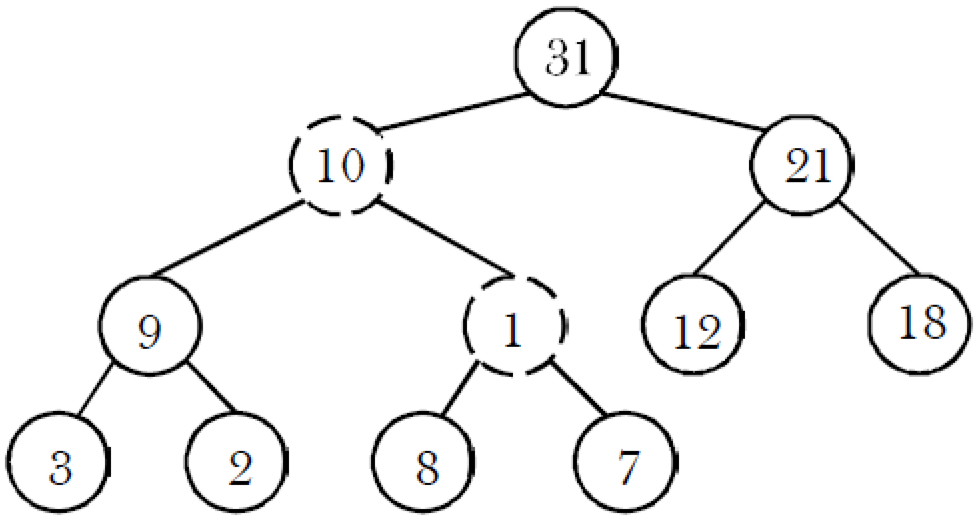
- Tekrar heap özelliğini sağlayabilmek için eklenen elemanın uygun düğüme yerleştirilmesi gerekir.

- Bu işlem **`heapifying`** olarak ifade edilir.

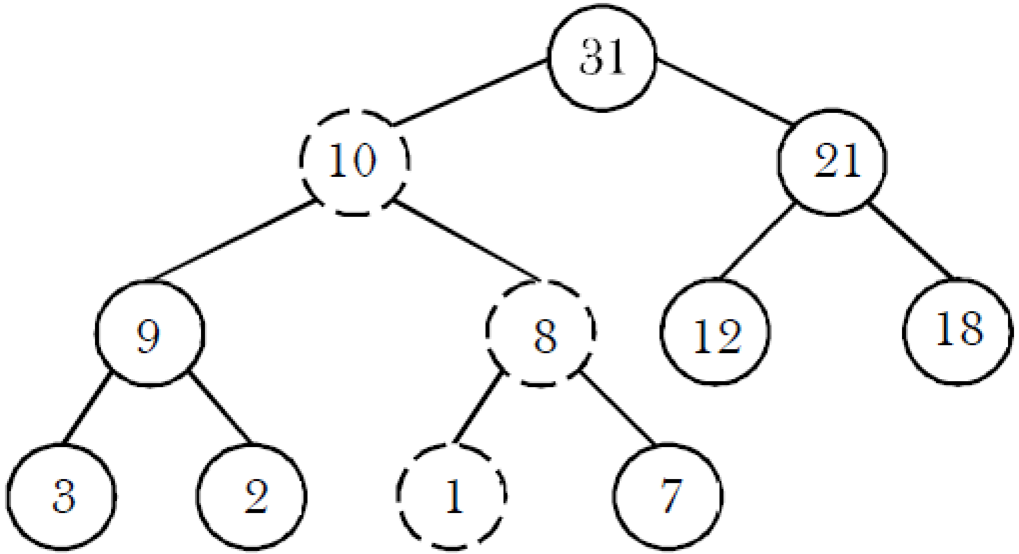
Heapifying



Heapifying



Heapifying



Heapifying

- **`H-heap`** yapısında aşağı doğru süzülme (**percolate down**) olarak ifade edilirken;

- **`H-heap`** yapısında ise yukarı doğru süzülme (**percolate up**) olarak ifade edilir.

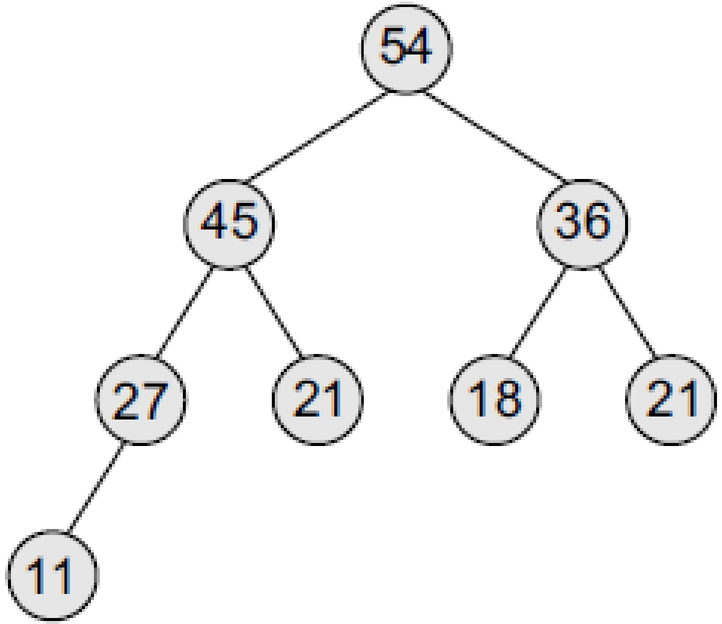
Ekleme

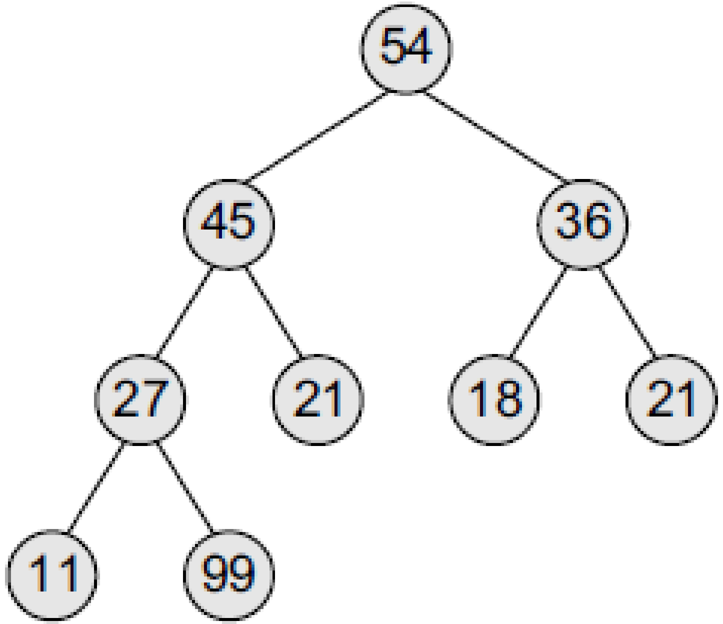
Insertion in Binary Heap

- Bir **`n`** elemanlı **`max-heap H`** yapısına yeni bir eleman eklendiğinde aşağıdaki iki adım takip edilir:
  - **`H`**’da tam ağaç oluşturacak şekilde eleman eklenir.
  - Yeni değer **`H`** ağacının **`heap property`** dikkate alınarak uygun pozisyona doğur takas (**`swap`**) edilir.

Ekleme

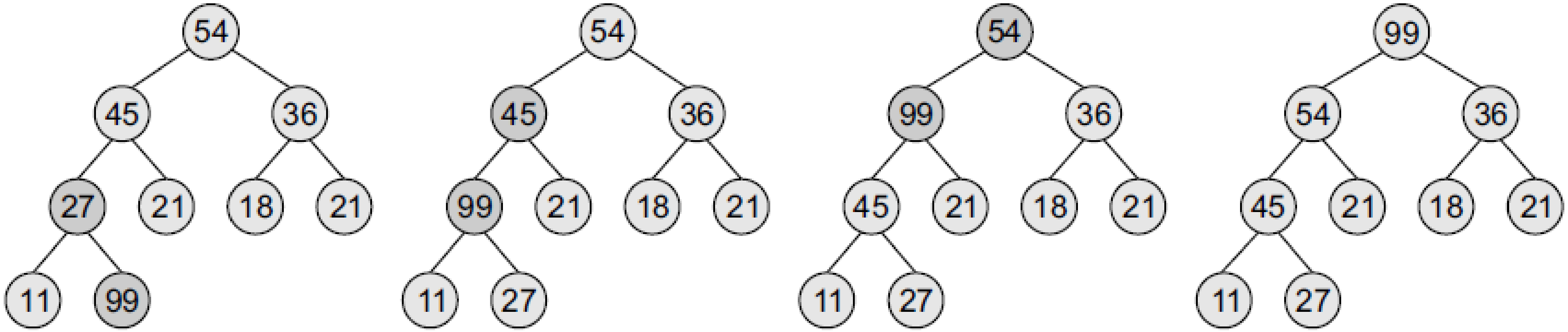
Insertion in Binary Heap



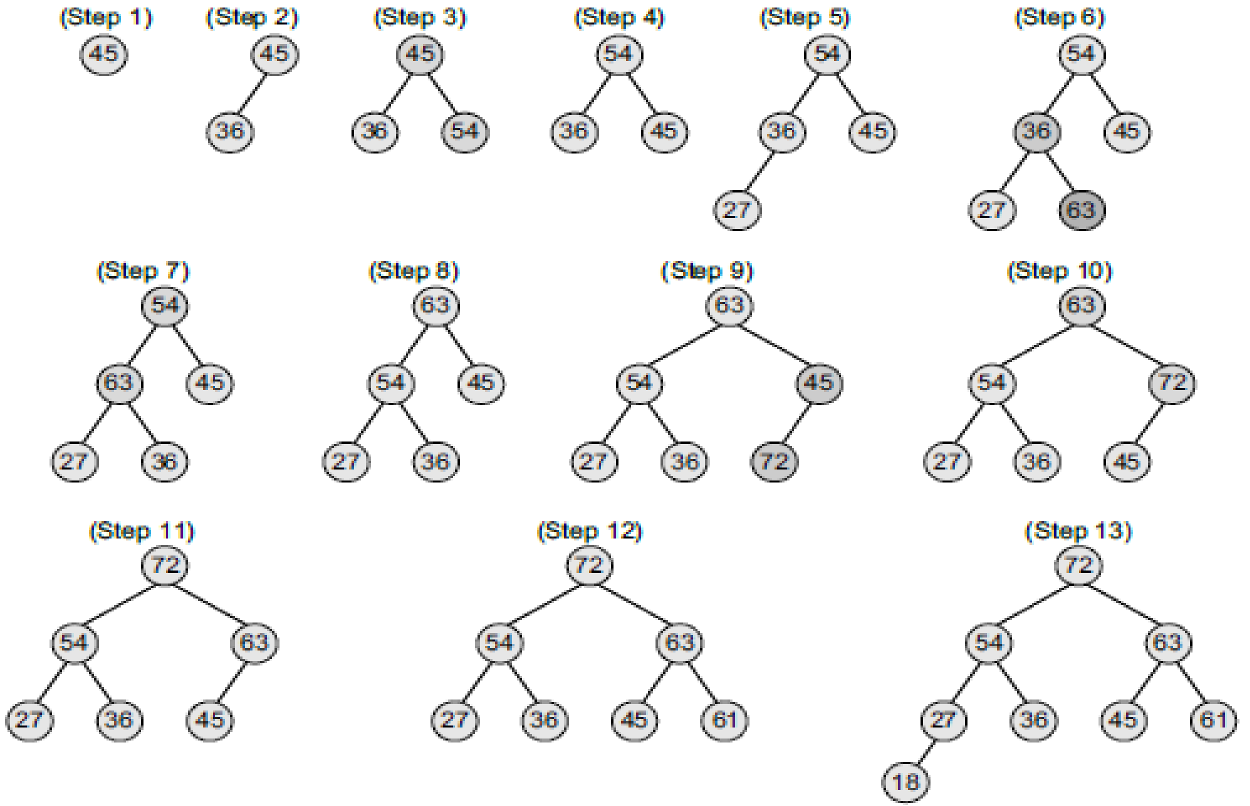


Ekleme

Insertion in Binary Heap



- **Verilen seriden bir H max-heap oluşturunuz:**
- 45, 36, 54, 27, 63, 72, 61 ve 18



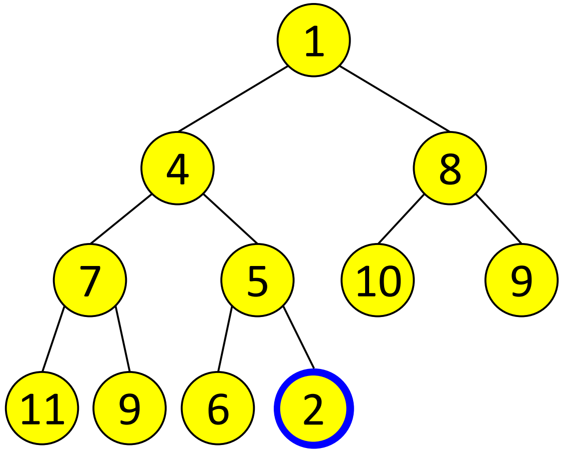
HEAP[1]	HEAP[2]	HEAP[3]	HEAP[4]	HEAP[5]	HEAP[6]	HEAP[7]	HEAP[8]	HEAP[9]	HEAP[10]
72	54	63	27	36	45	61	18		

Ekleme

Insertion in Binary Heap

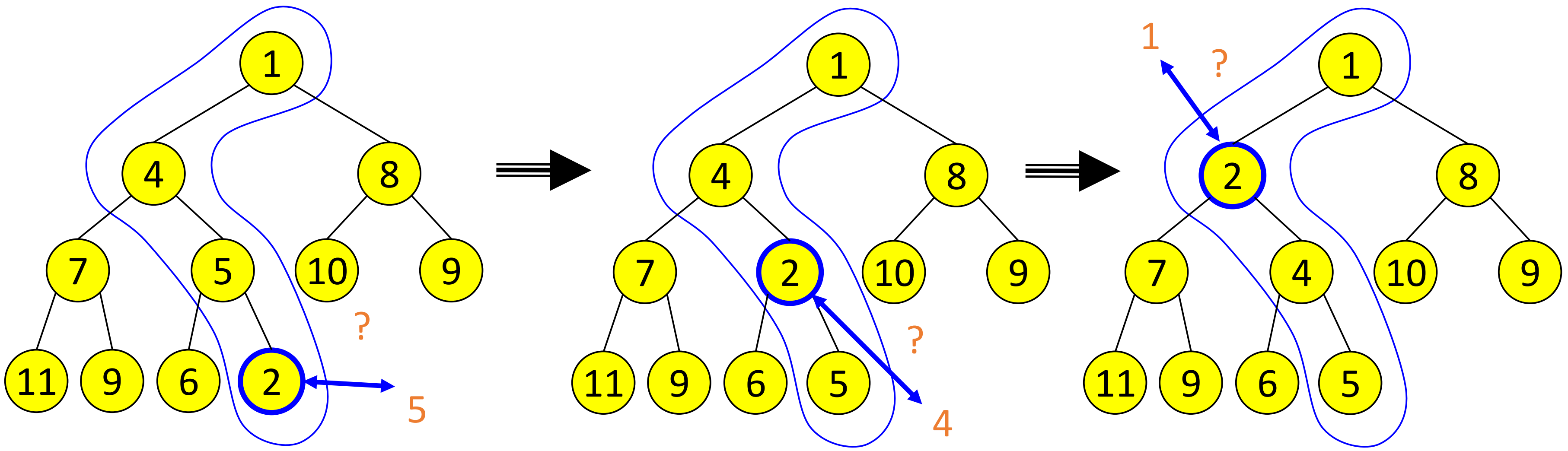
- Yeni bir değer eklediğimizde elimizde tam bir ağaç var ancak heap property bozulmuş durumda olabilir.

- Buna göre eklenen elemanı uygun bir pozisyona eklememiz gerekir.



Ekleme

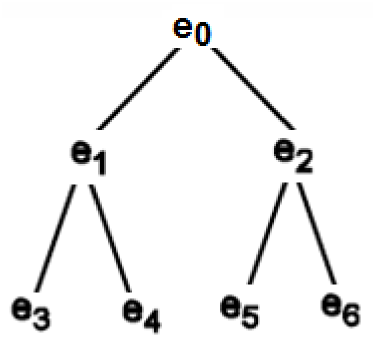
Insertion in Binary Heap

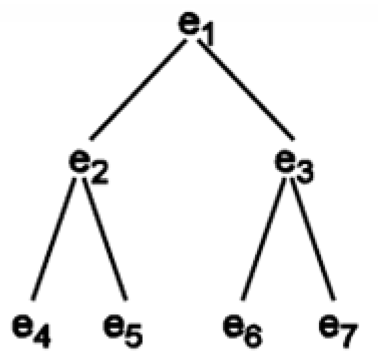


- Yukarı doğru süzülme (Percolate up):
- Yeni düğüm yerleştirilir.
- Eğer ebeveyn düğüm daha büyük ise takas yapılır.
- Gerekirse kök düğüme kadar bu işlev sürdürülür.

Bir Dizide Binary Heap

Binary Heap in an Array





0-based:

- For tree of height h, array length is 2^h-1
- For a node in array index i:
  - Parent is at array index: (i – 1)/2
  - Left child is at array index: 2i + 1
  - Right child is at array index: 2i + 2

1-based:

- For tree of height h, array length is 2^h
- For a node in array index i:
  - Parent is at array index: i /2
  - Left child is at array index: 2i
  - Right child is at array index: 2i + 1

Ekleme için Kabakod

Pseudo Code for Insertion in Binary Heap

```
Step 1: [Add the new value and set its POS]
        SET N = N + 1, POS = N
Step 2: SET HEAP[N] = VAL
Step 3: [Find appropriate location of VAL]
        Repeat Steps 4 and 5 while POS > 1
Step 4:     SET PAR = POS/2
Step 5:     IF HEAP[POS] <= HEAP[PAR],
            then Goto Step 6.
            ELSE
                  SWAP HEAP[POS], HEAP[PAR]
                  POS = PAR
            [END OF IF]
      [END OF LOOP]
Step 6: RETURN
```

Silme

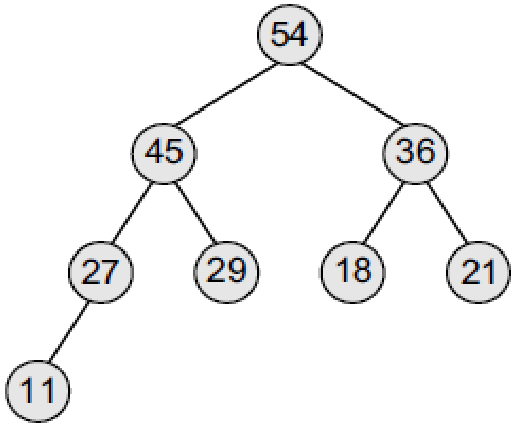
Deletion in Binary Heap

- Bir **`n`** elemanlı **`max-heap H`** yapısından bir eleman silinmek istediğinde; silme işlemi öncelikle kökten yapılır ve dolayısıyla aşağıdaki adımalar takip edilir:
  - **`Root`** düğümünün değeri ile düğümün son değeri yer değiştirilir böylelikle yine tam ağaç yapısı korunur ancak heap property bozulur.
  - Son düğüm silinir.
  - Yeni kök düğüme bağlı olarak **`precolate down/sink down`** işlevi gerçekleştirilerek **`H`** yapısının **`heap property`** özelliği yeniden sağlanır.

Silme

Deletion in Binary Heap

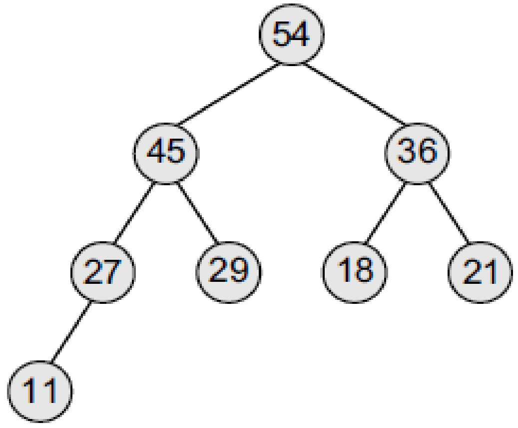
Kök düğümü silelim:

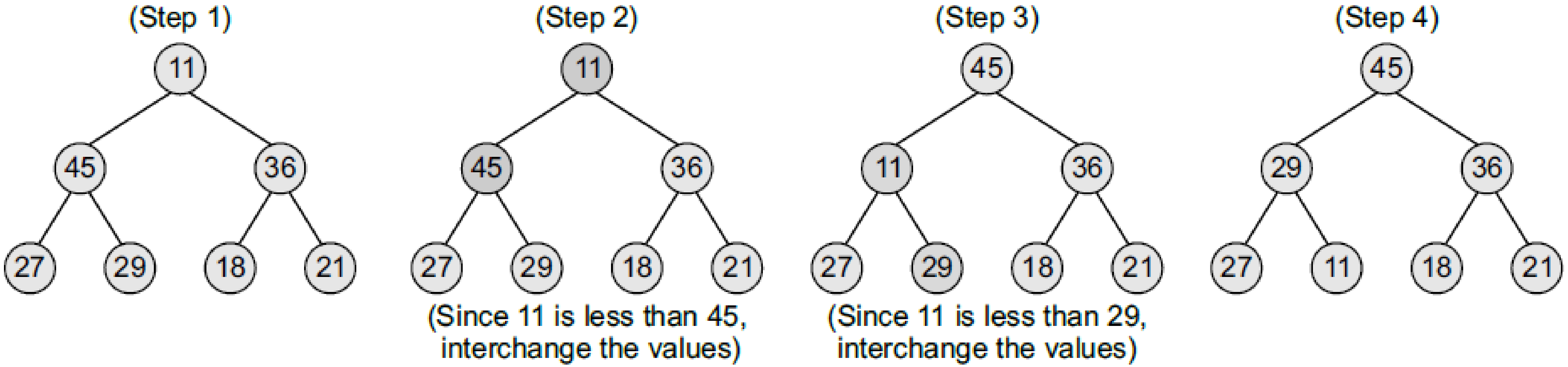


Silme

Deletion in Binary Heap

Kök düğümü silelim:





Silme

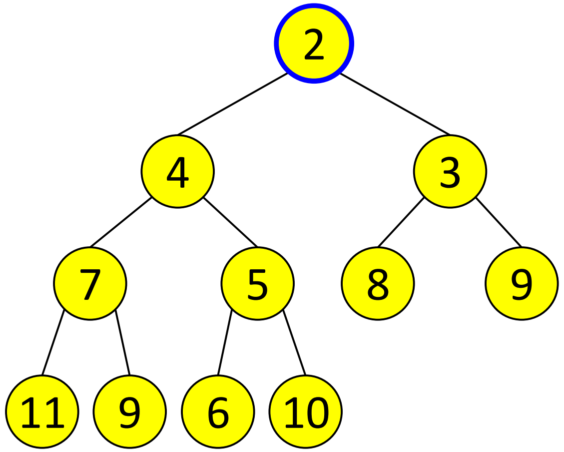
Deletion in Binary Heap

```
Step 1: [Remove the last node from the heap]
        SET LAST = HEAP[N], SET N = N - 1
Step 2: [Initialization]
        SET PTR = 1, LEFT = 2, RIGHT = 3
Step 3: SET HEAP[PTR] = LAST
Step 4: Repeat Steps 5 to 7 while LEFT <= N
Step 5: IF HEAP[PTR] >= HEAP[LEFT] AND
         HEAP[PTR] >= HEAP[RIGHT]
              Go to Step 8
        [END OF IF]
Step 6: IF HEAP[RIGHT] <= HEAP[LEFT]
              SWAP HEAP[PTR], HEAP[LEFT]
              SET PTR = LEFT
        ELSE
              SWAP HEAP[PTR], HEAP[RIGHT]
              SET PTR = RIGHT
        [END OF IF]
Step 7: SET LEFT = 2 * PTR and RIGHT = LEFT + 1
        [END OF LOOP]
Step 8: RETURN
```

Silme

Deletion in Binary Heap

- Root düğümdeki değeri sil.

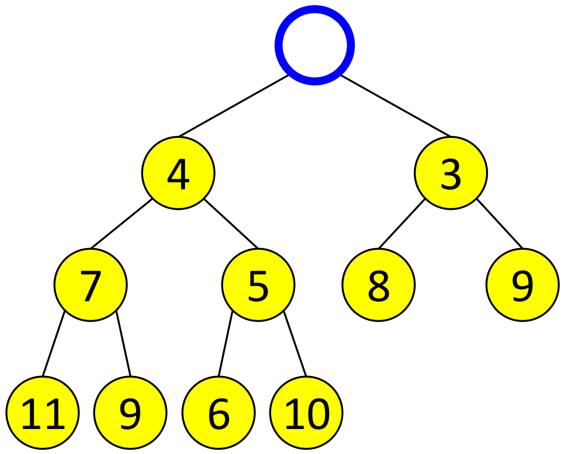


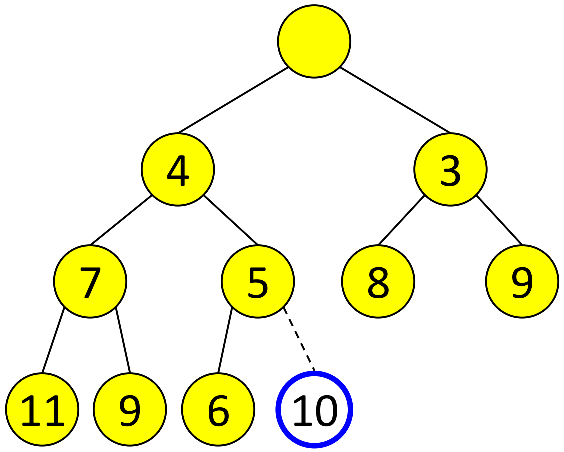
Silme

Deletion in Binary Heap

- Şimdi kökte bir delik var.
  - Bu deliği uygun bir değer ile doldurmalıyız.

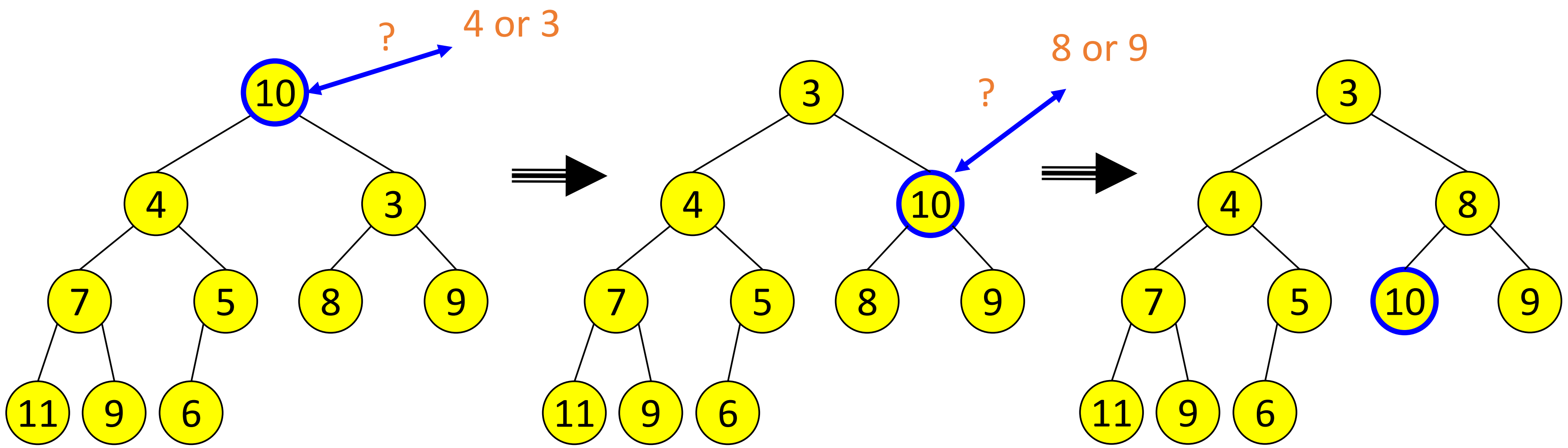
- Bu yaptığımızda, ağaç bir düğüm eksilmiş olacak ve bizim elimizde hala tam bir ağaç yapısı olacak.





Silme

Deletion in Binary Heap

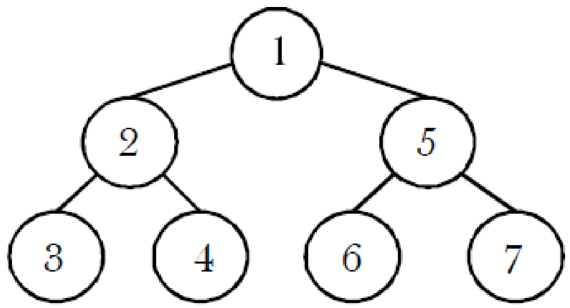


Aşağı doğru süzülme (Percolate down):

- İki çocukla karşılaştır.
- Daha küçük değere sahip olan takas yap.
  - Daha büyük değere sahip olan çocukla takas yapılırsa ne olur?
- Eğer düğüm çocuklardan büyükse ise yapraklara ulaşana kadar bu işleve devam et.

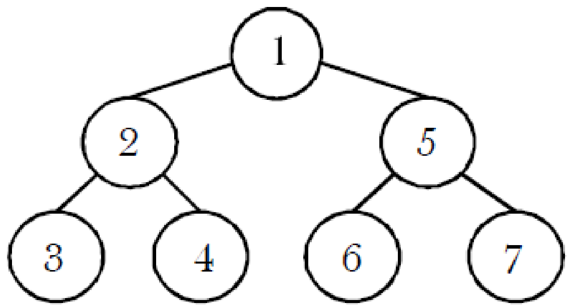
Min-Heap Ağacın PreOrder Dolaşılması

- Min-Heap ağacında PreOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



Min-Heap Ağacın PreOrder Dolaşılması

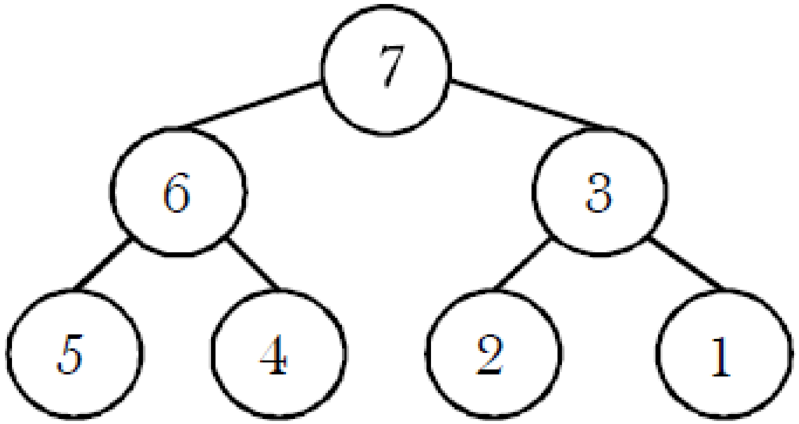
- Min-Heap ağacında PreOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



Yukarıdaki ağaç preOrder dolaşıldığında 1 – 2 – 3 – 4 – 5 – 6 – 7 elemanları sırasıyla gezilir. Dolasıyla ilgili ifade doğrudur.

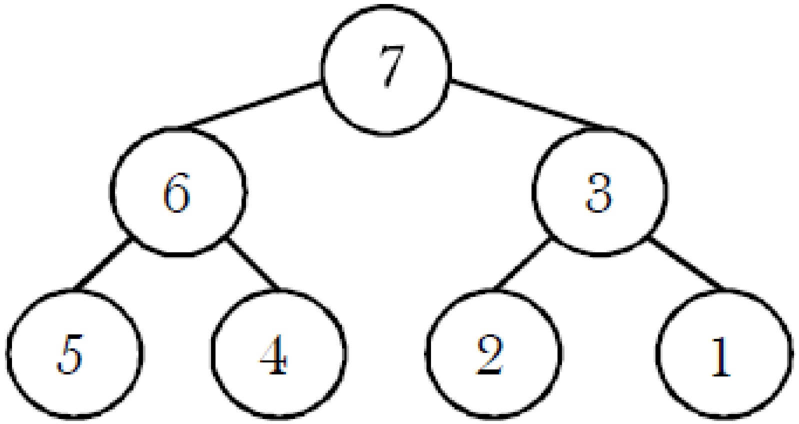
Max-Heap Ağacın PreOrder Dolaşılması

- Max-Heap ağacında PreOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



Max-Heap Ağacın PreOrder Dolaşılması

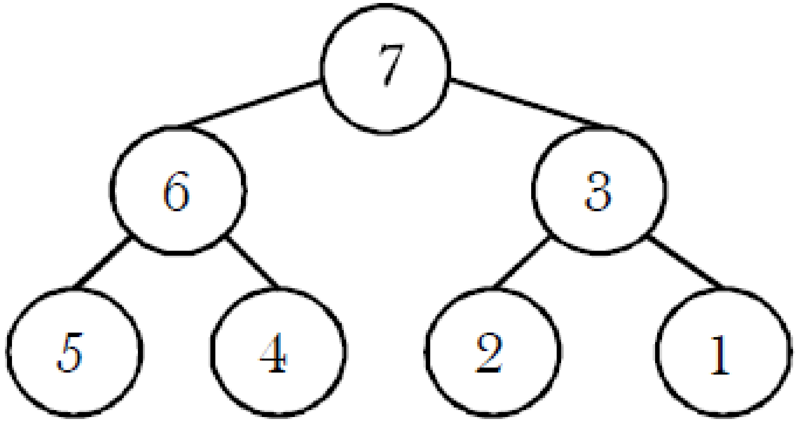
- Max-Heap ağacında PreOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



PreOrderTraversal : 7 – 6 – 5 – 4 – 3 – 2 – 1 üretir. Yani azalan sıralama ile karşılaşılır.

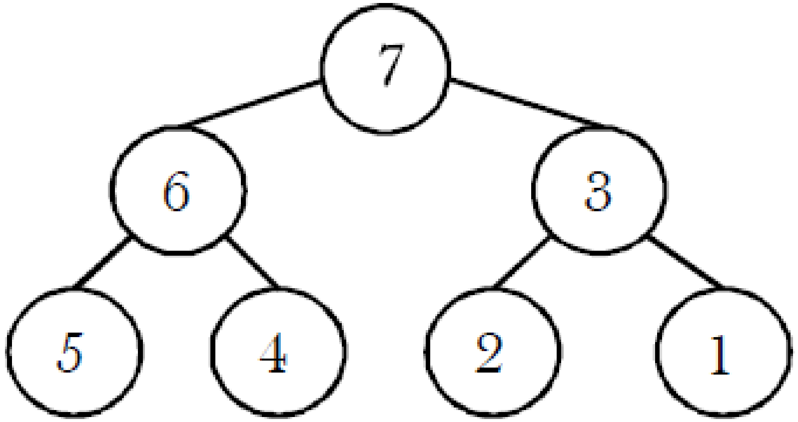
Max-Heap veya Min-Heap Ağacında InOrder Dolaşma

- Max-Heap veya Min-Heap ağacında InOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



Max-Heap veya Min-Heap Ağacında InOrder Dolaşma

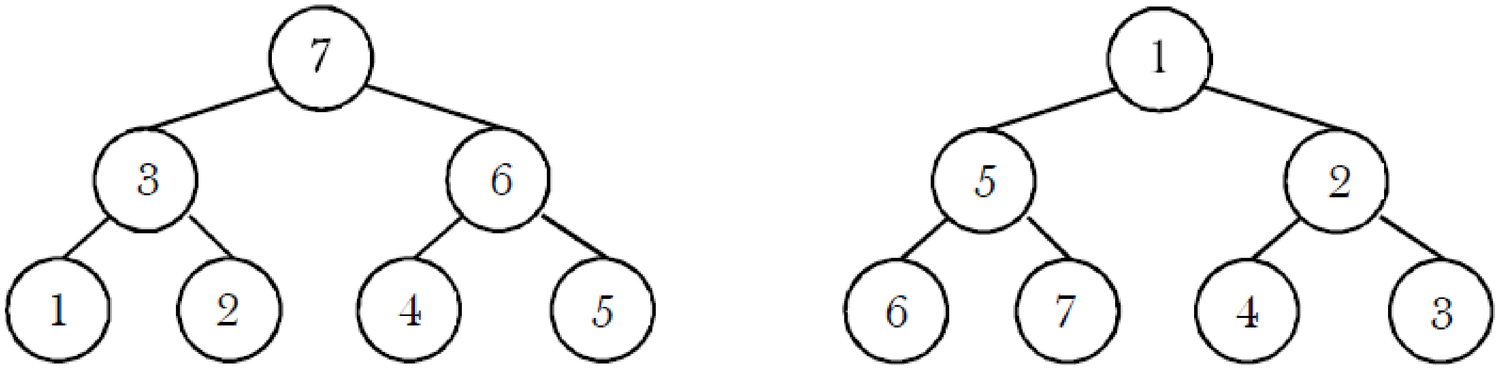
- Max-Heap veya Min-Heap ağacında InOrder Traversal dolaşma sonucunda sıralı bir dizi elde edilir mi?



Hayır. InOrder dolaşmada kök ortada yer alır. Dolasıyla min-heap ya da max-heap yapıda ise kökün en büyük ya da en küçük olması durumu söz konusudur. Dolasıyla InOrderTraversal yapıldığında heap tree sıralı bir dizi üretmeyecektir.

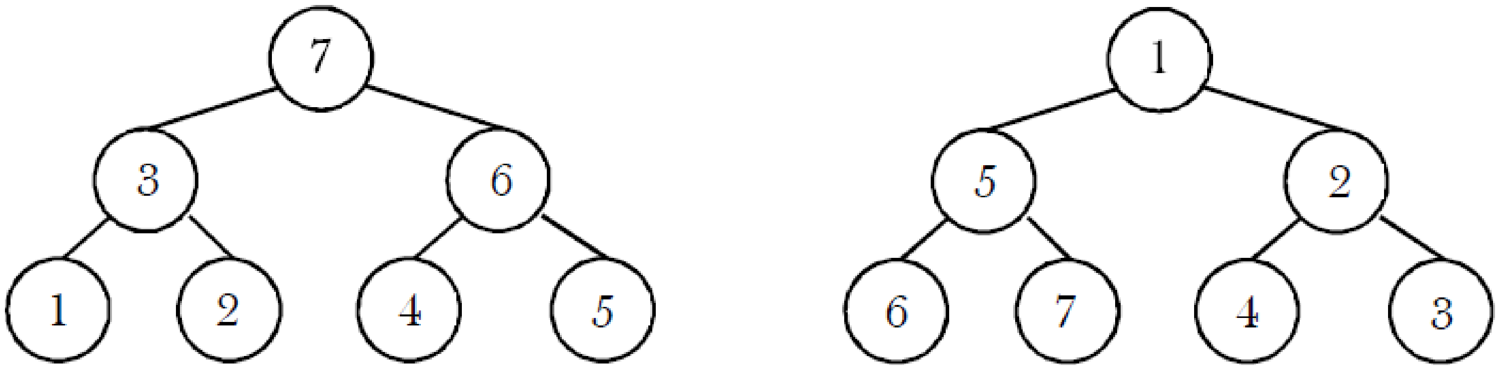
Max-Heap veya Min-Heap Ağacında PostOrder Dolaşma

- Min-heap ya da Max-heap bir yapıda PostOrder dolaşma sıralı bir dizi verir mi?



Max-Heap veya Min-Heap Ağacında PostOrder Dolaşma

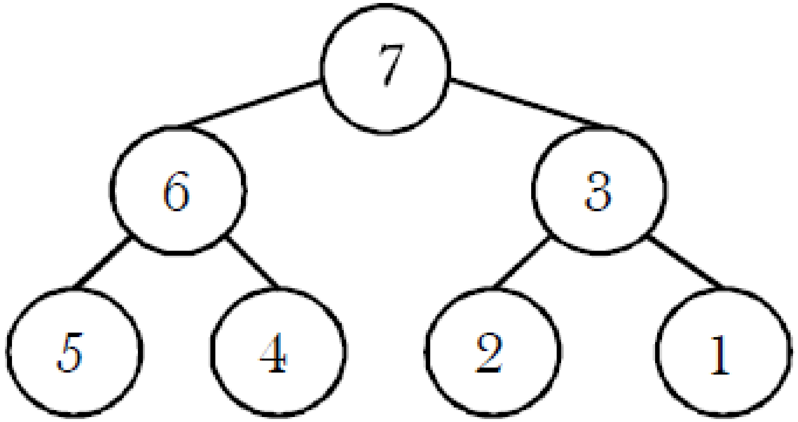
- Min-heap ya da Max-heap bir yapıda PostOrder dolaşma sıralı bir dizi verir mi?



- PostOrderTraversal: 1 – 2 – 3 – 4 – 5 – 6 – 7
- PosrOrderTraversal: 6 – 7 – 5 – 4 – 3 – 2 – 1

Evet, yukarıda görüldüğü üzere PostOrder dolaşma yapıldığında artan ya da azalan sıralanmış çıktı elde edilmektedir.

- h yüksekliğinde bir heap yapısında yer alan minimum ve maksimum eleman sayısı?



$$Maximum : 2^{h+1} - 1$$
$$Minimumu : 2^{h}$$

Heap yapısı bir tam ağaç olduğundan;

$Maximum : 2^{h+1} - 1 \; ve \; Minimumu : 2^{h} \; elemana \; sahip \; olur.$

Heapsort

- Zaman karmaşıklı $\boldsymbol{O(nlogn)}$ olan bir sıralama algoritmasıdır.

- **`n`** elemanlı bir **`Arr`** dizisi iki aşamada sıralanır:
  - Birinci adımda, **`Arr`** elemanları kullanılarak bir heap **`H`** elde edilir.
  - İkinci adımda, kök değer tekrarlı silinerek birinci adım oluşturulur.
  - **`Max-heap`** yapısında en büyük elemanın her zaman kökte olduğu bilinir. Bu nedenle sürekli kökten silme işleminin yapılması aslında azalan bir sıralamanın yapılması anlamına gelir.

```
HEAPSORT(ARR, N)

Step 1: [Build Heap H]
        Repeat for I = 0 to N-1
                CALL Insert_Heap(ARR, N, ARR[I])
        [END OF LOOP]
Step 2: (Repeatedly delete the root element)
        Repeat while N>0
                CALL Delete_Heap(ARR, N, VAL)
                SET N = N + 1
        [END OF LOOP]
Step 3: END
```

Heapsort

- Heapsort iki işlev gerçekleştirir: bunlar ekleme ve kök silmedir. Kökten çıkarılan her eleman dizinin sonuna yerleştirilir.

- Birinci adımda, heap inşa edildiğinde, yeni eleman için uygun pozisyonun aranması için yapılan karşılaştırma ağacın derinliğini geçemez.

- **H** bir tam ağaç olduğundan, **H**'ın derinliği m geçemez. Burada **m** heap'deki eleman sayısını temsil etmektedir.

Heapsort

- Böylelikle toplam karşılaştırma g(n), dizideki elemanı ekleme n H için aşağıdaki gibi sınırlanır:

$$g(n) \leq n \log n$$

- Heapsort çalışma zamanı **`O(nlogn)`** olarak ifade edilir.

Veri Yapıları ve Algoritmalar

ZAFER CÖMERT
Öğretim Üyesi