Heap Sort

Heapsort Algorithm

Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```

Build Max Heap

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

Heap

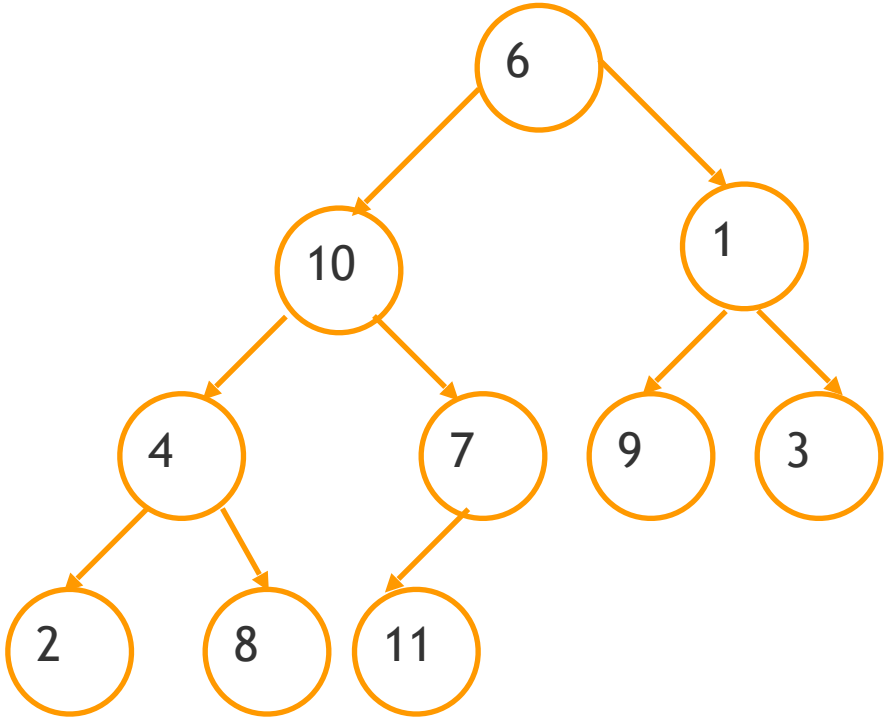
- The root of the tree is A[1], and given the index i of a node, we can easily compute the indices of its parent, left child, and right child:

```
def parent(i):
    return i/2

def left(i):
    try:
        return 2*i
    except:
        pass

def right(i):
    try:
        return 2 *i + 1
    except:
        pass
```

Max-Heapify

```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```

Start with an array (it is not a max heap)

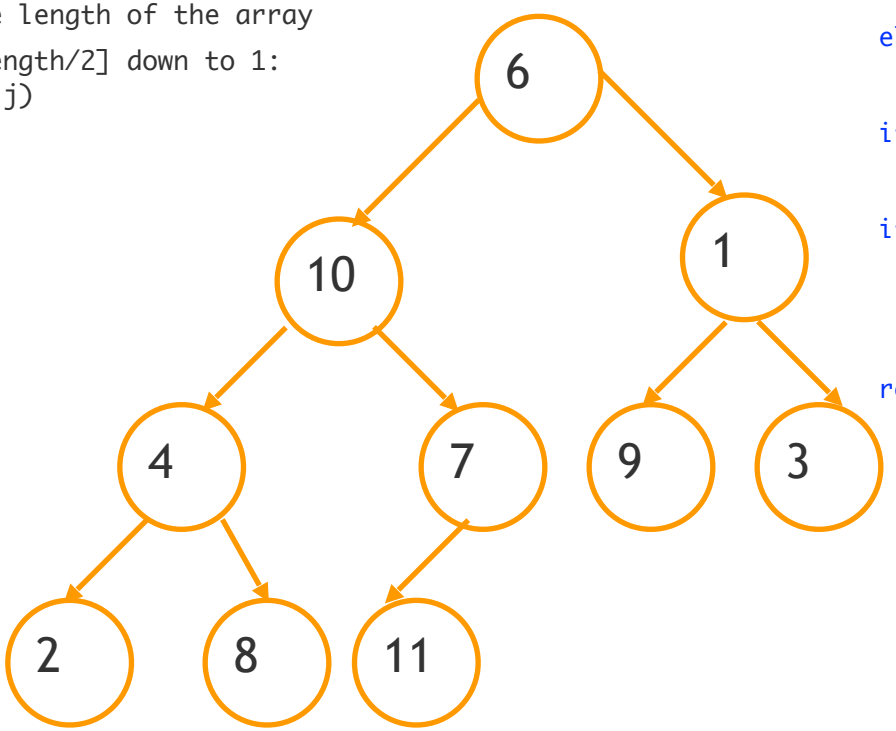


6	10	1	4	7	9	3	2	8	11

Exchange 7 and 11

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

```
def parent(i):
      return i/2

  def left(i):
      try:
          return 2*i
      except:
          pass

  def right(i):
      try:
          return 2 *i + 1
      except:
          pass
```



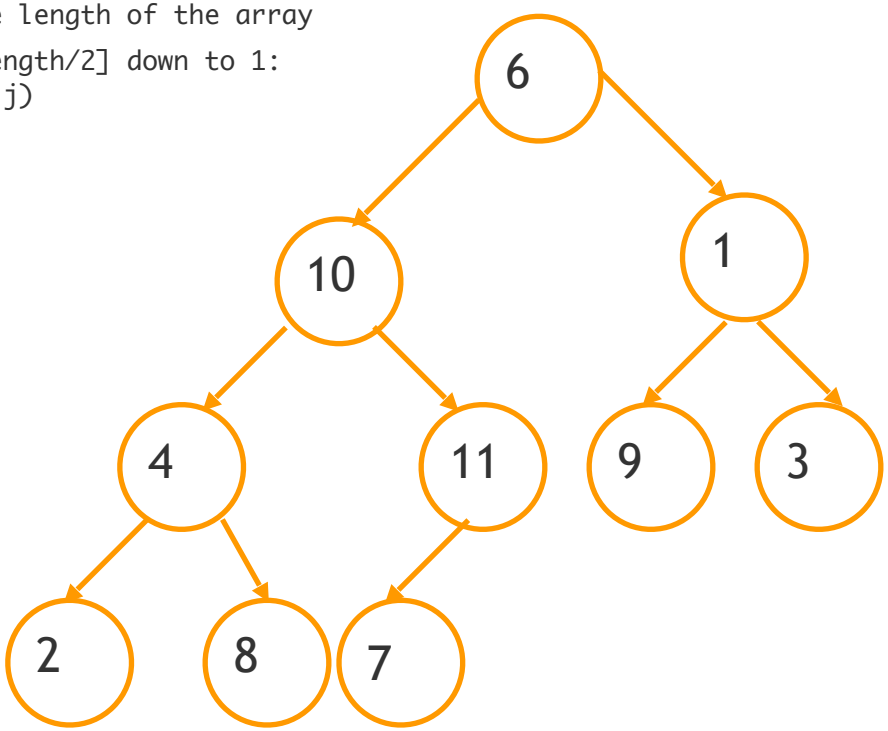
6	10	1	4	7	9	3	2	8	11

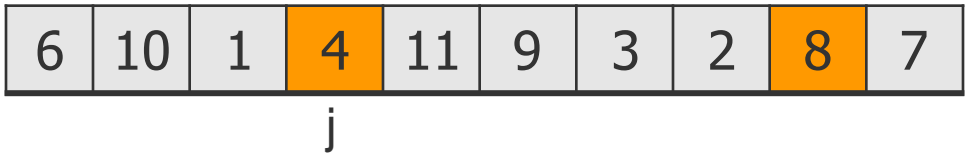
j

```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```

Exchange 4 and 8

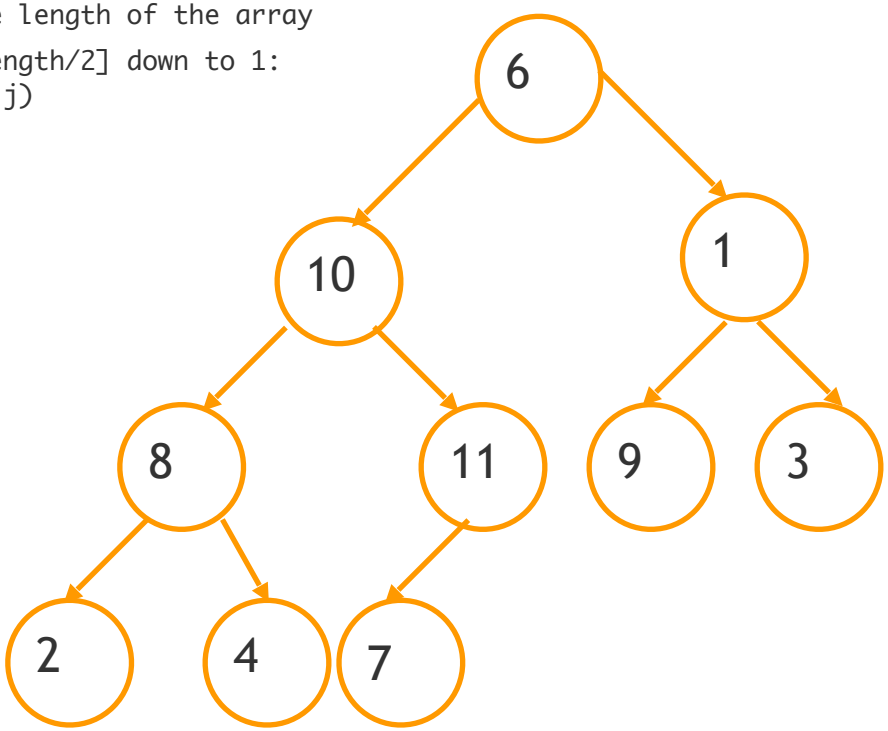
```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```





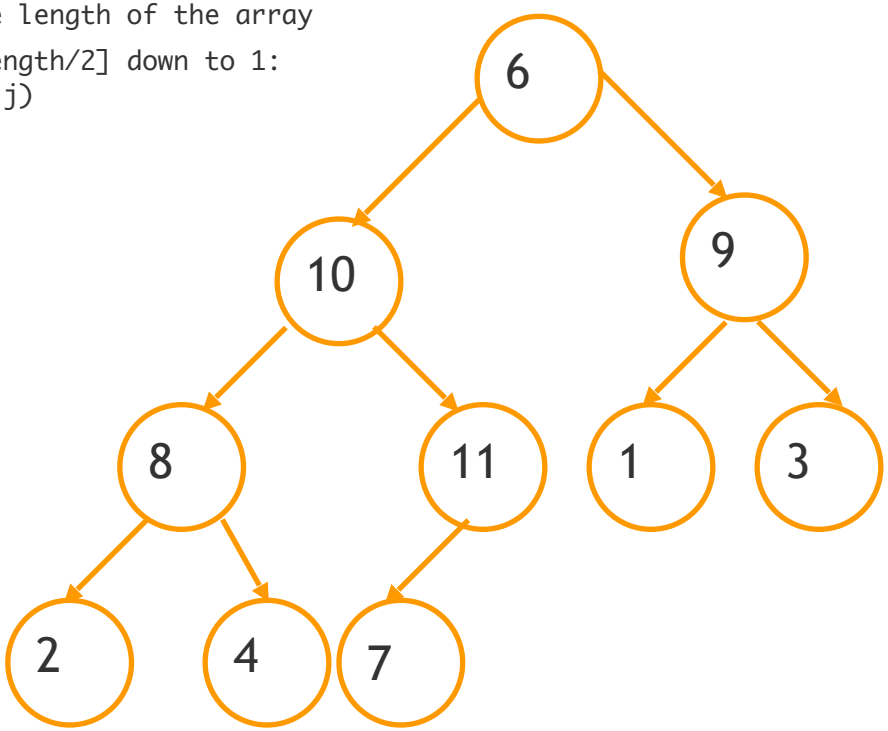
Exchange 9 and 1

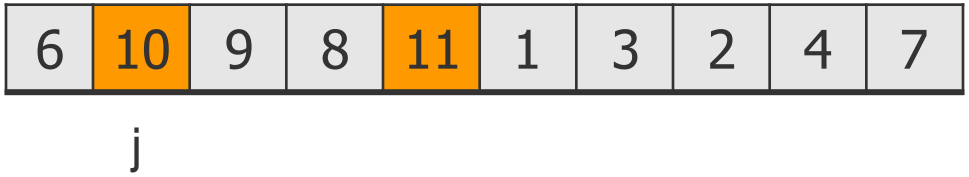
```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```



Exchange 10 and 11

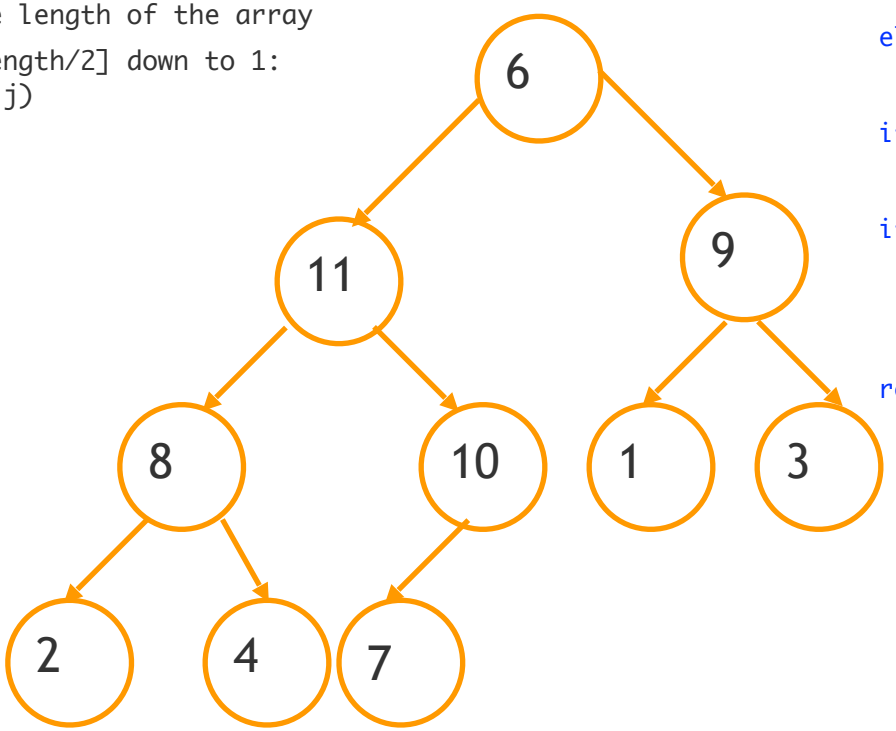
```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

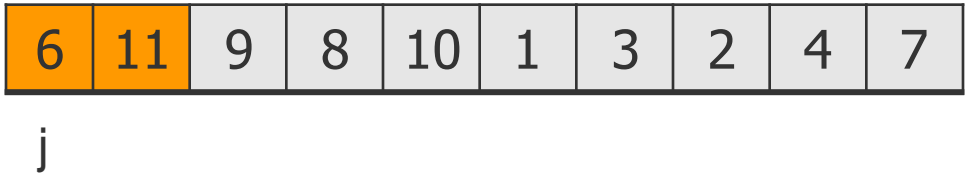




Exchange 6 and 11

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```





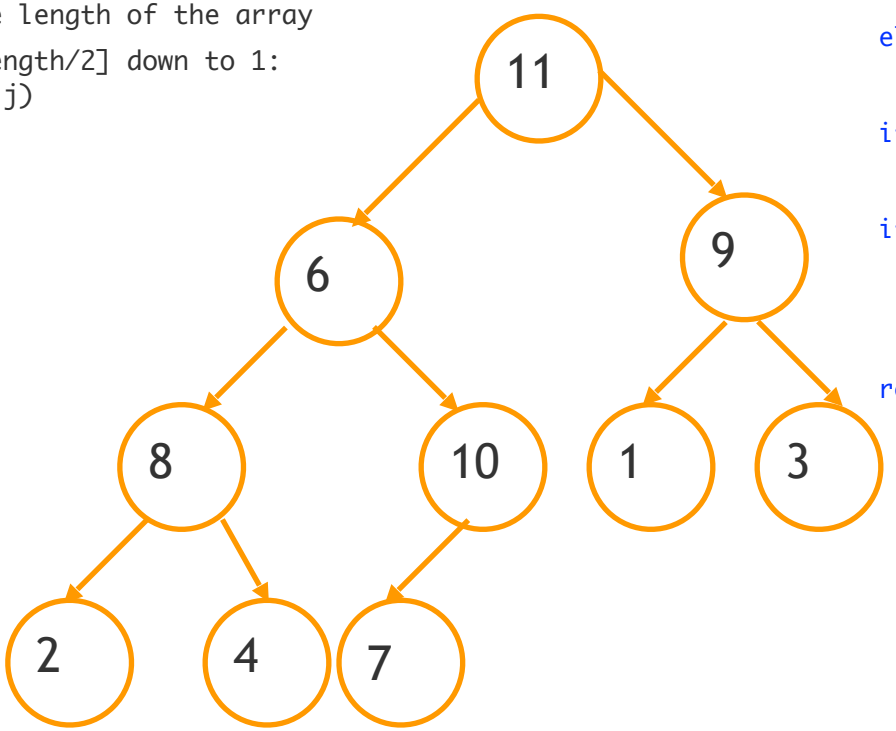
j

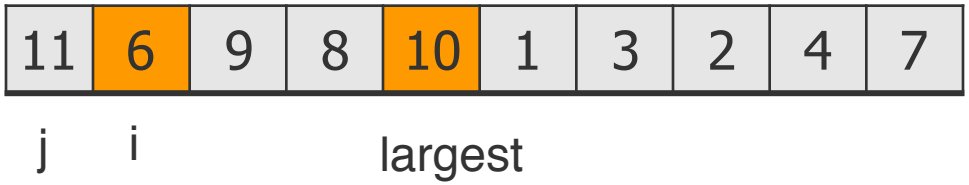
```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```

Exchange 6 and 10

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```

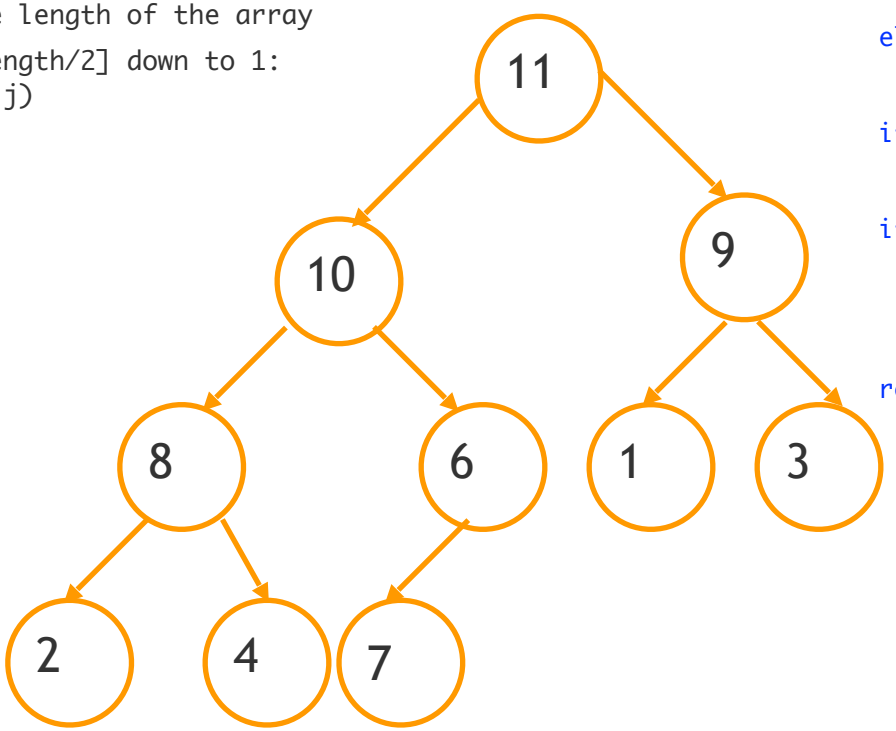




Exchange 6 and 7

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

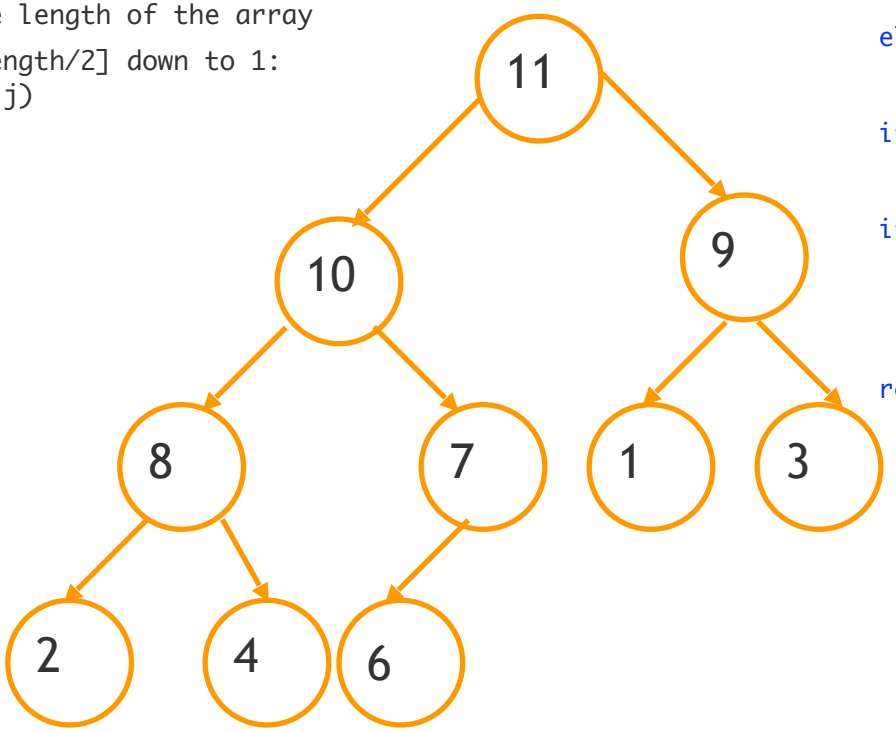
```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```



max_heapify

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```



11	10	9	8	7	1	3	2	4	6

j i largest

Sorting

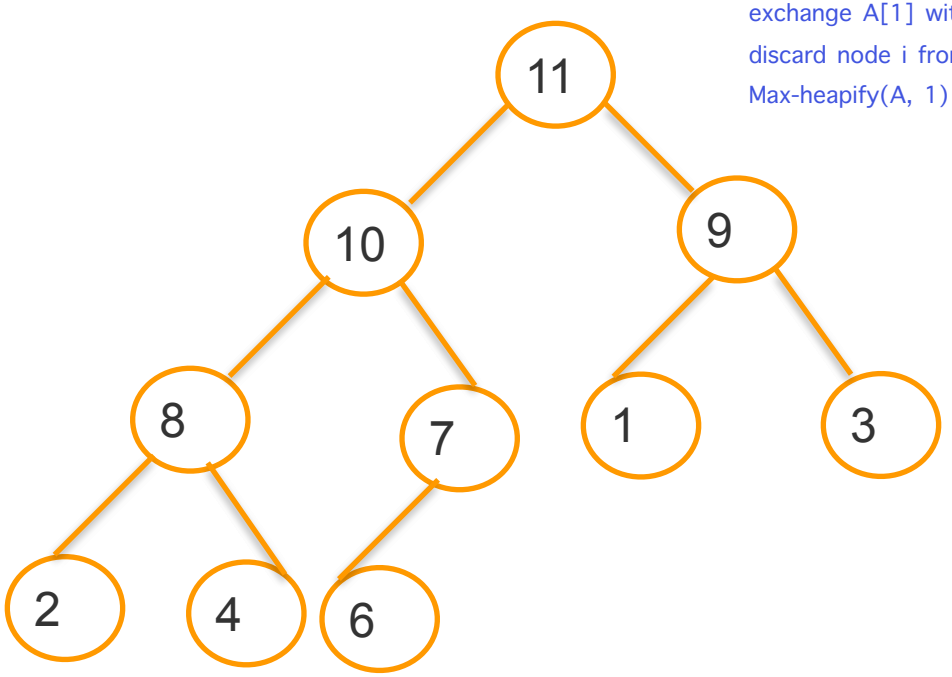
Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A,1) because new root may violate max heap property
```

Exchange 11 and 6

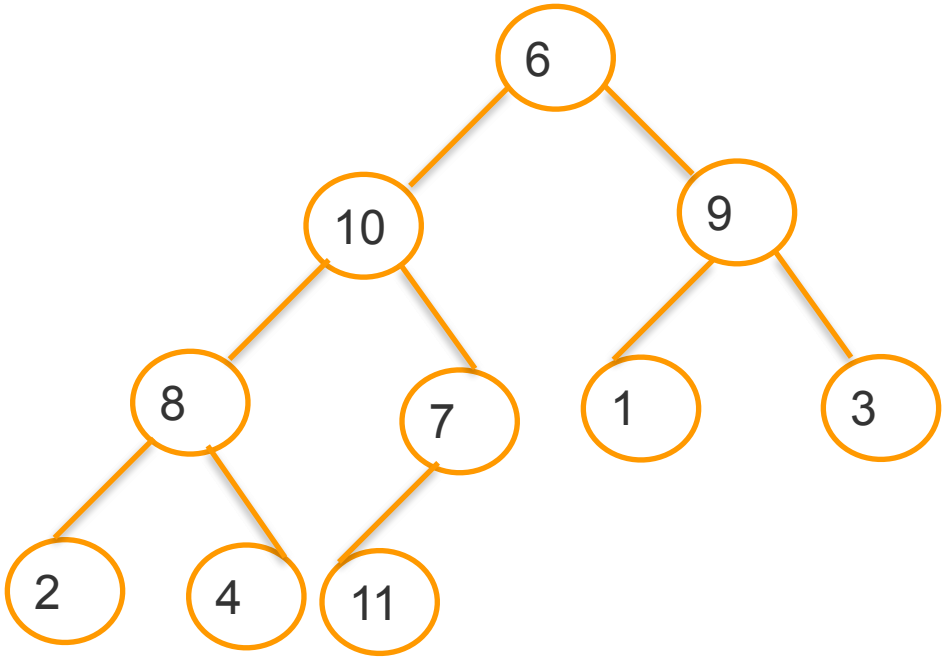
Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```



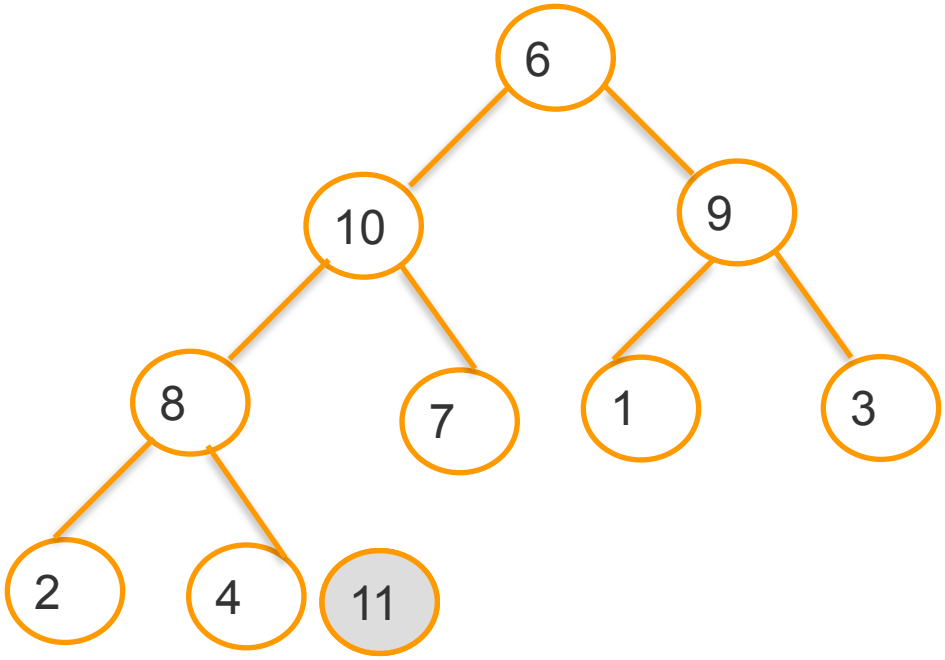
11	10	9	8	7	1	3	2	4	6

Remove 11 from the heap



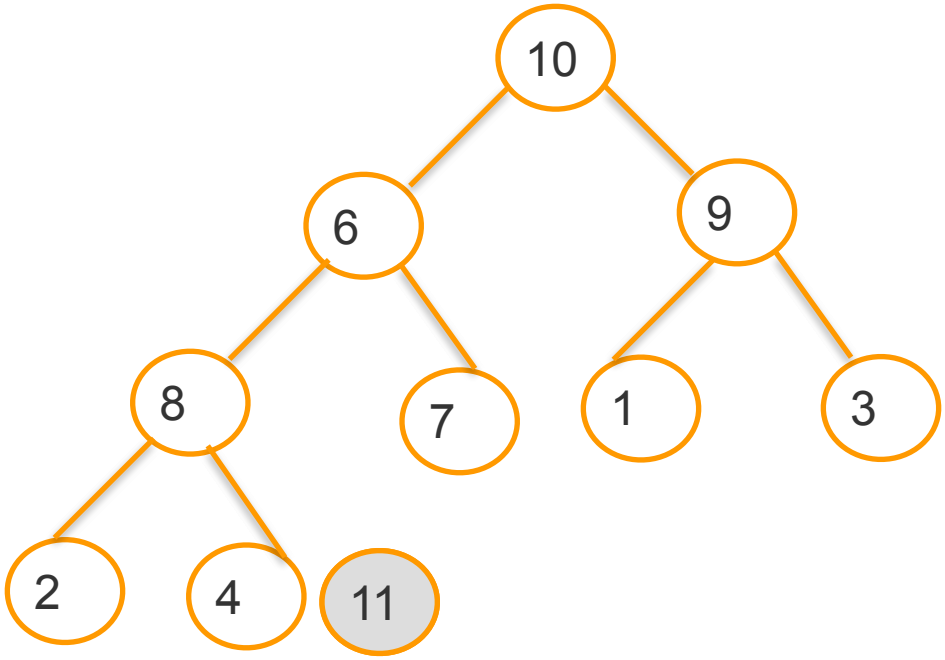
6	10	9	8	7	1	3	2	4	11

Swap 6 and 10



6	10	9	8	7	1	3	2	4	11

Exchange 6 and 8

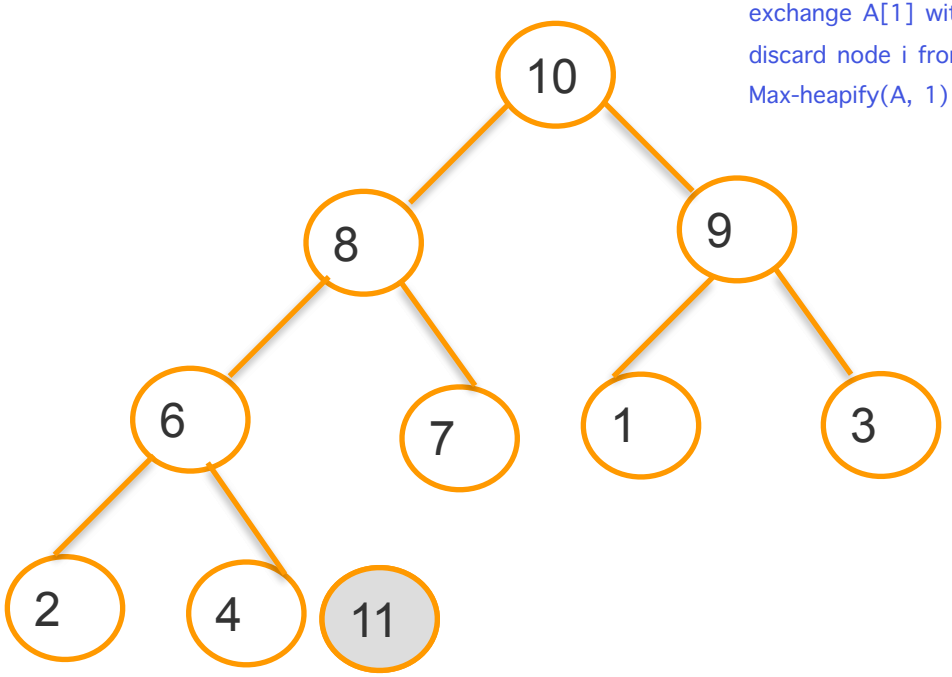


10	6	9	8	7	1	3	2	4	11

Exchange 10 and 4

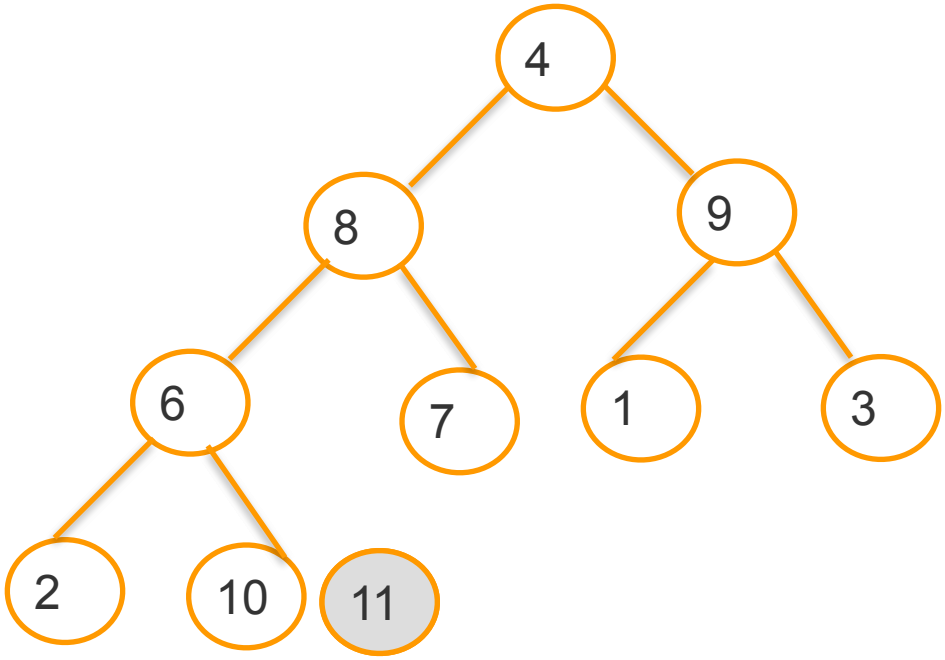
Heapsort(A):

#Create max heap
Build_Max_Heap from unordered array A
Finish sorting
iterate i from A.length downto 2
    exchange A[1] with A[i]
    discard node i from heap (decrement heap size)
    Max-heapify(A, 1) because new root may violate max heap property



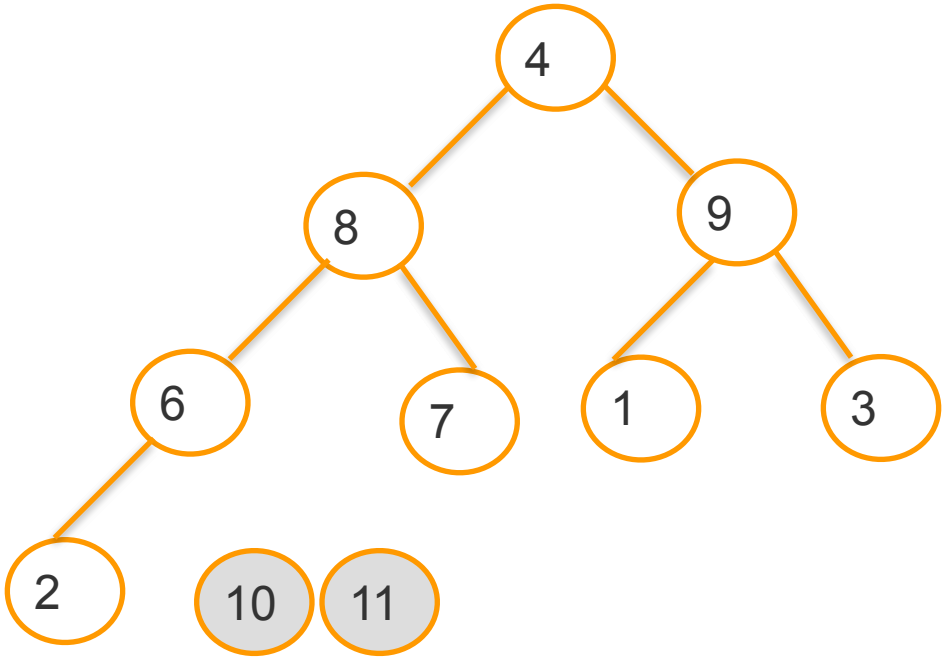
10	8	9	6	7	1	3	2	4	11

Remove 10 from the heap



4	8	9	6	7	1	3	2	10	11

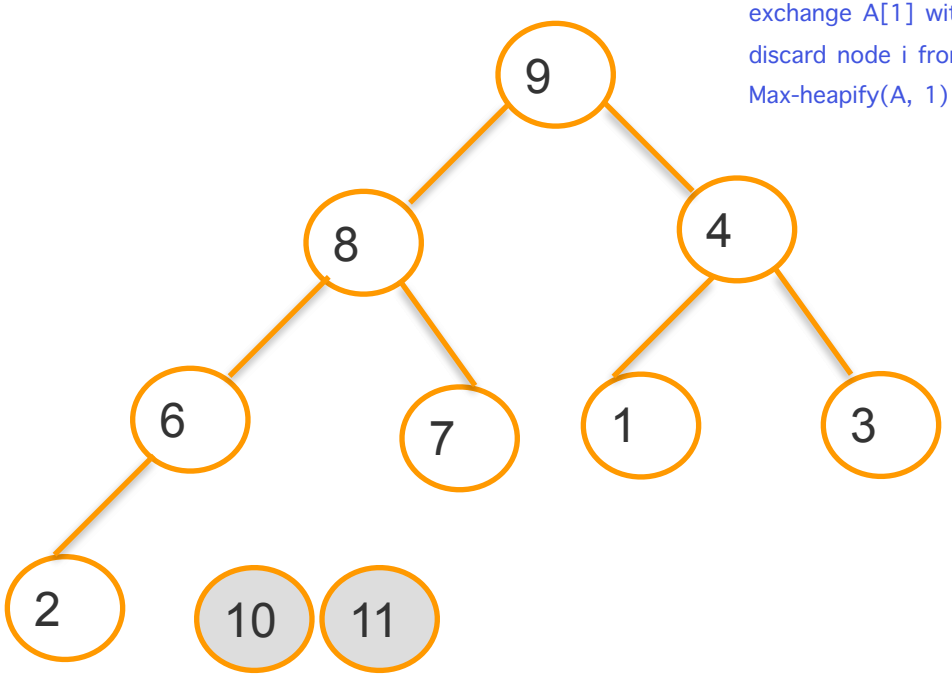
Exchange 4 and 9



Exchange 9 and 2

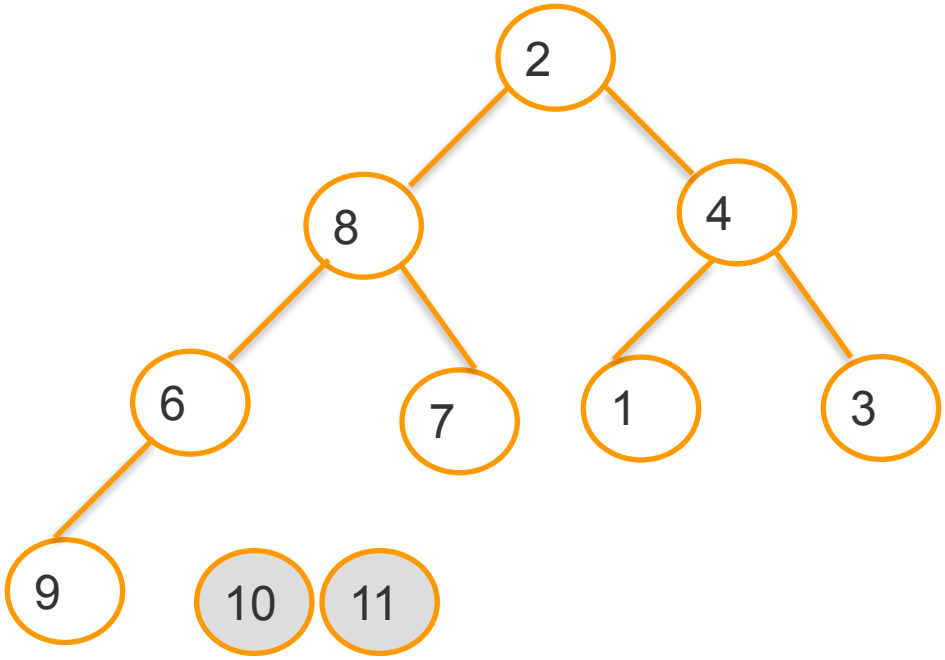
Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```



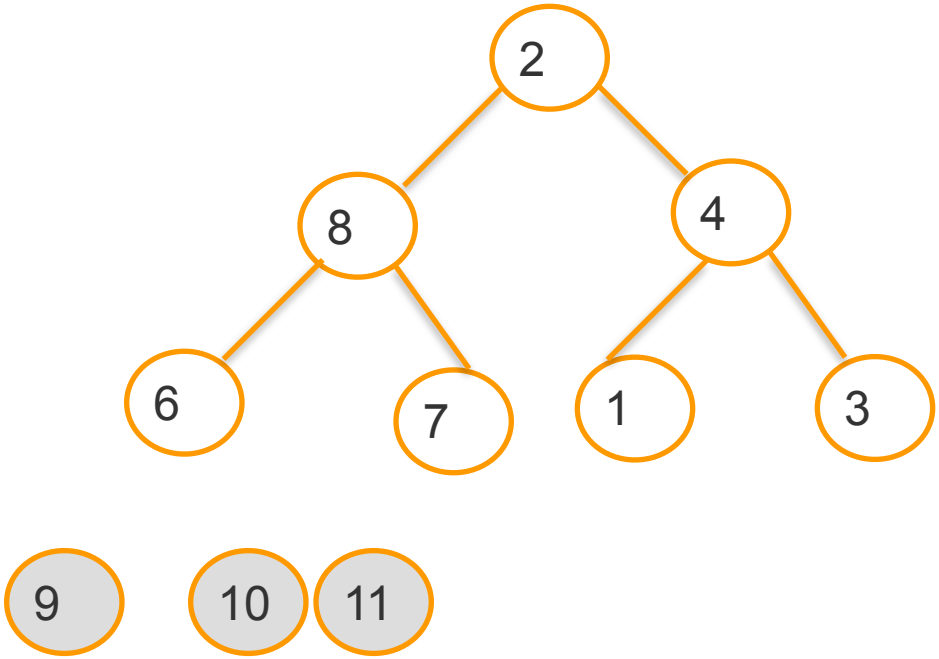
9	8	4	6	7	1	3	2	10	11

Remove 9 from the heap

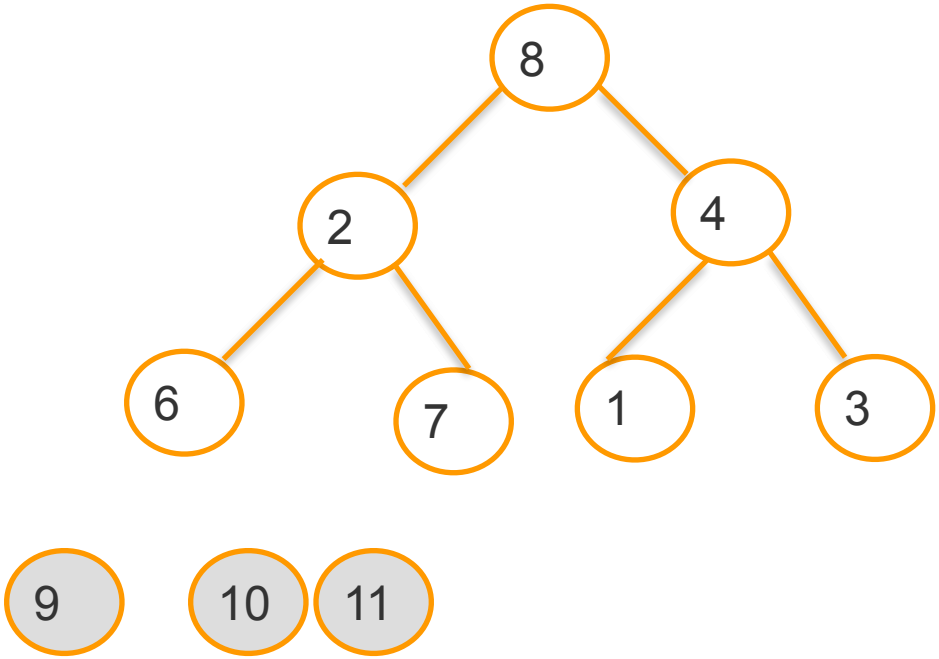


2	8	4	6	7	1	3	9	10	11

Exchange 2 and 8



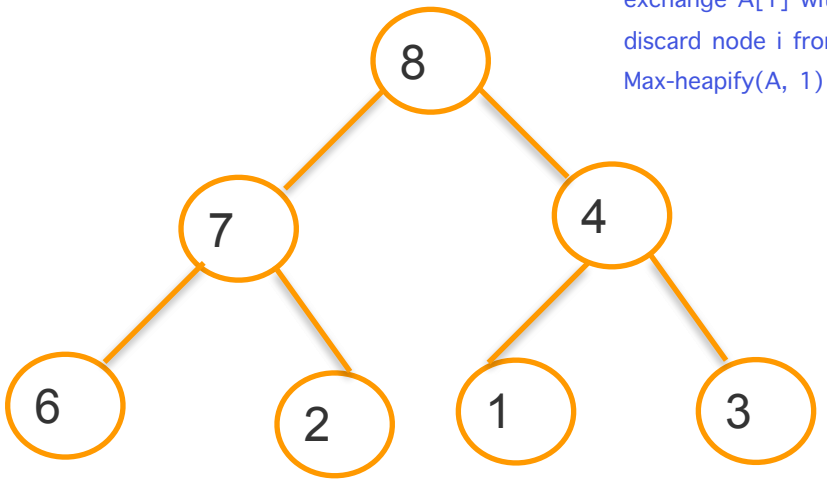
Exchange 2 and 7



Exchange 8 and 3

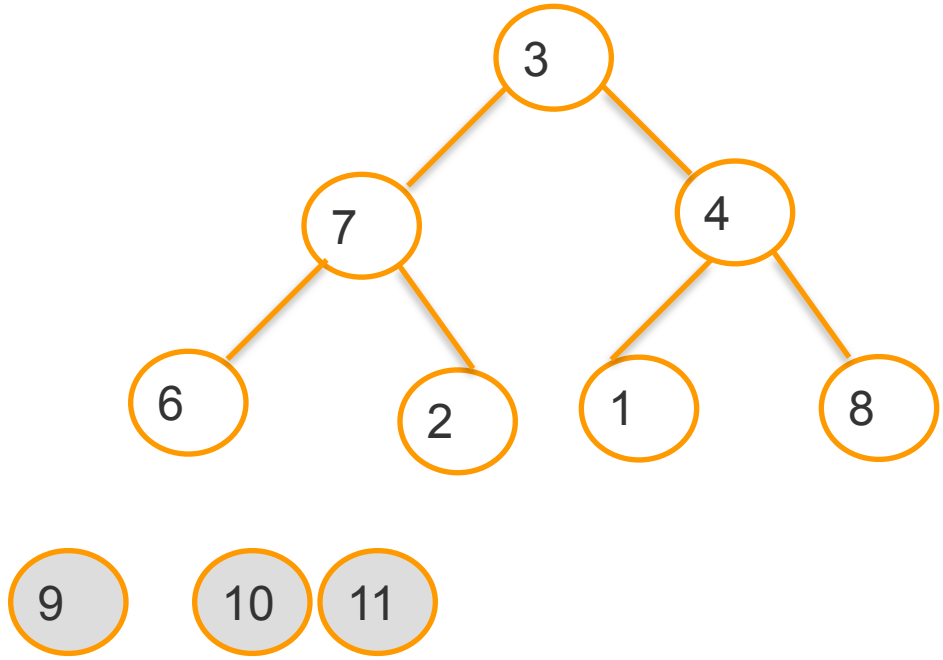
Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```



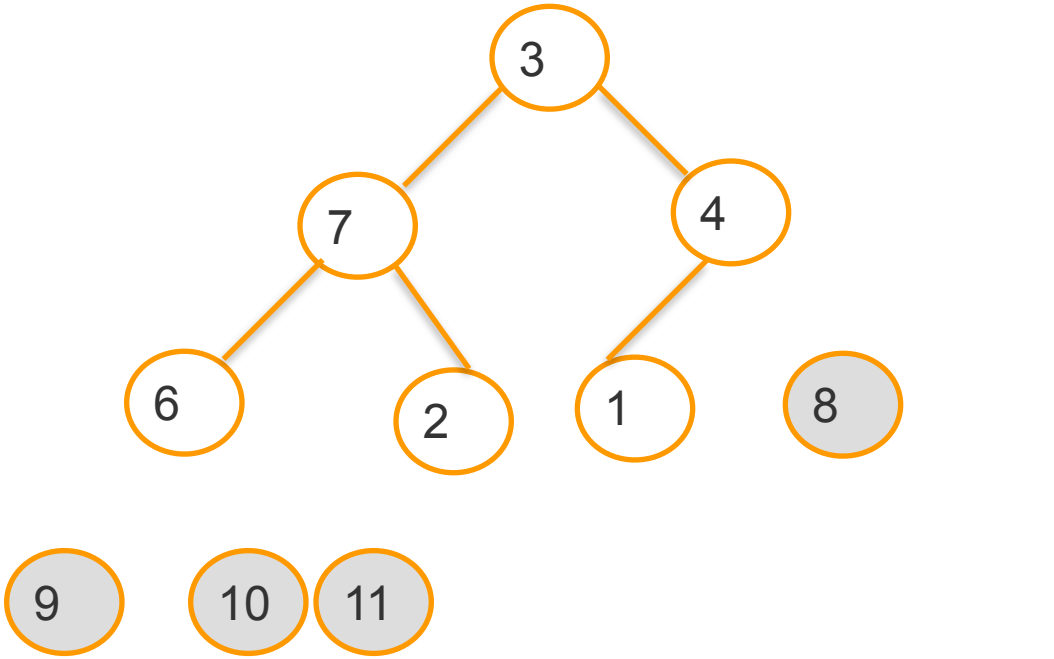
8	7	4	6	2	1	3	9	10	11

Remove 8 from the heap



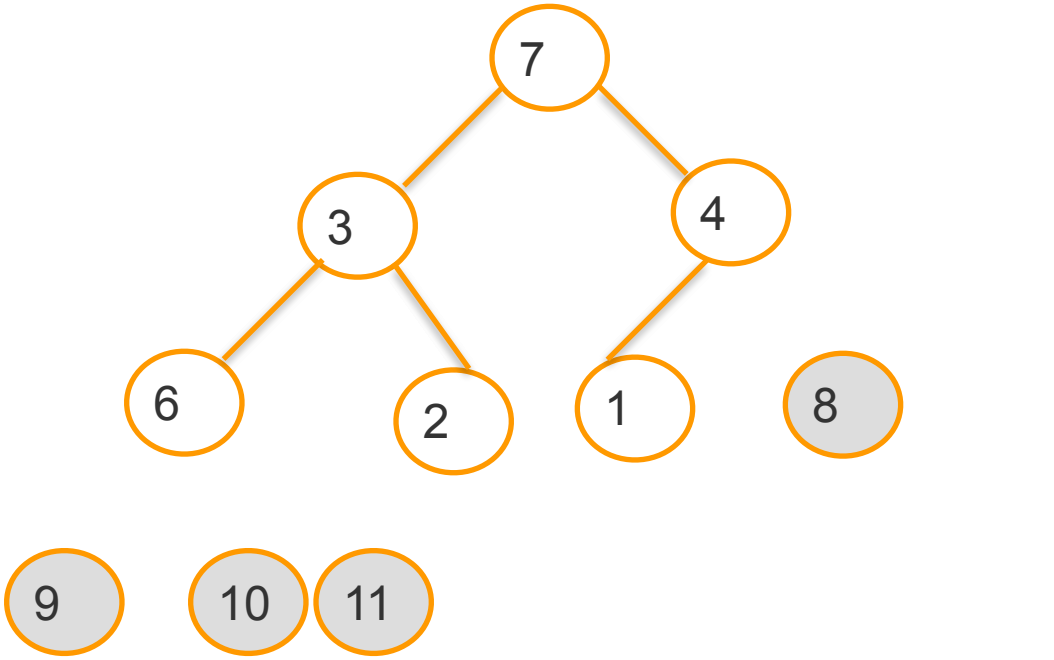
3	7	4	6	2	1	8	9	10	11

Exchange 3 and 7



3	7	4	6	2	1	8	9	10	11

Exchange 3 and 6

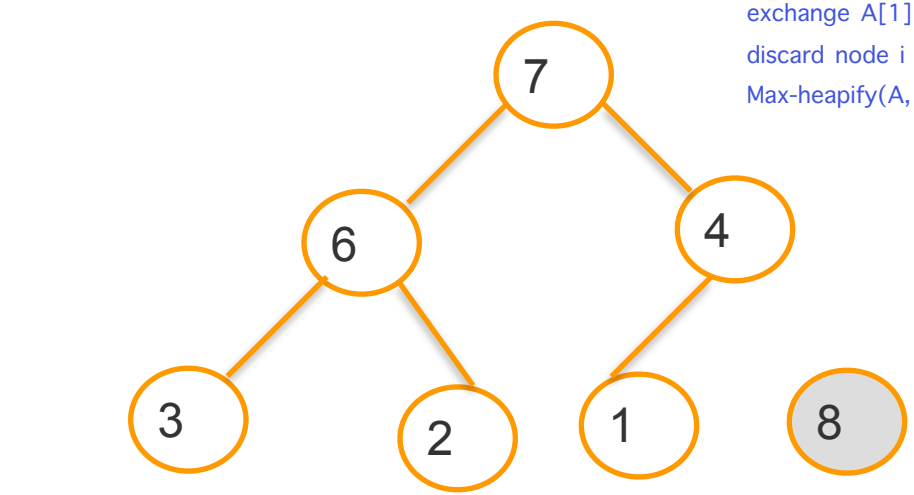


7	3	4	6	2	1	8	9	10	11

Exchange 1 and 7

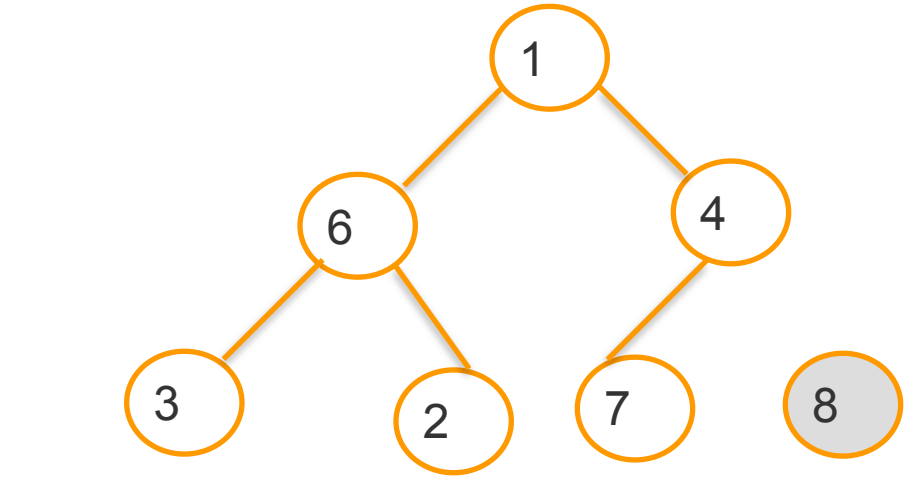
Heapsort(A):

#Create max heap
Build_Max_Heap from unordered array A
Finish sorting
iterate i from A.length downto 2
    exchange A[1] with A[i]
    discard node i from heap (decrement heap size)
    Max-heapify(A, 1) because new root may violate max heap property



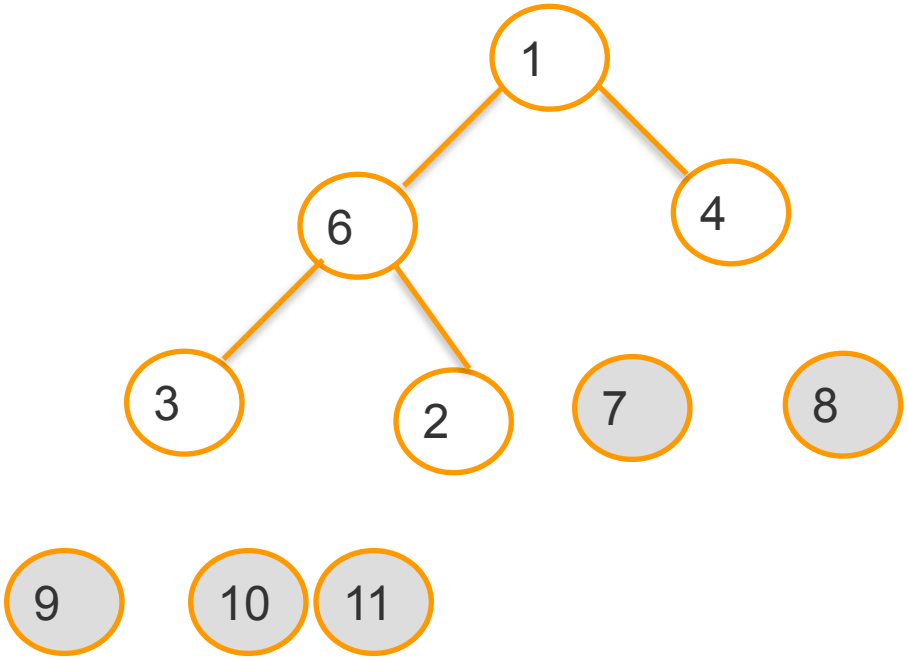
7	6	4	3	2	1	8	9	10	11

Remove 7 from the heap



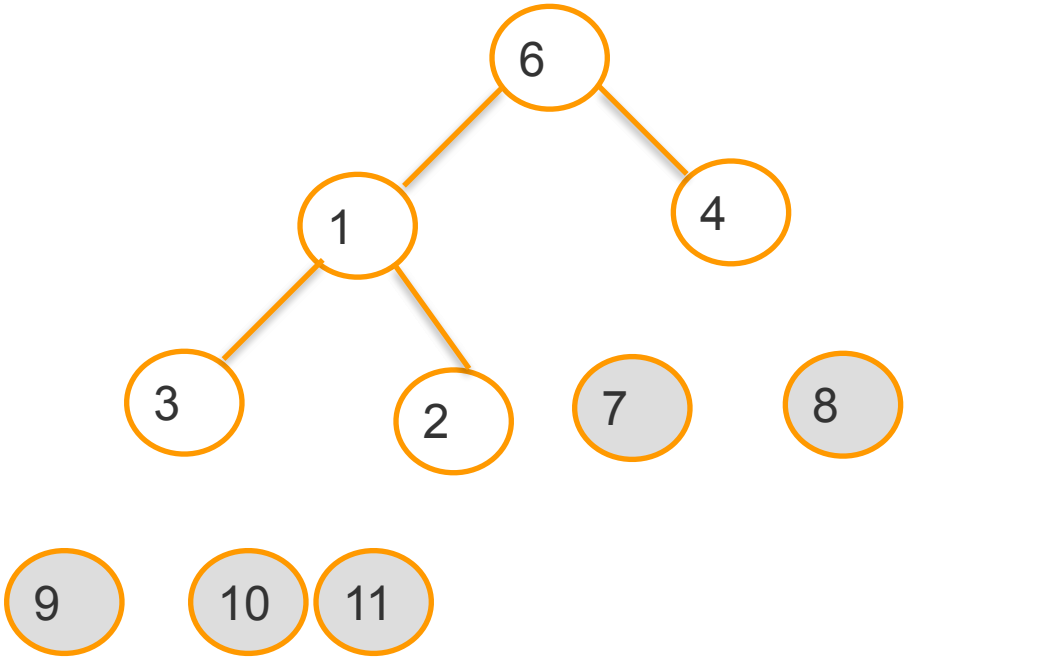
1	6	4	3	2	7	8	9	10	11

Exchange 1 and 6



1	6	4	3	2	7	8	9	10	11

Exchange 1 and 3

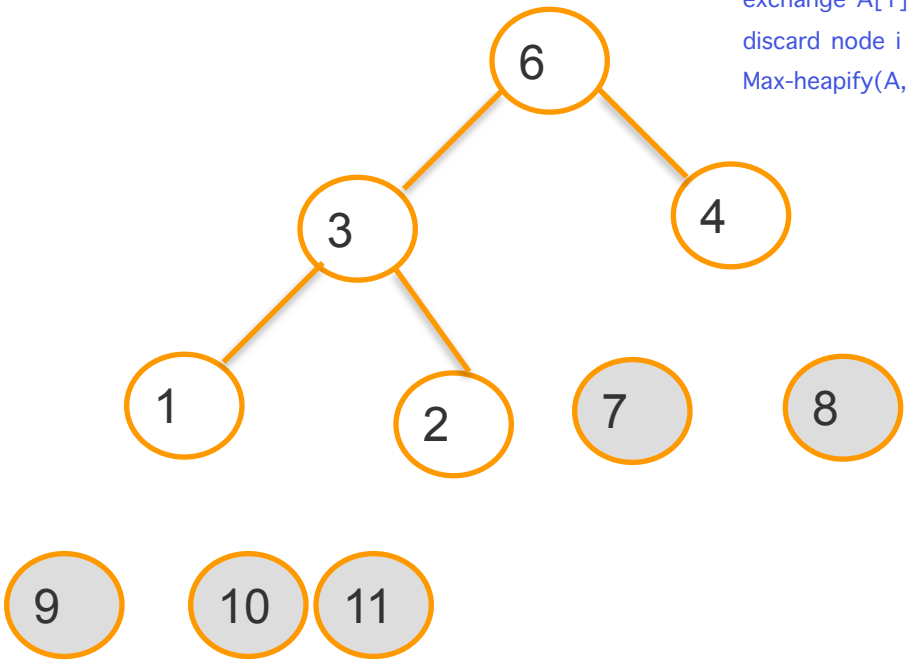


6	1	4	3	2	7	8	9	10	11

Exchange 6 and 2 and remove from the heap

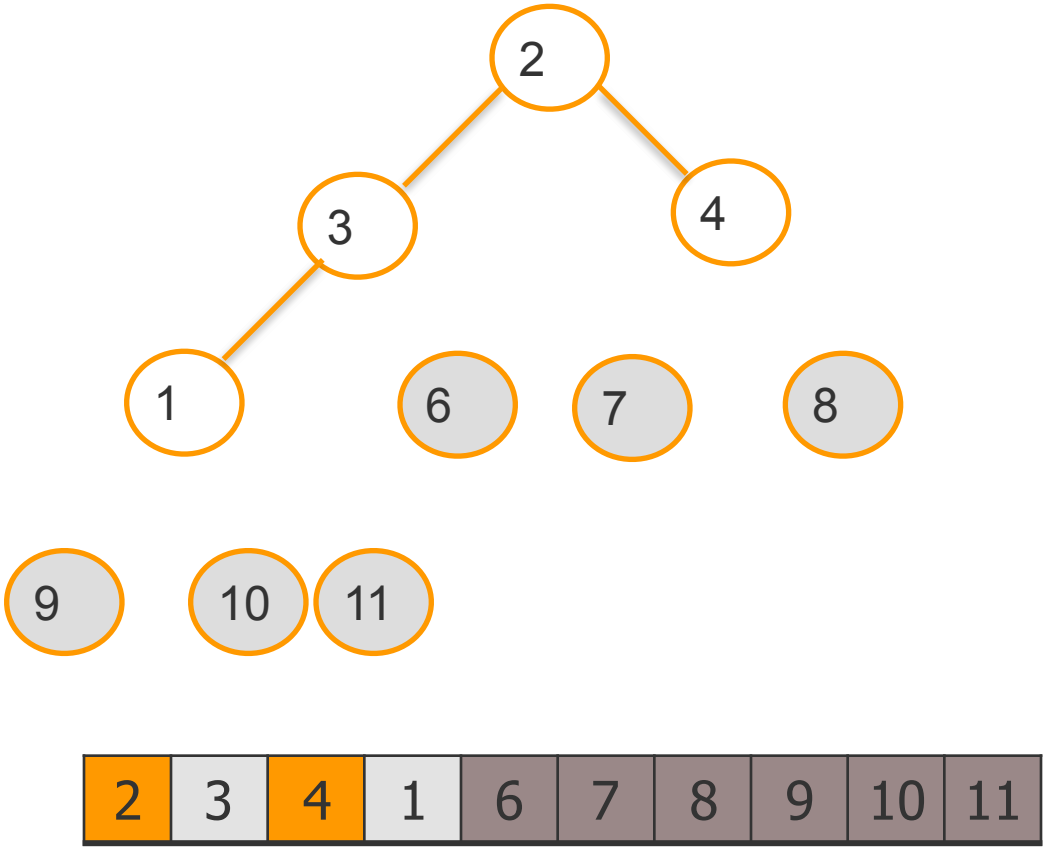
Heapsort(A):

#Create max heap
Build_Max_Heap from unordered array A
Finish sorting
iterate i from A.length downto 2
    exchange A[1] with A[i]
    discard node i from heap (decrement heap size)
    Max-heapify(A, 1) because new root may violate max heap property



6	3	4	1	2	7	8	9	10	11

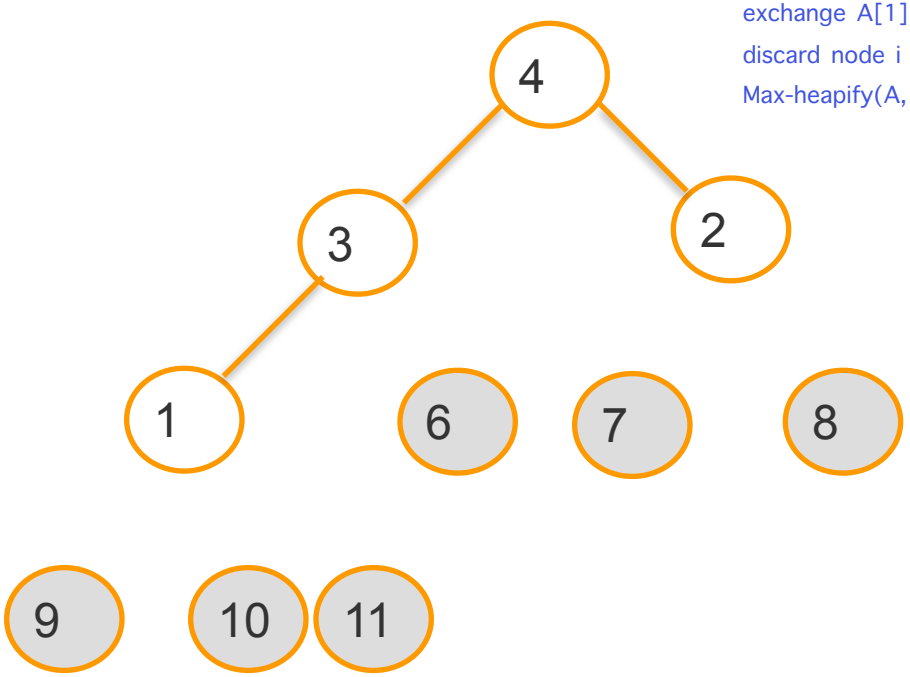
Exchange 4 and 2



Exchange 4 and 1

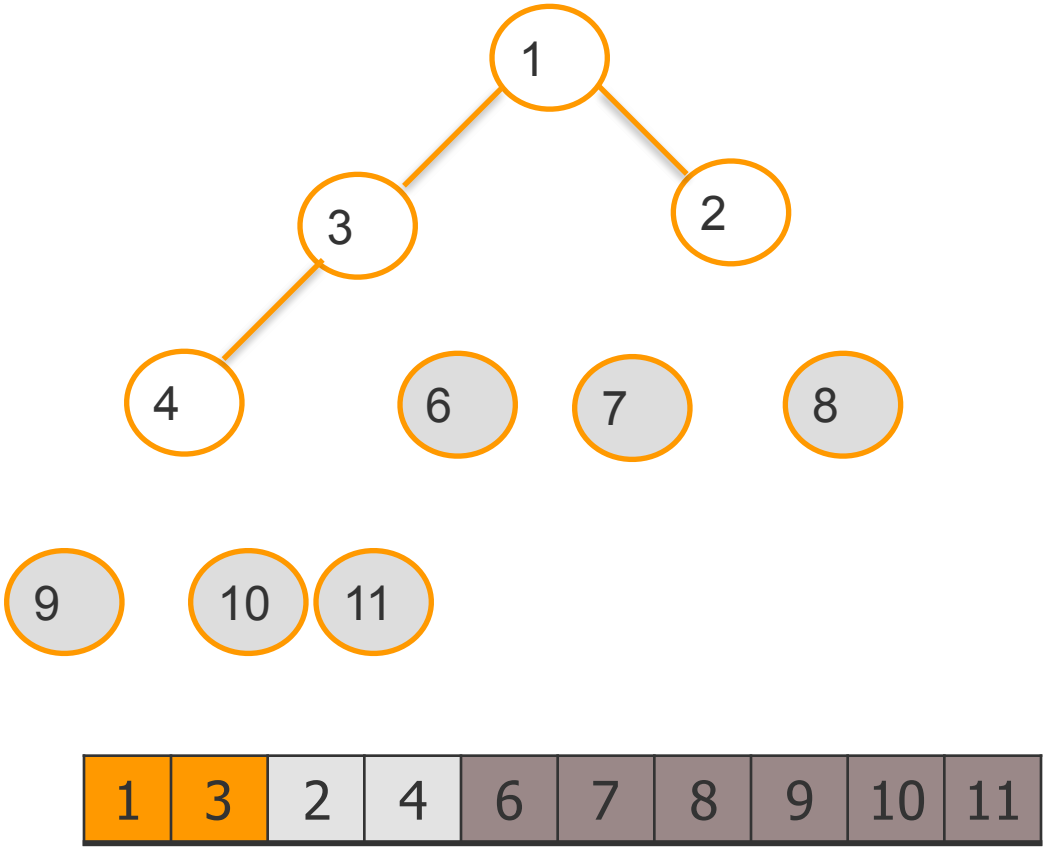
Heapsort(A):

#Create max heap
Build_Max_Heap from unordered array A
Finish sorting
iterate i from A.length downto 2
    exchange A[1] with A[i]
    discard node i from heap (decrement heap size)
    Max-heapify(A, 1) because new root may violate max heap property



4	3	2	1	6	7	8	9	10	11

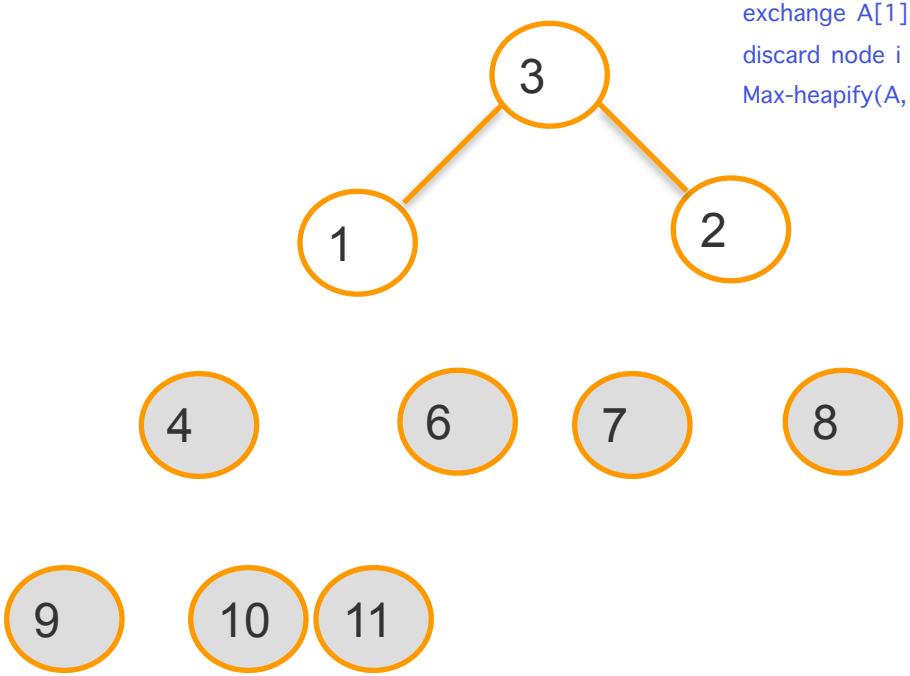
Remove 4, exchange 1 and 3



Exchange 2 and 3, and remove 3 from heap

Heapsort(A):

#Create max heap
Build_Max_Heap from unordered array A
Finish sorting
iterate i from A.length downto 2
    exchange A[1] with A[i]
    discard node i from heap (decrement heap size)
    Max-heapify(A, 1) because new root may violate max heap property

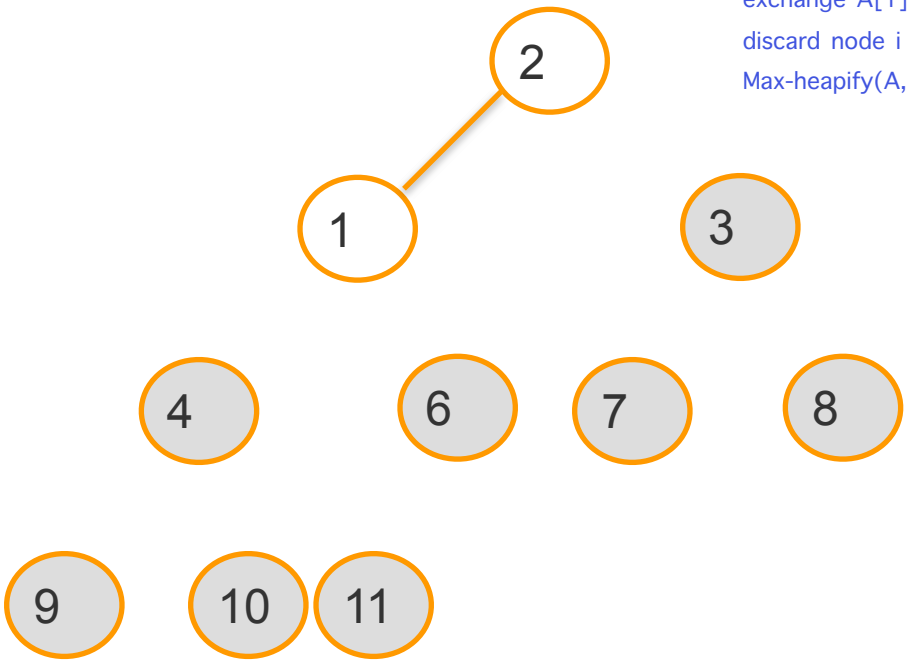


3	1	2	4	6	7	8	9	10	11

Exchange 1 and 2 and remove from heap

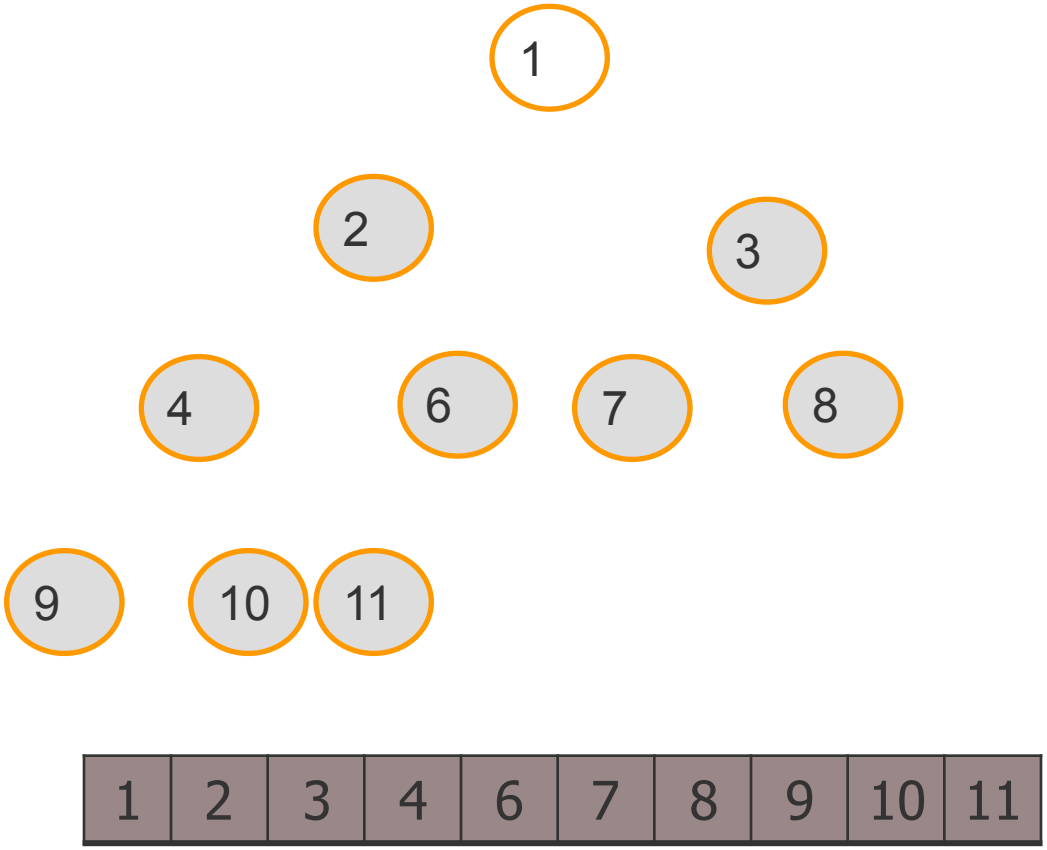
Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```

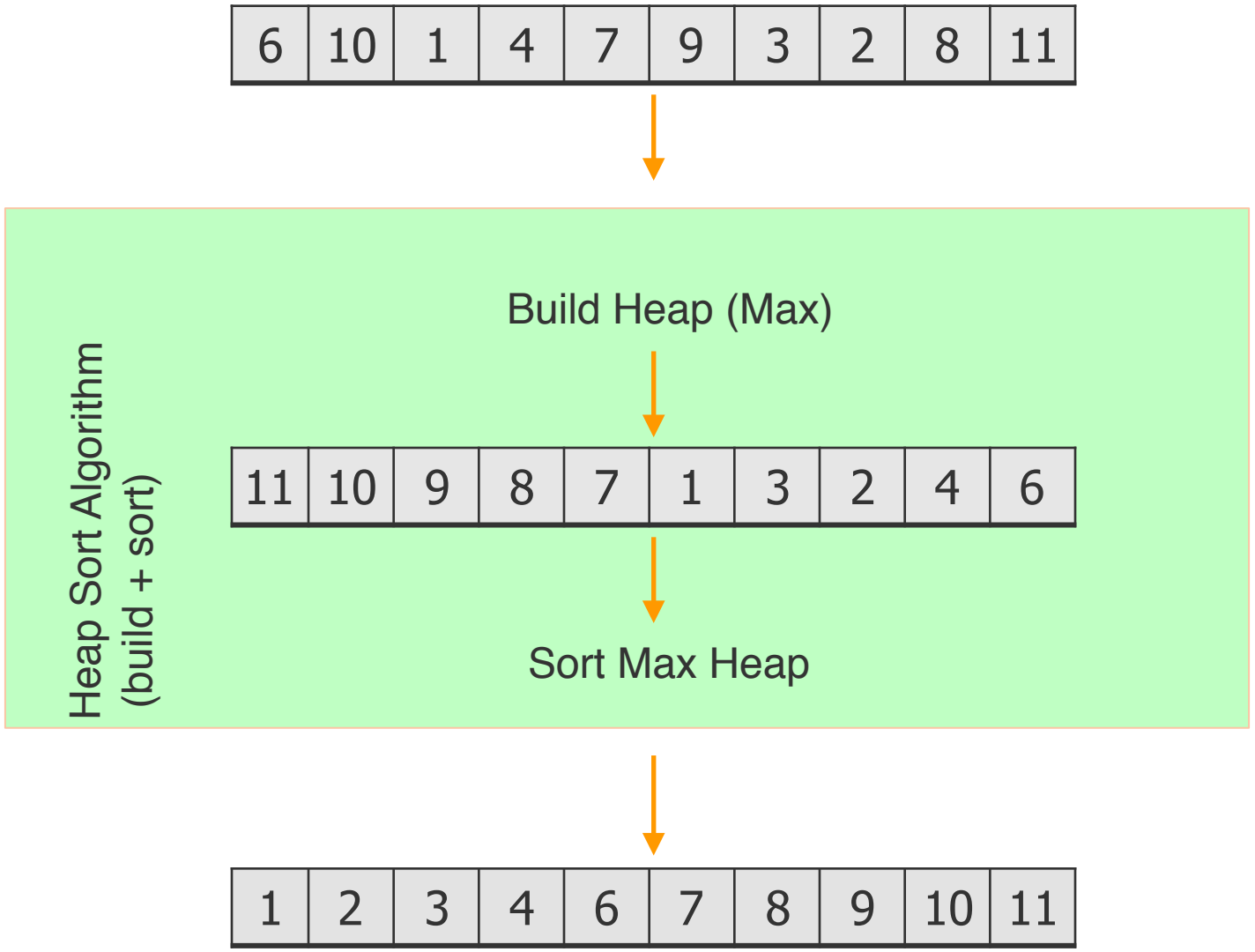


2	1	3	4	6	7	8	9	10	11

The array is sorted



Sorted Output



Heapsort Algorithm

Heapsort(A):

```
#Create max heap
Build_Max_Heap from unordered array A
# Finish sorting
iterate i from A.length downto 2
        exchange A[1] with A[i]
        discard node i from heap (decrement heap size)
        Max-heapify(A, 1) because new root may violate max heap property
```

Build Max Heap

```
Build_Max_Heap(A):
    set heap size to the length of the array
    iterate j from [A.length/2] down to 1:
        Max-heapify(A, j)
```

Heap

- The root of the tree is A[1], and given the index i of a node, we can easily compute the indices of its parent, left child, and right child:

```
def parent(i):
    return i/2

def left(i):
    try:
        return 2*i
    except:
        pass

def right(i):
    try:
        return 2 *i + 1
    except:
        pass
```

Max-Heapify

```
def max_heapify(arr,i):
        n = len(arr)
        l = left(i)
        r = right(i)

        if l <= n and arr[l] > arr[i]:
            largest = l
        else:
            largest = i

        if r <= n and arr[r] > arr[largest]:
            largest = r

        if largest != i:
            temp = arr[i]
            arr[i] = arr[largest]
            arr[largest] = temp
            max_heapify(arr,largest)
        return arr
```