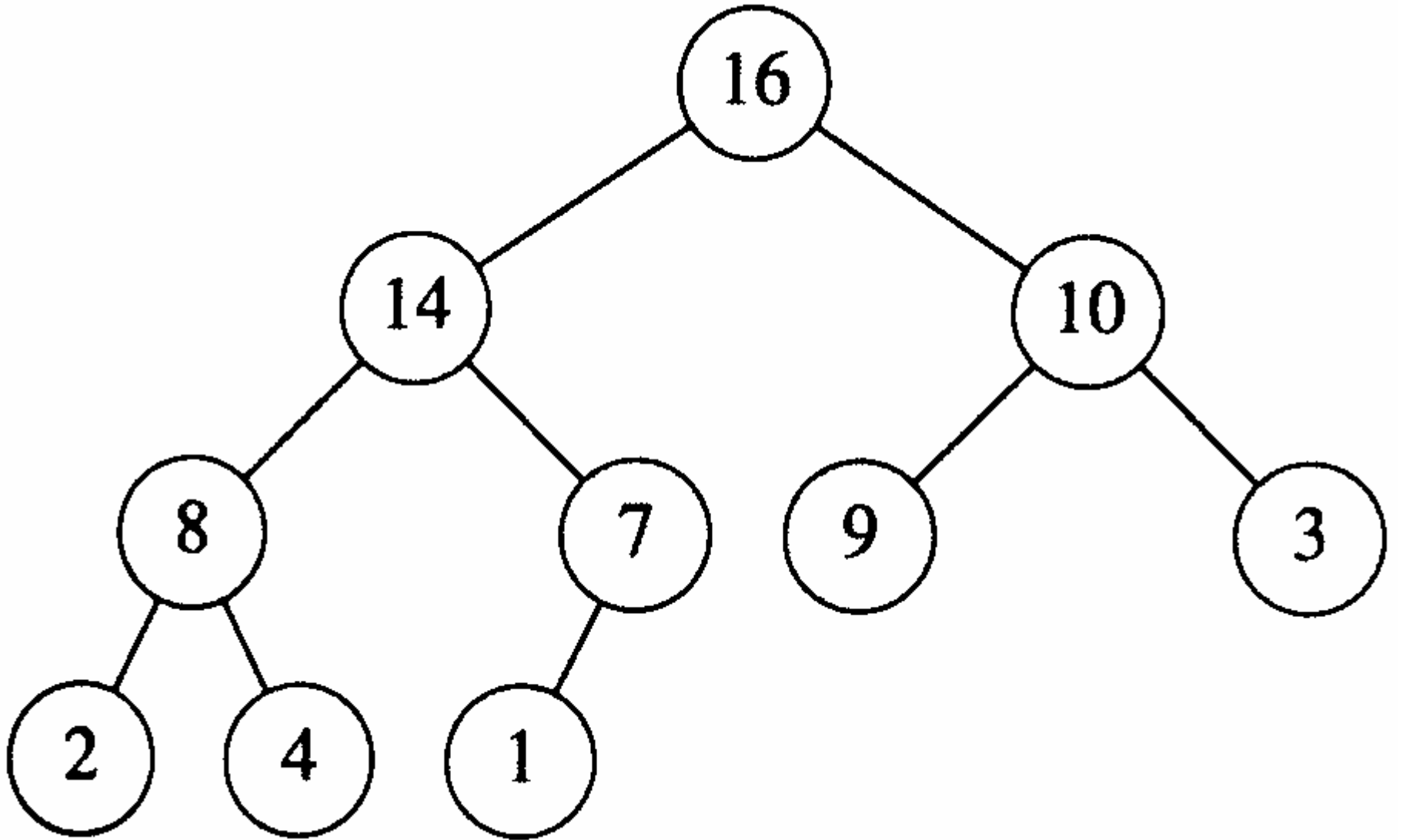


Bir İkili Ağaç Uygulaması

HeapSort

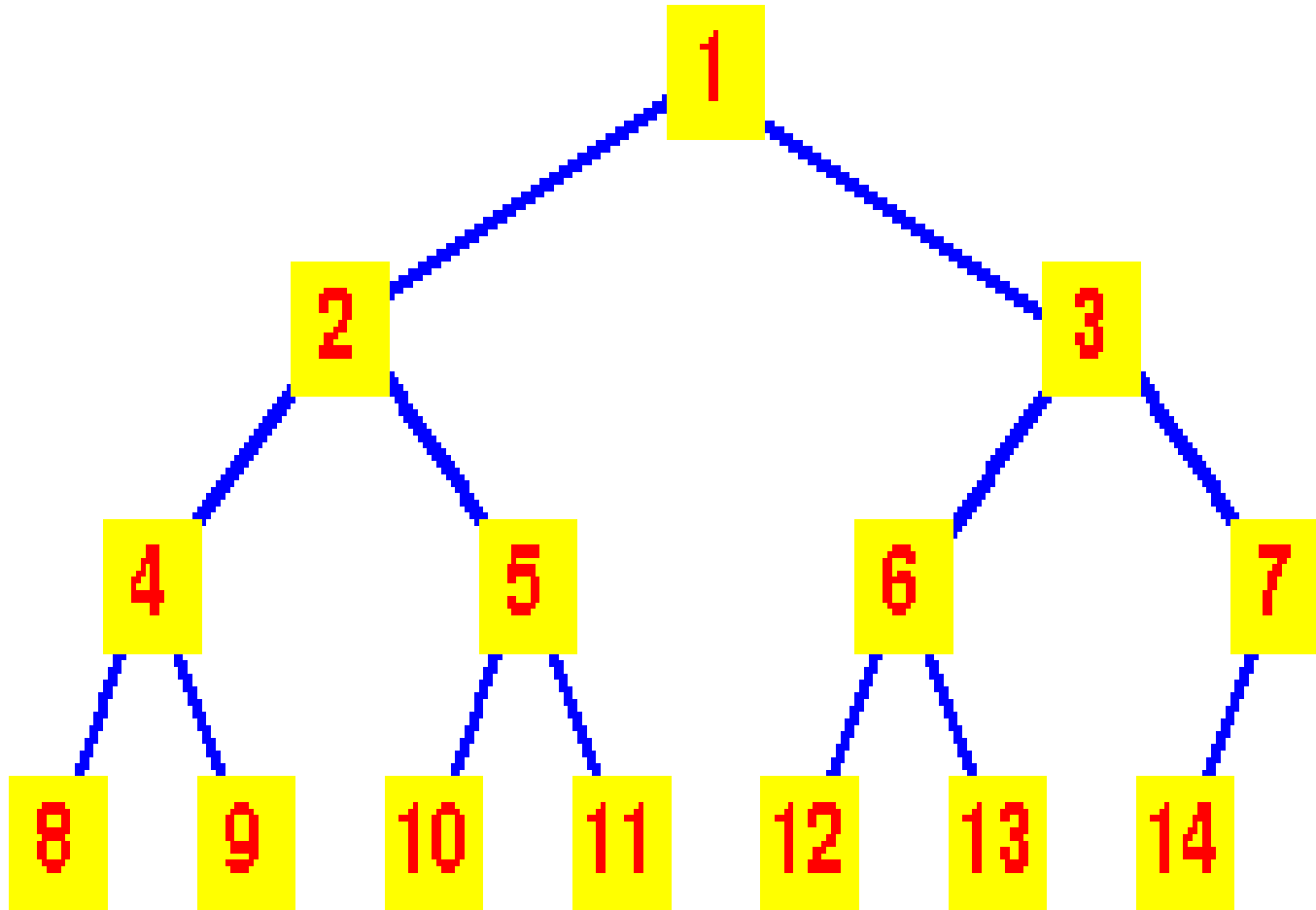
Heap özelliği

- Bir ikili ağacın heap özellikli olabilmesi için aşağıdaki iki özelliği sağlamalıdır.
 - a) Ağacın bütün seviyeleri (yapraklar hariç) doludur. Yani ağaç tam ve dengeli bir ağaçtır.
 - b) Bir düğüm çocuklarından ya büyüktür yada çocuklarına eşittir.



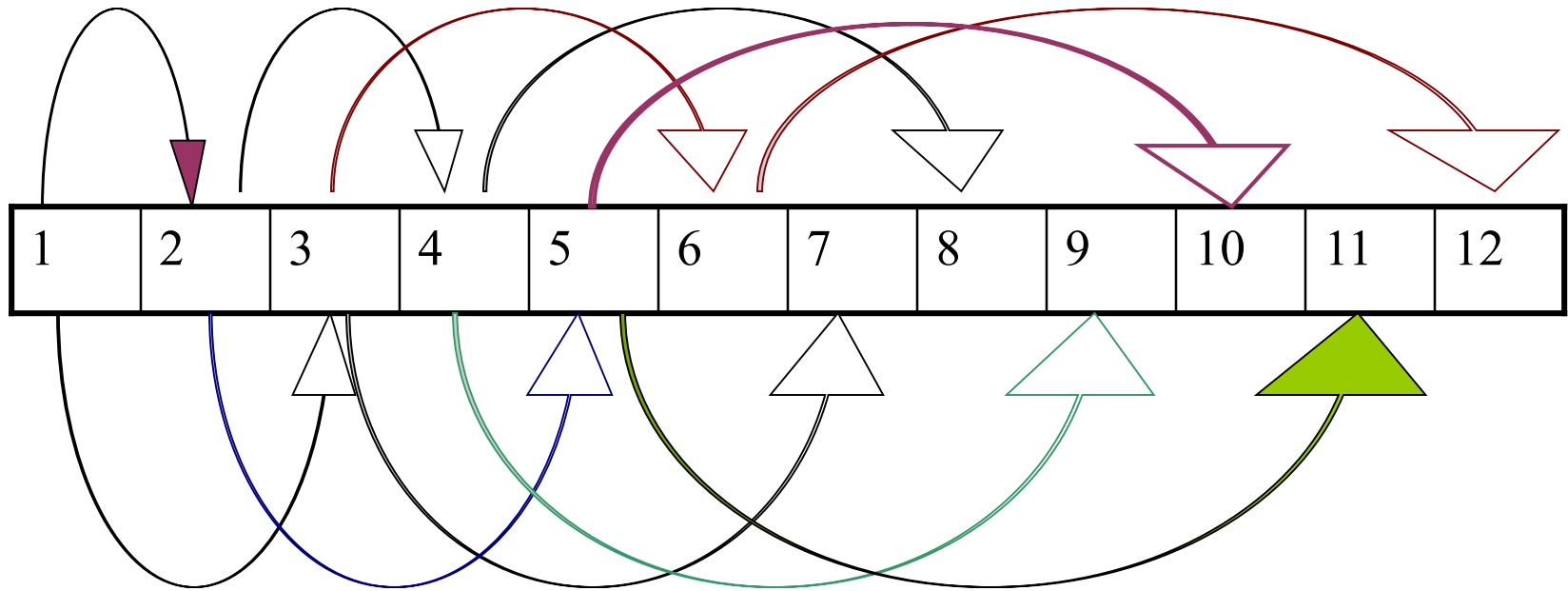
1)Her düğümün içeriği çocuklarından büyüktür.

2)Ağaç tam ikili ağaçtır.



Ağaç tam bir ikili ağaçtır. Ama hiçbir düğüm çocuklarından büyük değildir.

i indisinin sol çocukları $2*i$



i indisinin sağ çocukları $2*i+1$

Heap üzerindeki temel işlemler

```
int HeapTree::SolCocukIndeks (int dugumIndeks) {  
    return 2*dugumIndeks + 1;  
}  
int HeapTree::SagCocukIndeks (int dugumIndeks) {  
    return 2*dugumIndeks + 2;  
}  
int HeapTree::EbeveynDugumIndeks (int dugumIndeks) {  
    return (dugumIndeks-1)/2;  
}
```

Heapify (A, i)

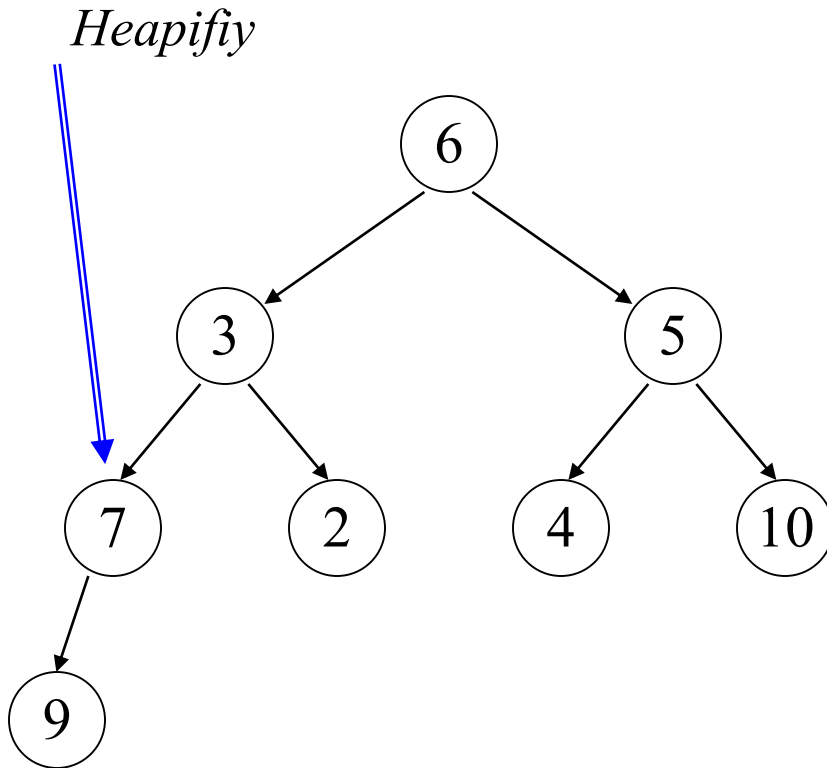
1. $l \leftarrow \text{solÇocuk}(i)$
2. $r \leftarrow \text{SağÇocuk}(i)$
3. if $l \leq \text{heap-size}[A]$ and $A[l] > A[i]$
4. then $\text{max} \leftarrow l$
5. else $\text{max} \leftarrow i$
6. if $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{max}]$
7. then $\text{max} \leftarrow r$
8. if $\text{max} \neq i$
9. then exchange $A[i] \leftrightarrow A[\text{max}]$
10. Heapify(A, max)

Build-Heap(A)

1. $heap\text{-}size[A] \leftarrow length[A]$
2. for $i \leftarrow \lfloor length[A]/2 \rfloor$ downto 1
3. do Heapify(A, i)

Bir diziyi heap ağacına dönüştürmek

6	3	5	7	2	4	10	9
0	1	2	3	4	5	6	7

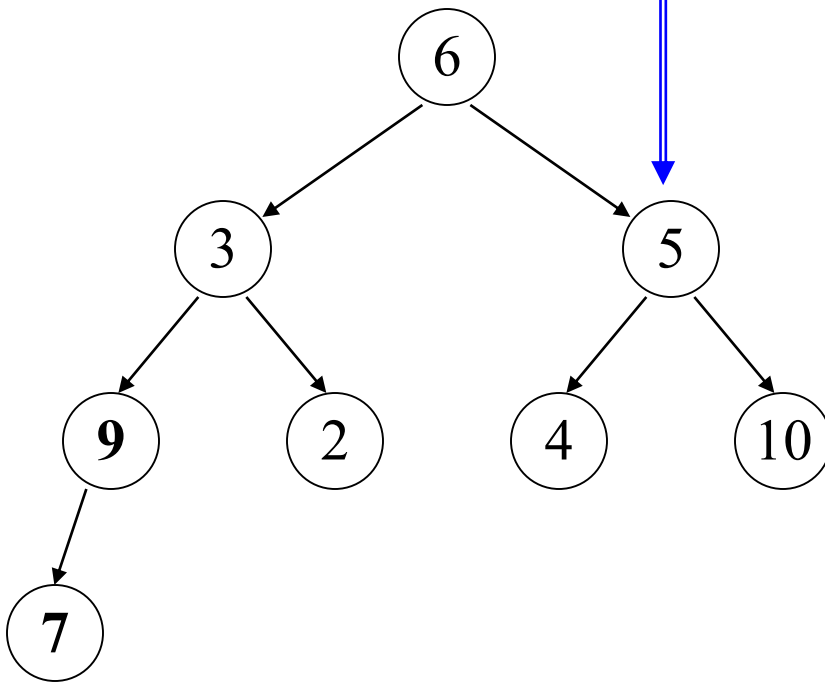


- Yukarıdaki dizi bir ikili ağaca şu şekilde dizilebilir.
- Yapraklar(2, 4, 9 ve 10) heap'tir.

Bir diziyi heap ağacına dönüştürmek

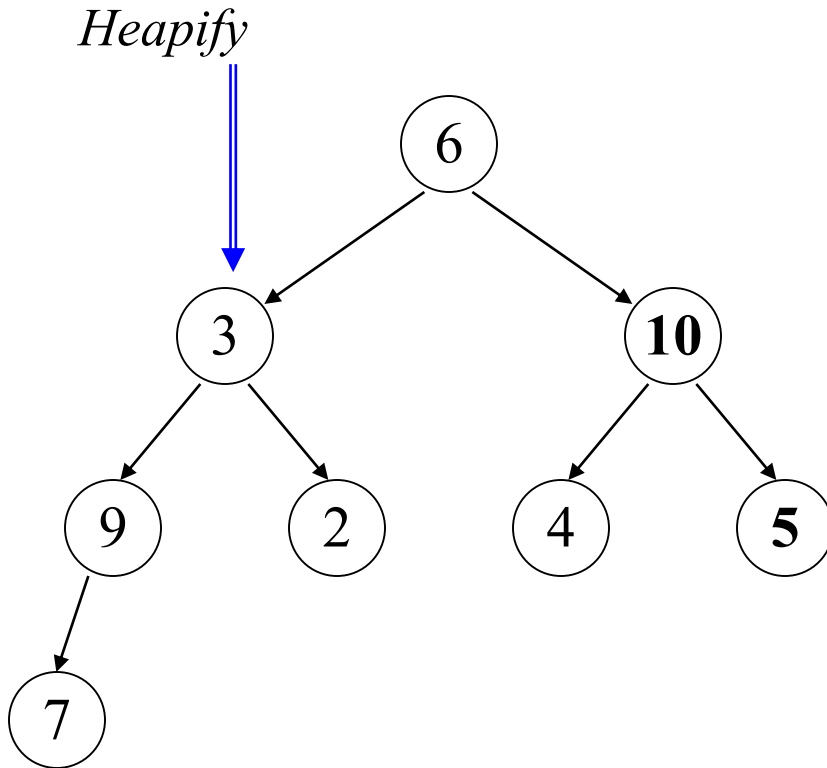
6	3	5	9	2	4	10	7
0	1	2	3	4	5	6	7

Heapify



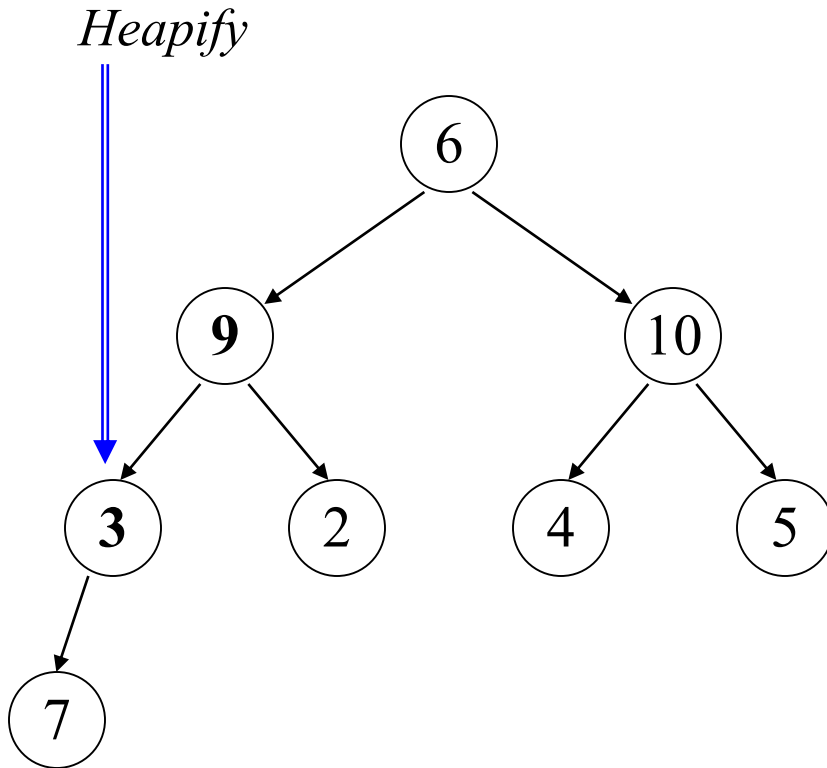
Bir diziyi heap ağacına dönüştürmek

6	3	10	9	2	4	5	7
0	1	2	3	4	5	6	7



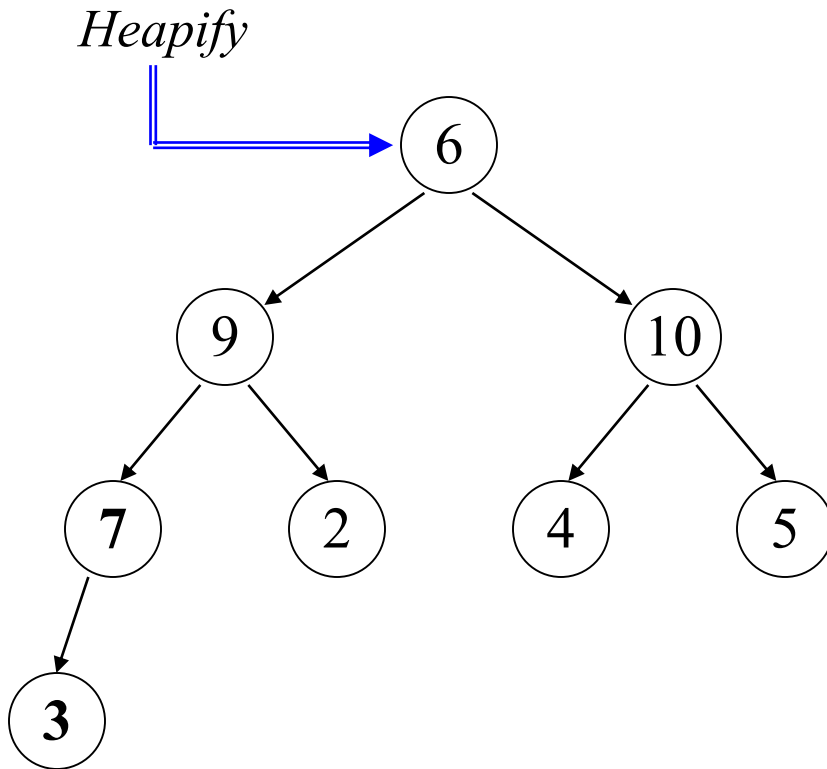
Bir diziyi heap ağacına dönüştürmek

6	9	10	3	2	4	5	7
0	1	2	3	4	5	6	7



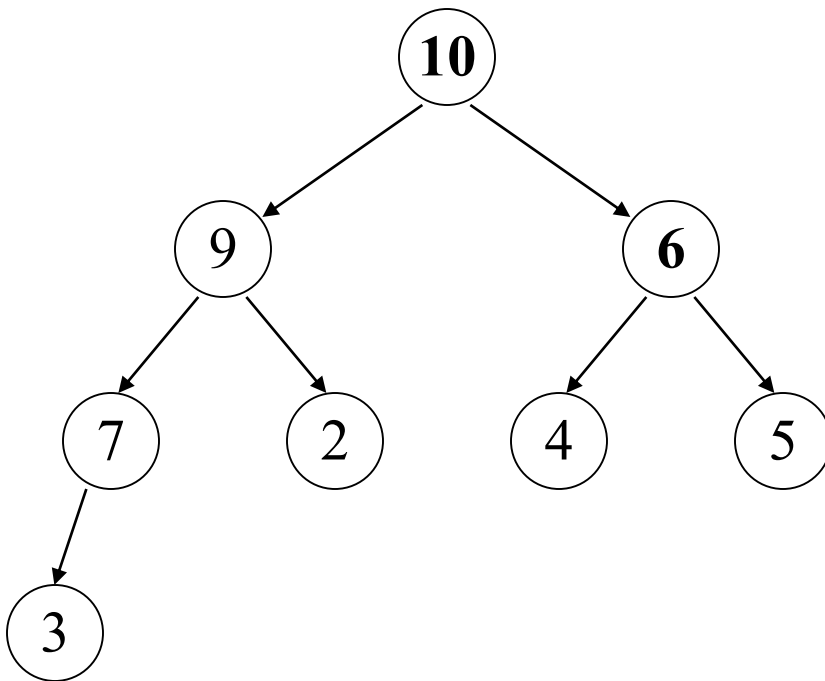
Bir diziyi heap ağacına dönüştürmek

6	9	10	7	2	4	5	3
0	1	2	3	4	5	6	7



Bir diziyi heap ağacına dönüştürmek

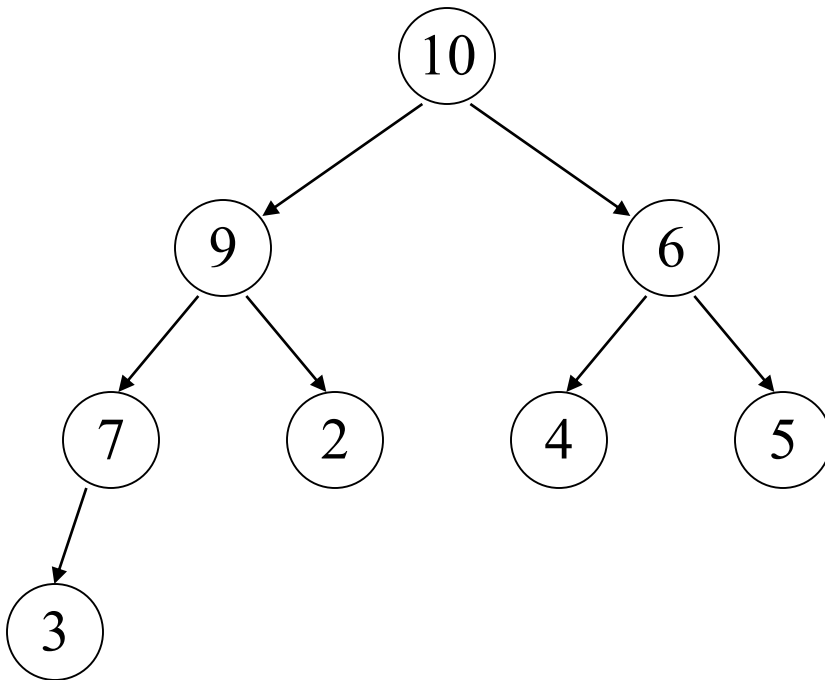
10	9	6	7	2	4	5	3
0	1	2	3	4	5	6	7



Heap Sıralama

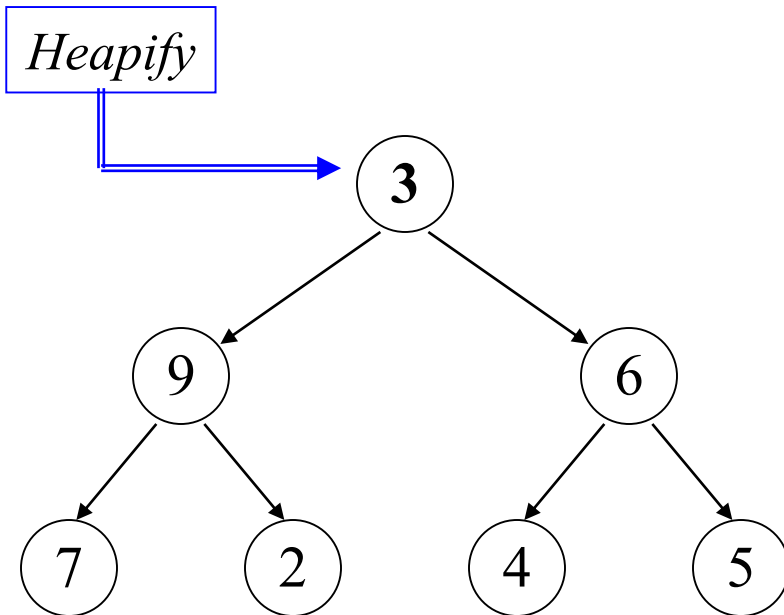
	0	1	2	3	4	5	6	7
a[]:	10	9	6	7	2	4	5	3

└────────────────── Heap ─────────────────┘



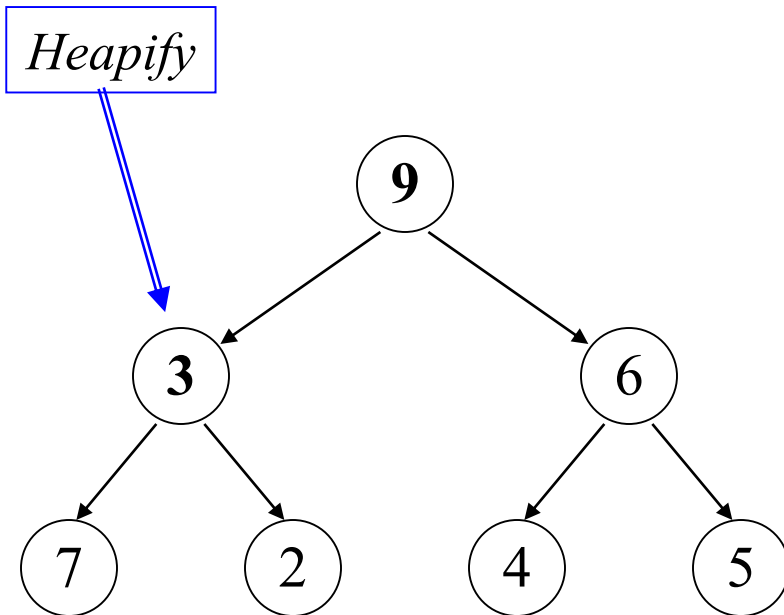
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	3	9	6	7	2	4	5	10
	Heap değil						sıralı	



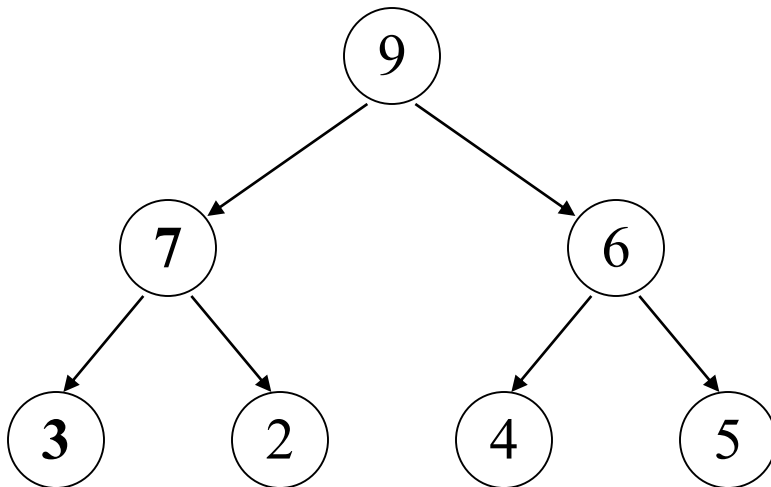
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	9	3	6	7	2	4	5	10
	heapify					Sıralı		



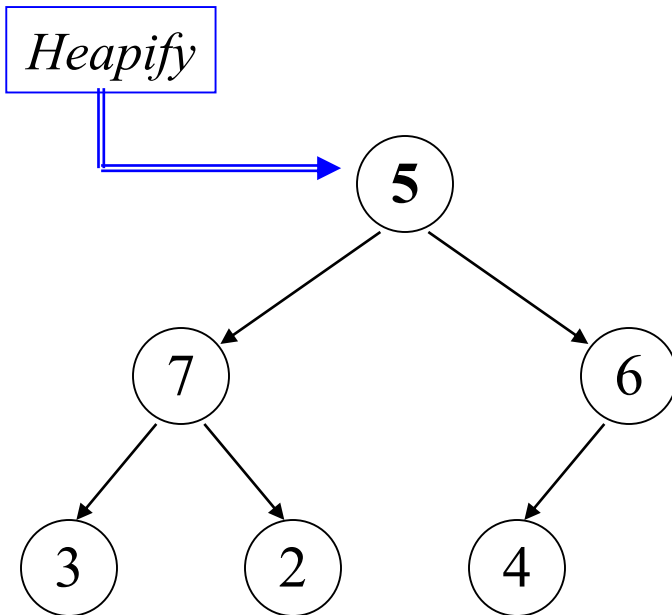
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	9	7	6	3	2	4	5	10
	Heap						Sıralı	



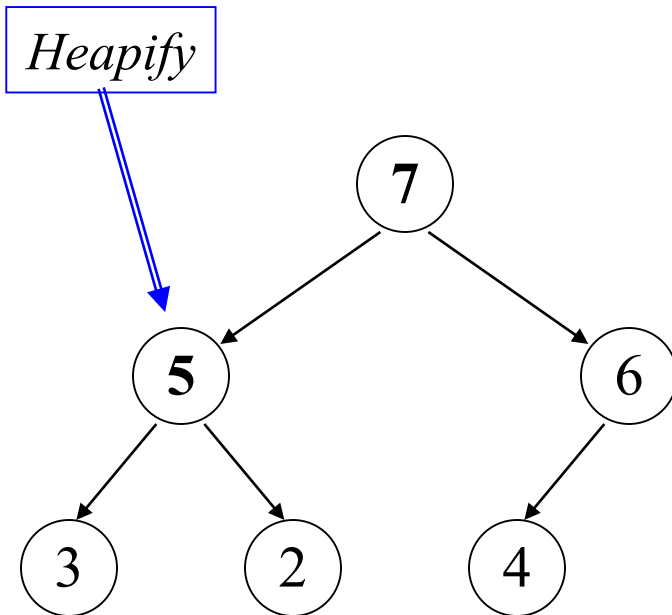
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	5	7	6	3	2	4	9	10
	Heap değil						Sıralı	



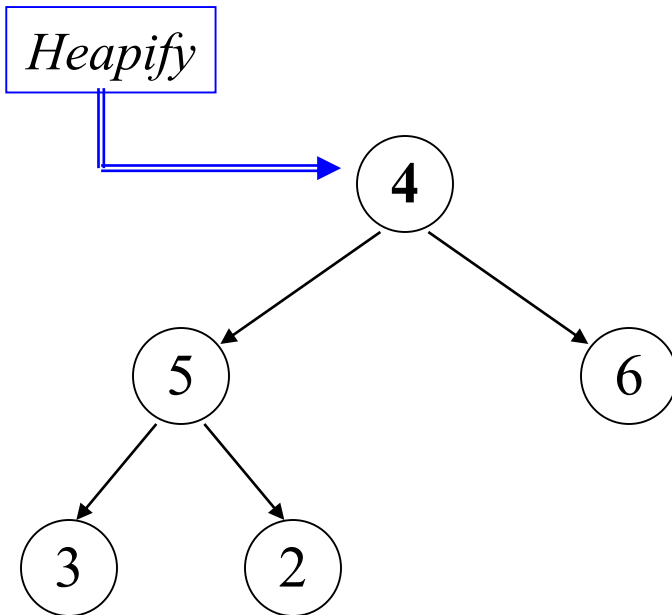
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	7	5	6	3	2	4	9	10
	Heap					Sıralı		



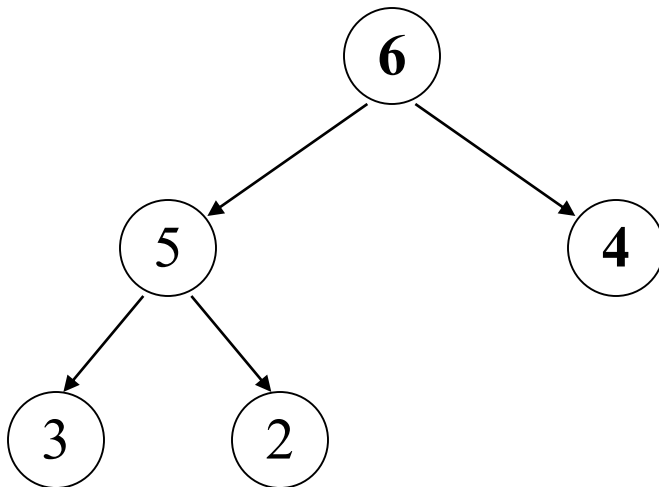
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	4	5	6	3	2	7	9	10
	Heap değil					Sıralı		



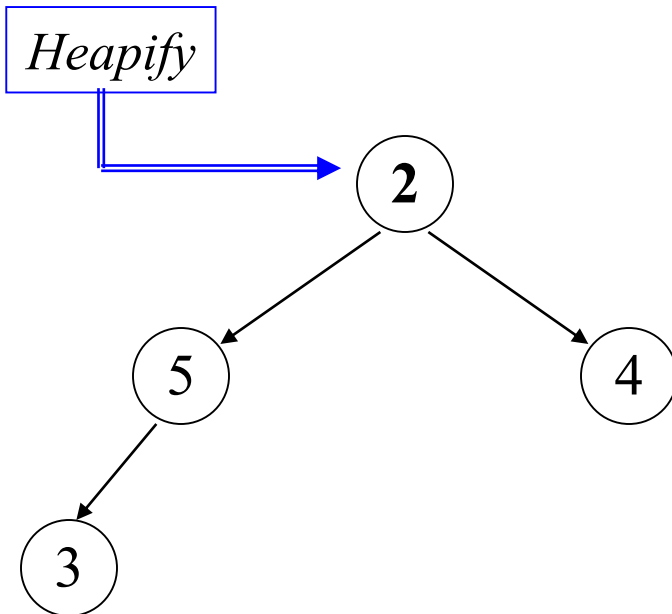
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	6	5	4	3	2	7	9	10
	Heap					Sıralı		



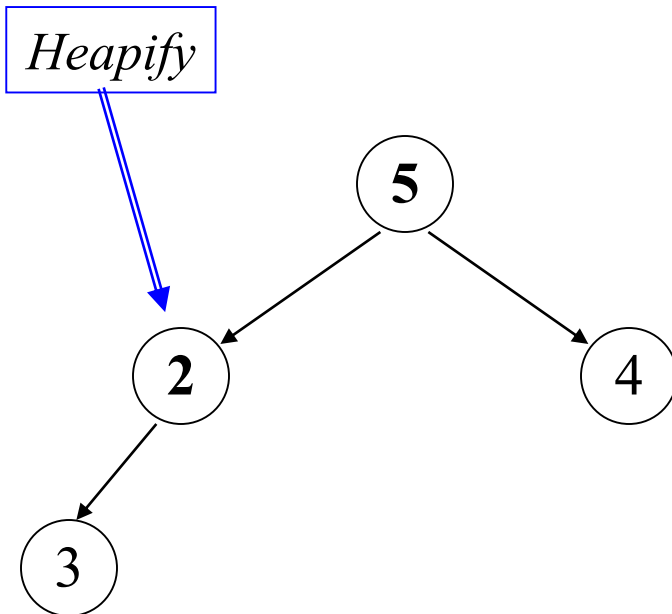
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	2	5	4	3	6	7	9	10
	Heap değil				Sıralı			



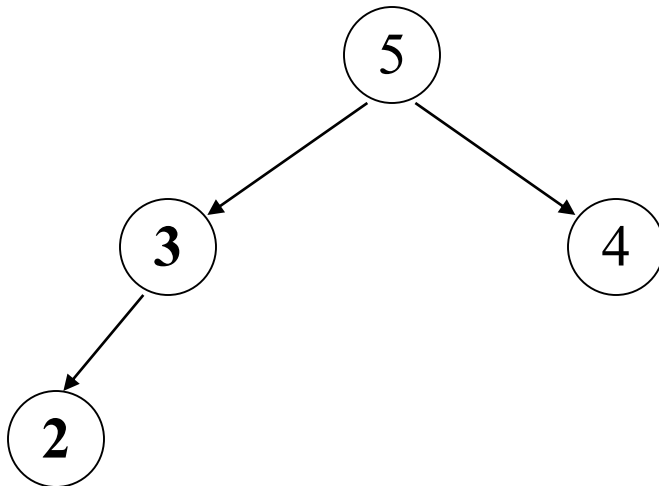
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	5	2	4	3	6	7	9	10
	Heapify				Sıralı			



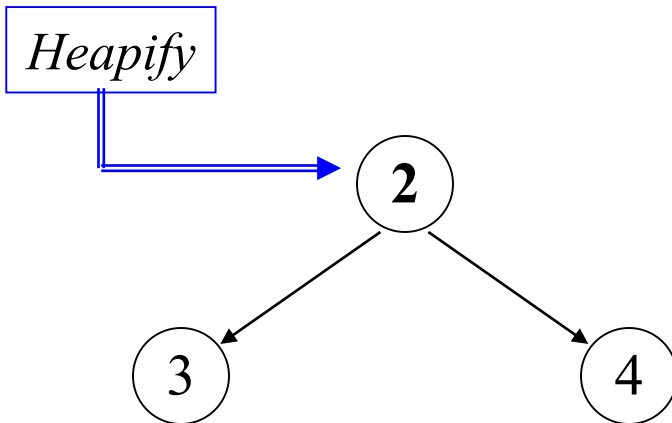
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	5	3	4	2	6	7	9	10
	Heap				Sıralı			



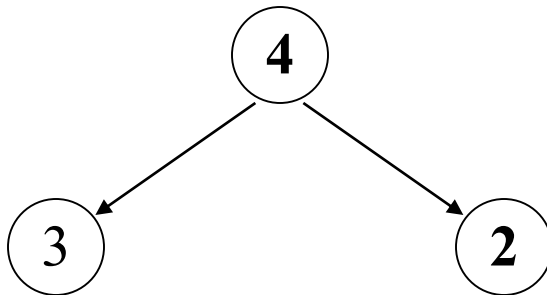
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	2	3	4	5	6	7	9	10
	Heap değil			Sıralı				



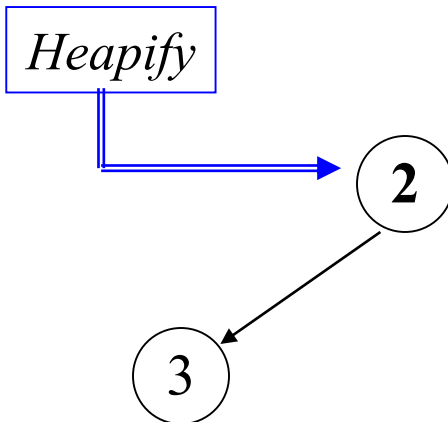
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	4	3	2	5	6	7	9	10
	Heap			Sıralı				



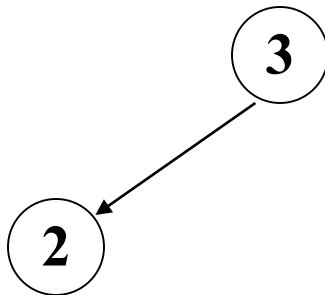
Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	2	3	4	5	6	7	9	10
	Heap değil		Sıralı					



Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	3	2	4	5	6	7	9	10
	Heap		Sıralı					



Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	2	3	4	5	6	7	9	10
	Heap			Sıralı				

2

Heap Sıralama

	0	1	2	3	4	5	6	7
a[]:	2	3	4	5	6	7	9	10
	Sıralı							

Hatırlatma

	<i>Best case</i>	<i>Average case</i>	<i>Worst case</i>
Selection sort	n^2	n^2	n^2
Bubble sort	n	n^2	n^2
Insertion sort	n	n^2	n^2
Mergesort	$n * \log_2 n$	$n * \log_2 n$	$n * \log_2 n$
Quicksort	$n * \log_2 n$	$n * \log_2 n$	n^2
Heapsort	$n * \log_2 n$	$n * \log_2 n$	$n * \log_2 n$