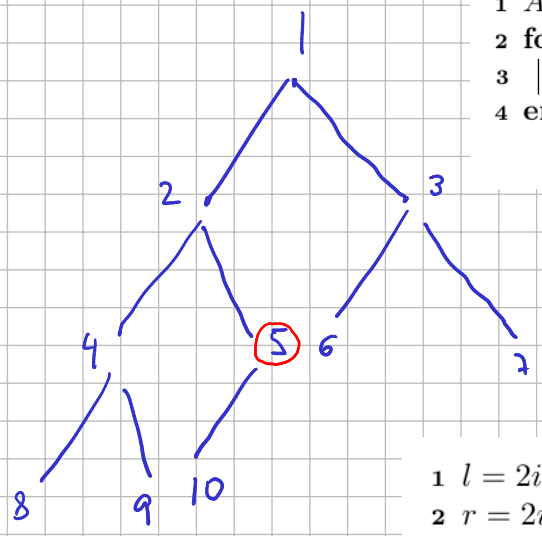



```
1  A.heap-size ← A.length;
2  for i = ⌊A.length/2⌋ downto 1 do
3  |  Max-Heapify(A,i);
4  end
```

Algorithm 10: Build-Max-Heap(A)

$$T_w(n) = \sum_{i=1}^{n/2} O(\lg n) = O\left(\lg n \cdot \sum_{i=1}^{n/2} 1\right)$$

$$= O(n \cdot \lg n).$$

$T(n) = T\left(\frac{2n}{3}\right) + \theta(n^0)$

$a = 1,\ b = 3/2,\ d = 0$

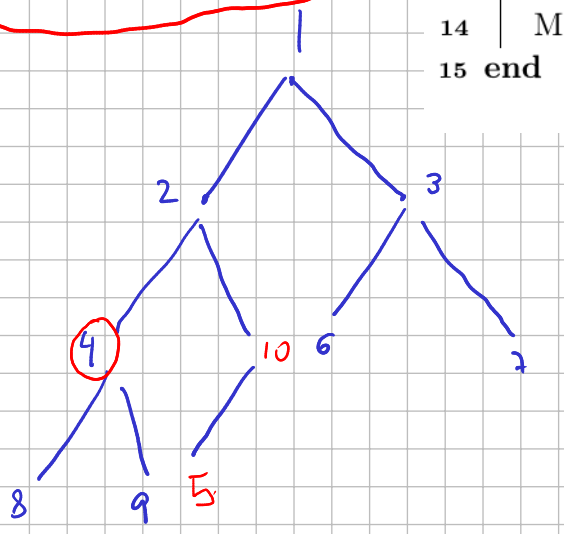
$a = 1 = b^d$

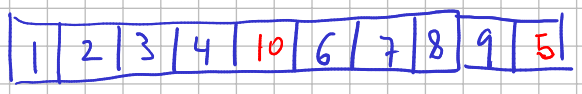
$T(n) = \theta(n^0 \cdot \lg n)$

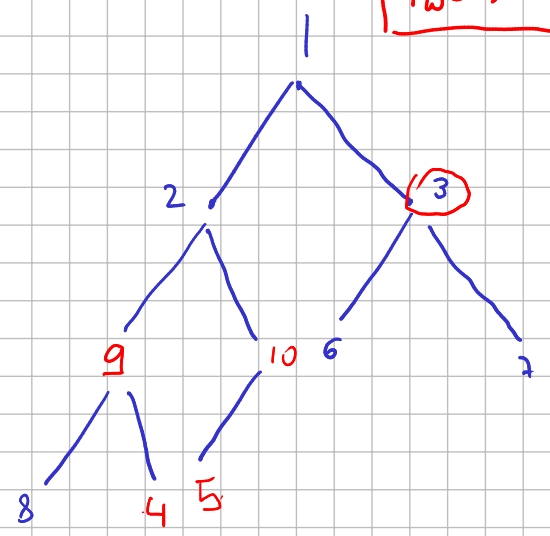
```
1  l = 2i;                                   } θ(1)
2  r = 2i + 1;                               }
3  if l ≤ A.heap-size and A[l] > A[i] then   }
4  |  largest = l;                           } θ(1)
5  end                                       }
6  else                                      }
7  |  largest = i;                           }
8  end                                       }
9  if r ≤ A.heap-size and A[r] > A[largest] then } θ(1)
10 |  largest = r;                           }
11 end
12 if largest ≠ i then                       } Pior caso:
13 |  exchange A[i] with A[largest];         } T_w(n) ≤ T_w(2n/3) + θ(1)
14 |  Max-Heapify(A,largest);
15 end
```

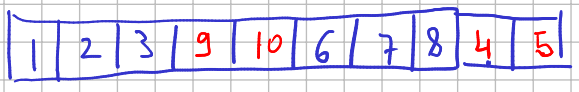
Algorithm 11: Max-Heapify(A,i)

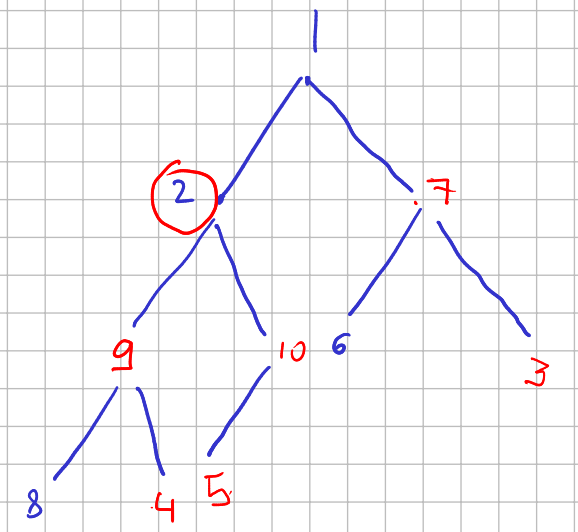
$T_w(n) = O(\lg n)$

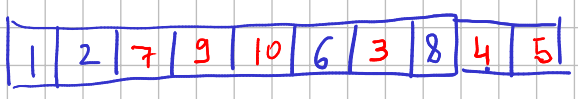


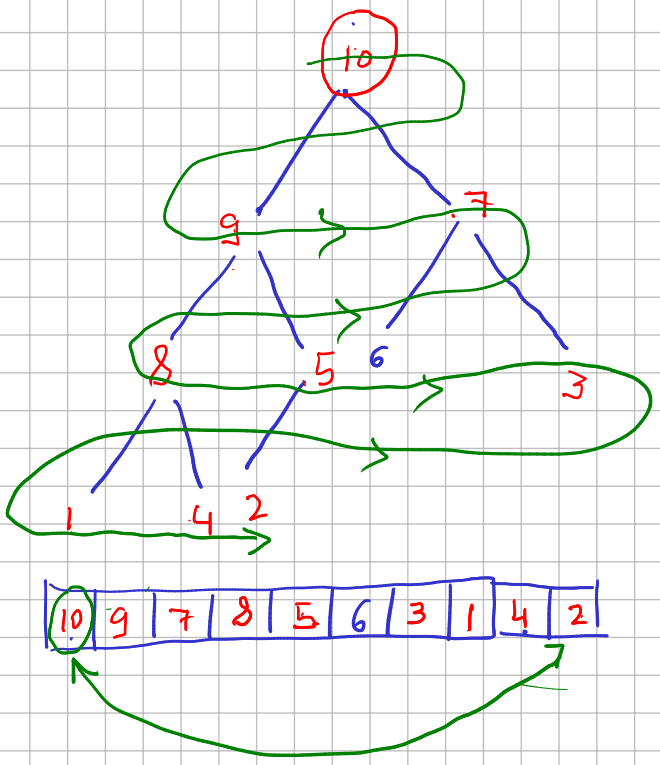












$|A| = n$.

```
1 Build-Max-Heap(A);                                  → O(n)
2 for i = A.length downto 2 do
3 |   exchange A[1] with A[i];
4 |   A.heap-size = A.heap-size - 1;
5 |   Max-Heapify(A,1);                               → O(lg n)
6 end
```

$$\sum_{i=2}^{n} O(\lg n) = O\left(\lg n \cdot \sum_{i=2}^{n} 1\right) = O(n \cdot \lg n)$$

Algorithm 12: Heapsort(A)

$$T_w(n) = O(n \cdot \lg n).$$