COMP250-Heap

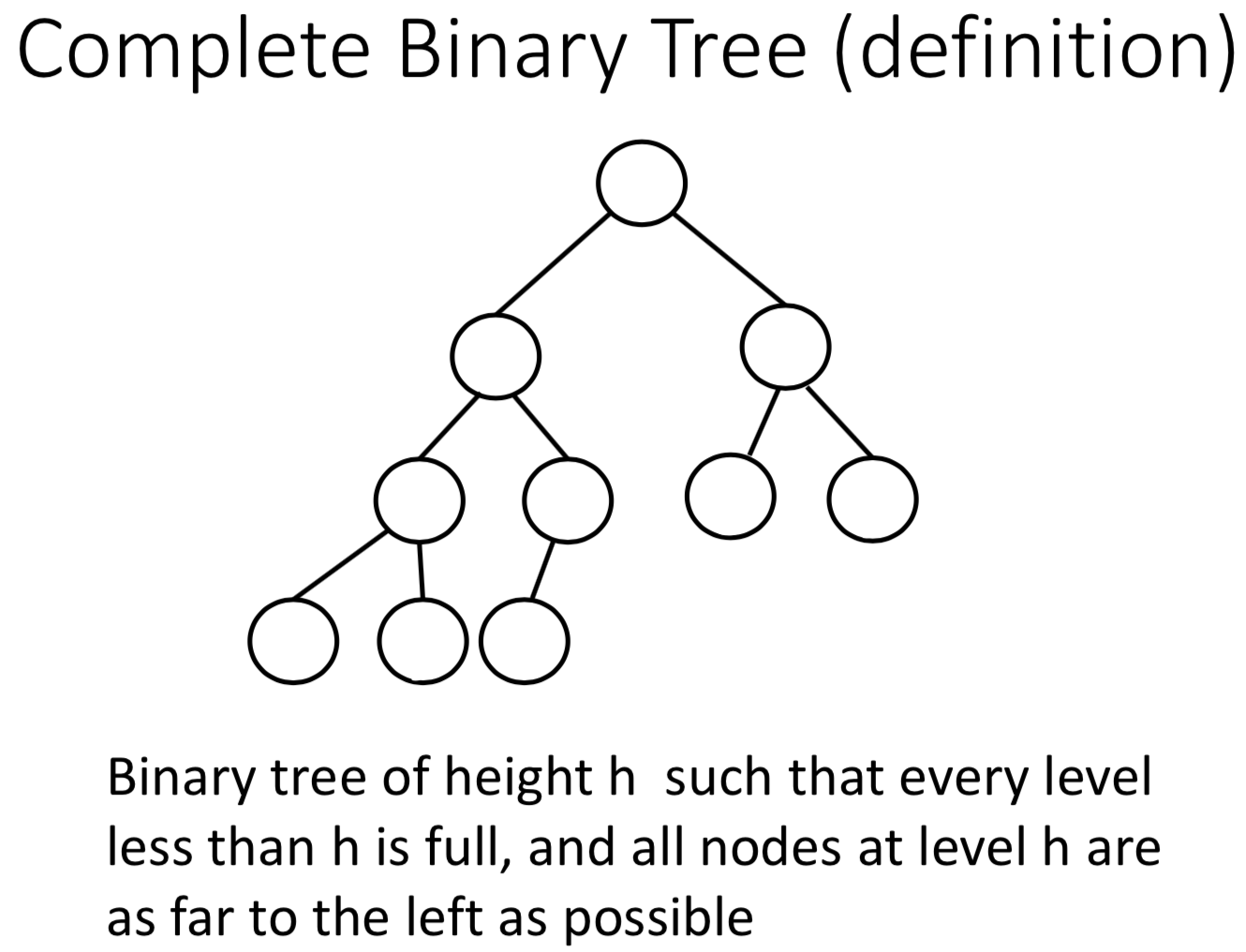
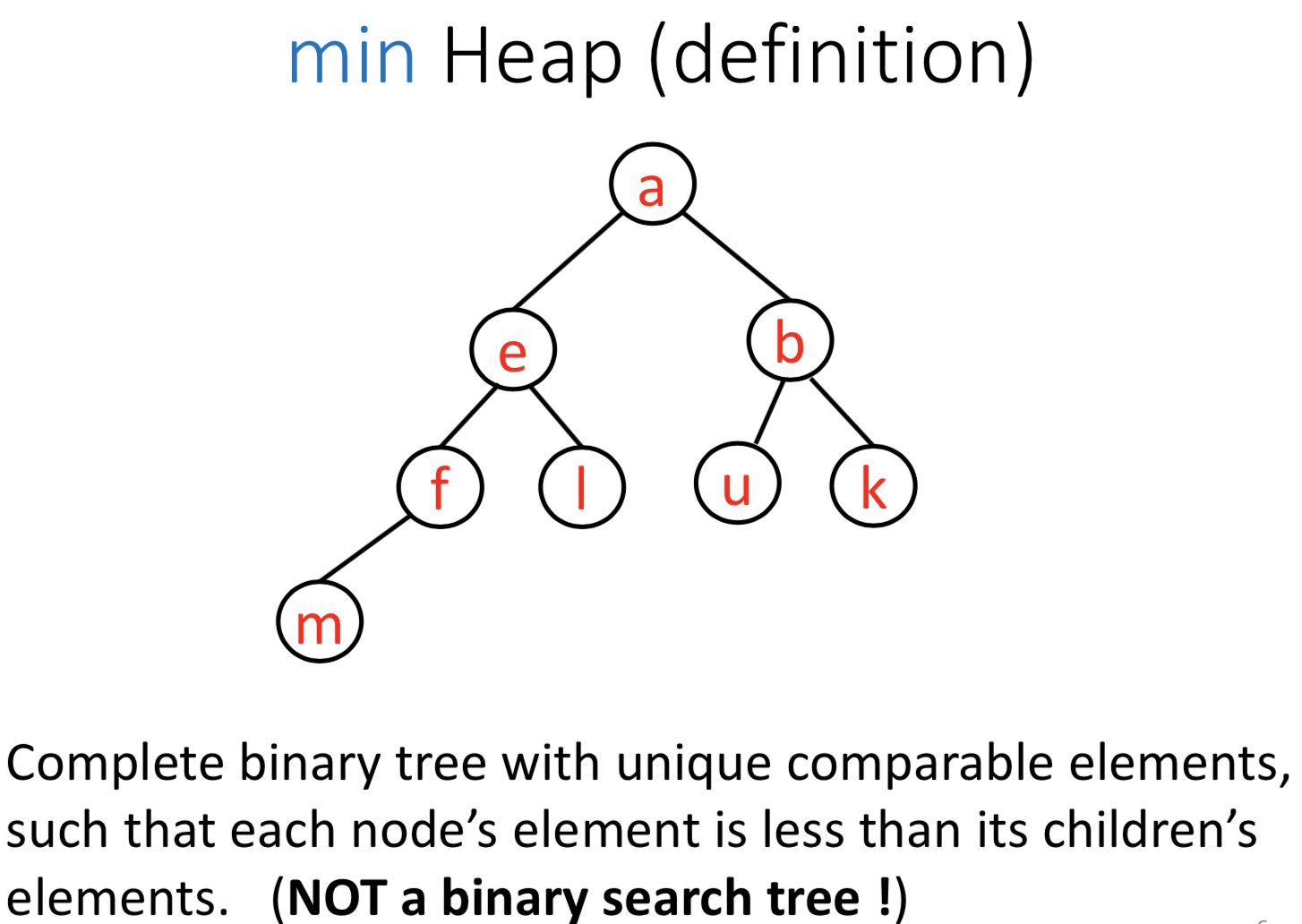
*Priority Queue (ADT)*

Like a queue, but now we have a more general definition of which element to remove next, namely the one with highest priority.

Assume a set of comparable elements or “keys”.

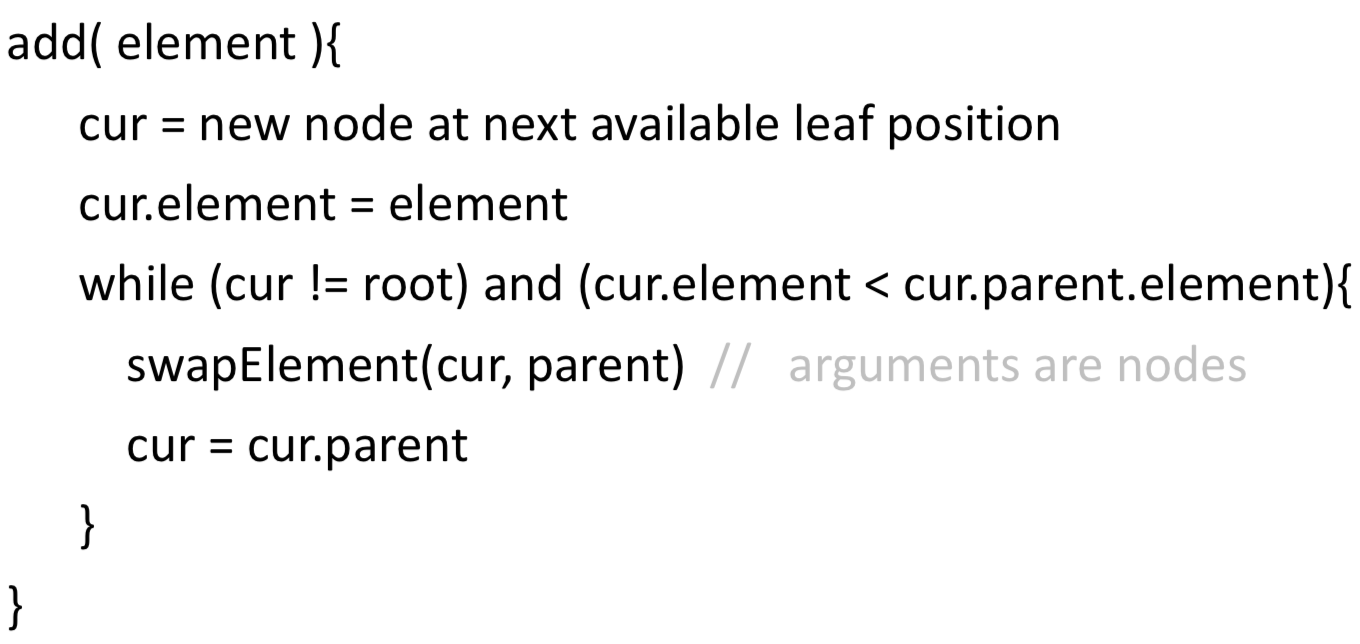
(Comparable means that there is an ordering, as in the Java Comparable interface.)

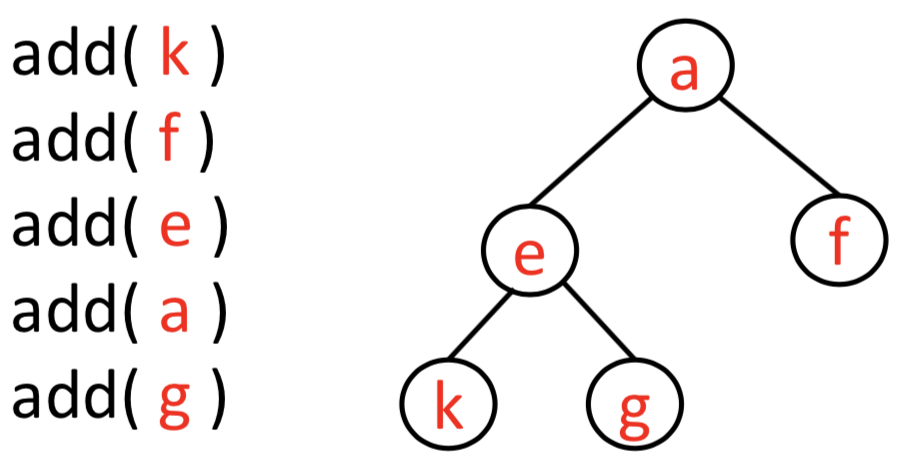
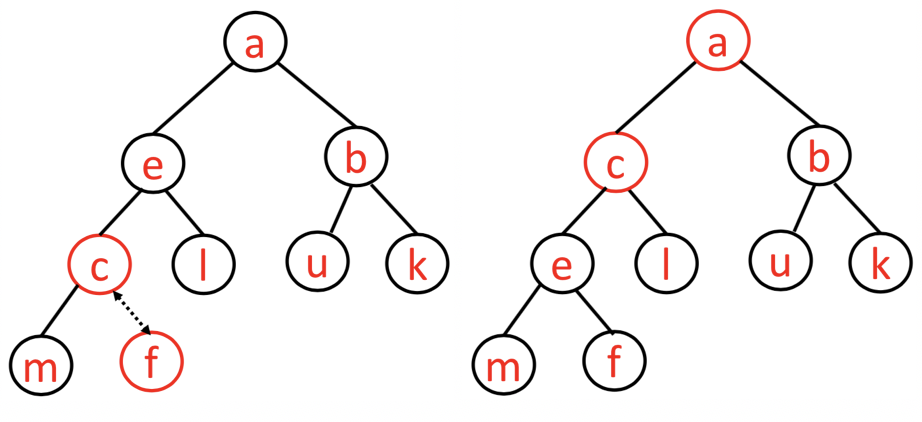
* add(element)
* removeMin() //“highest” priority = “number 1” priority

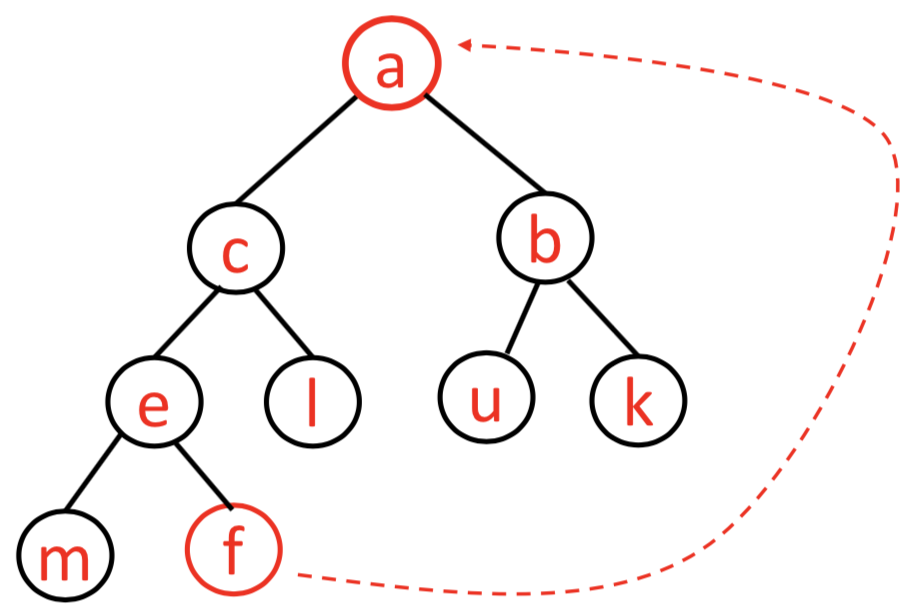
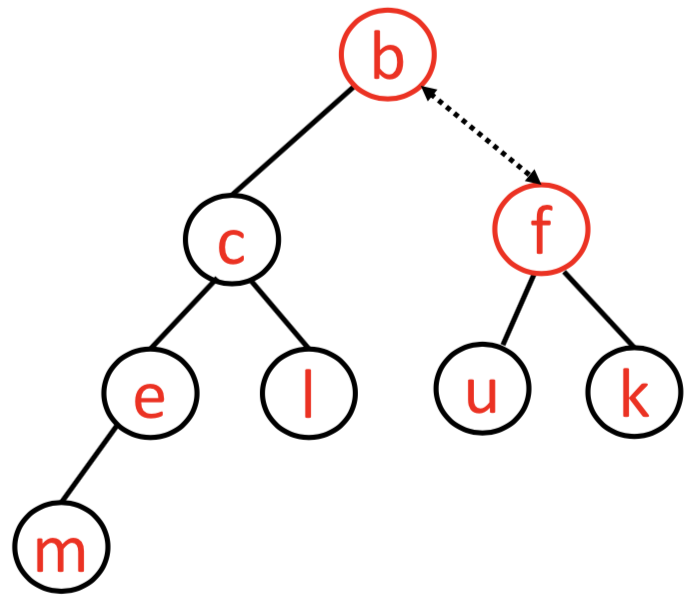
no sibling relationship

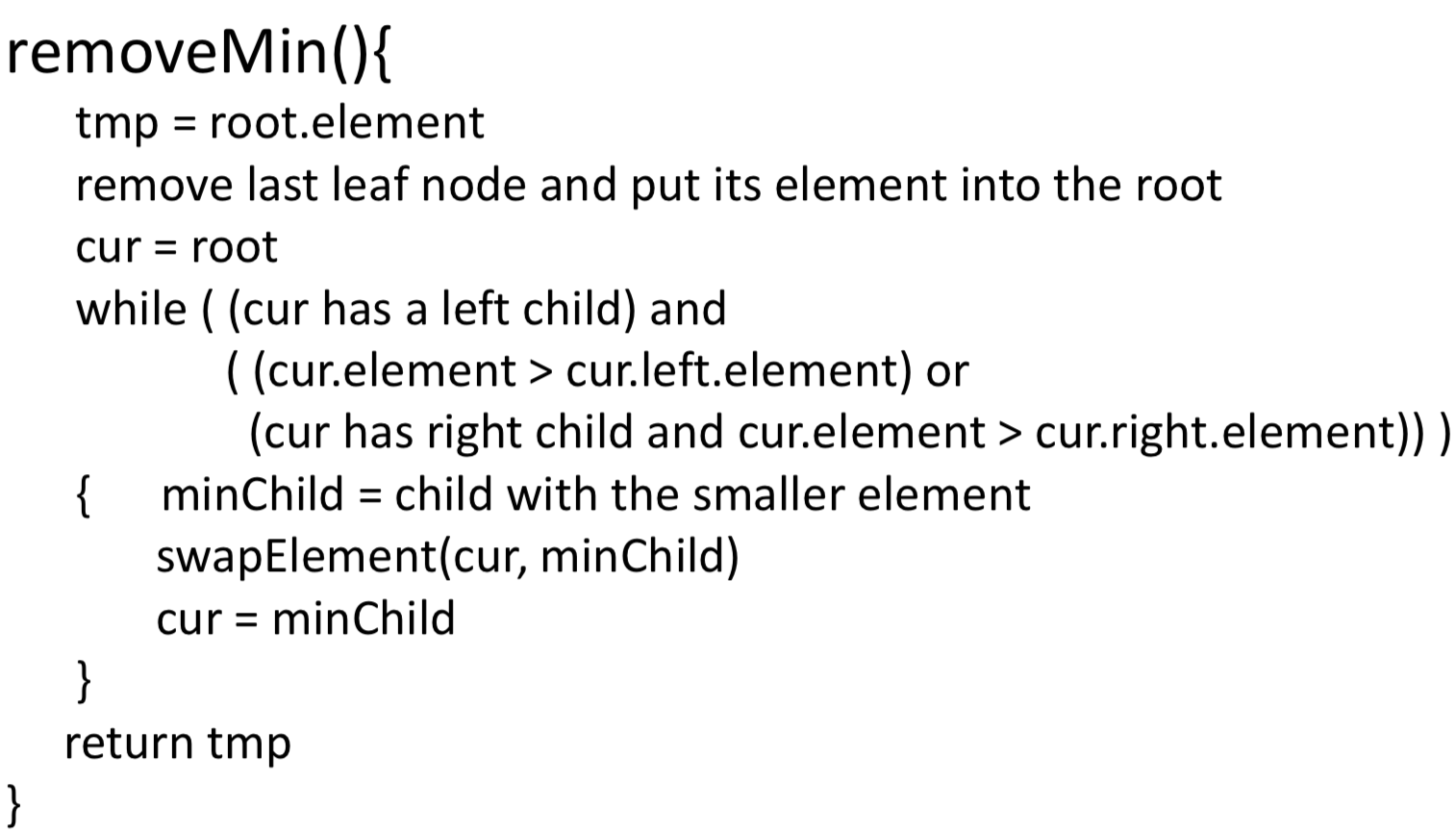
* *add(c)*

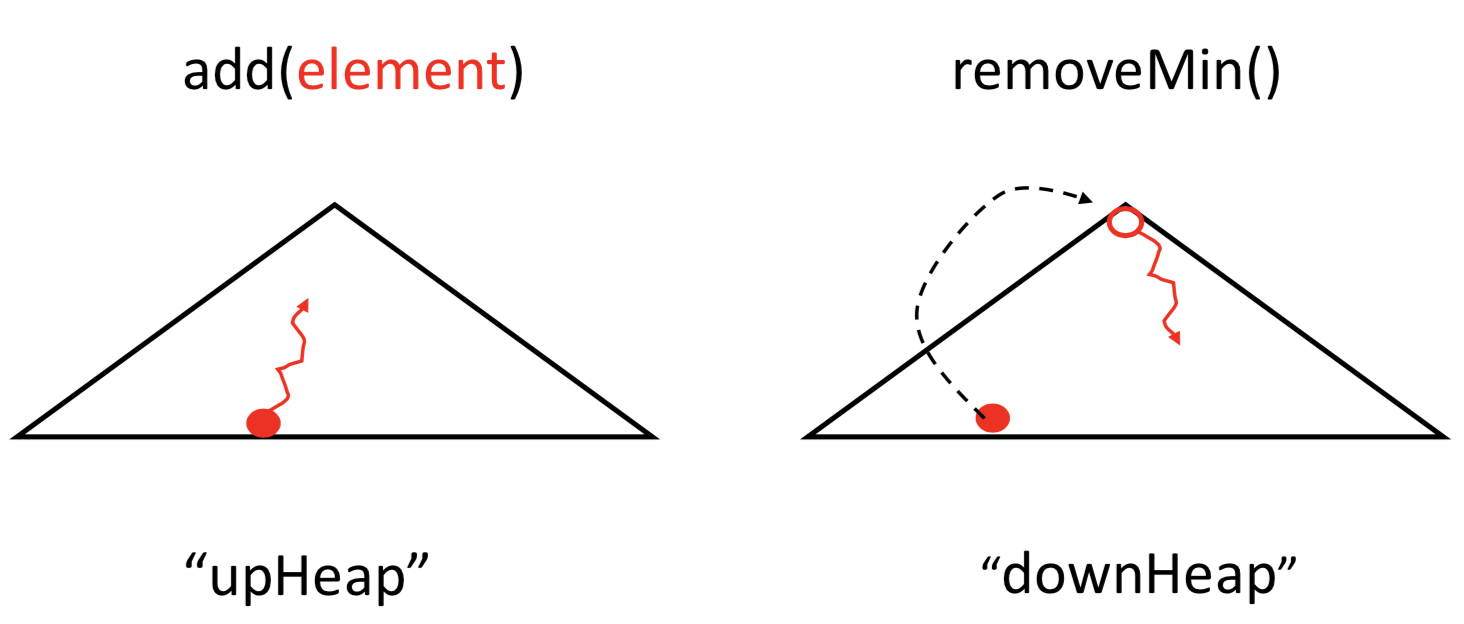




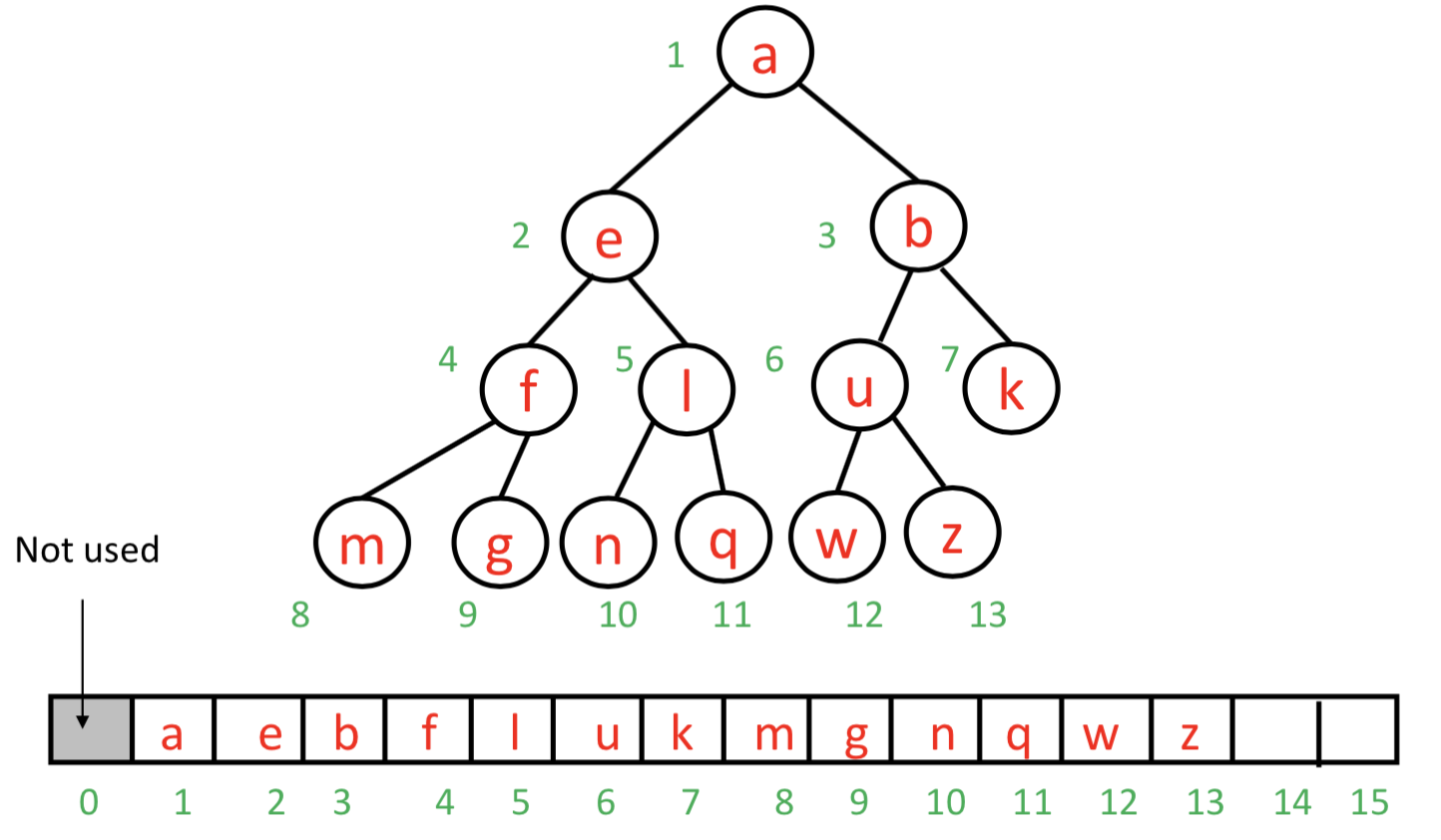
* *removeMin()*





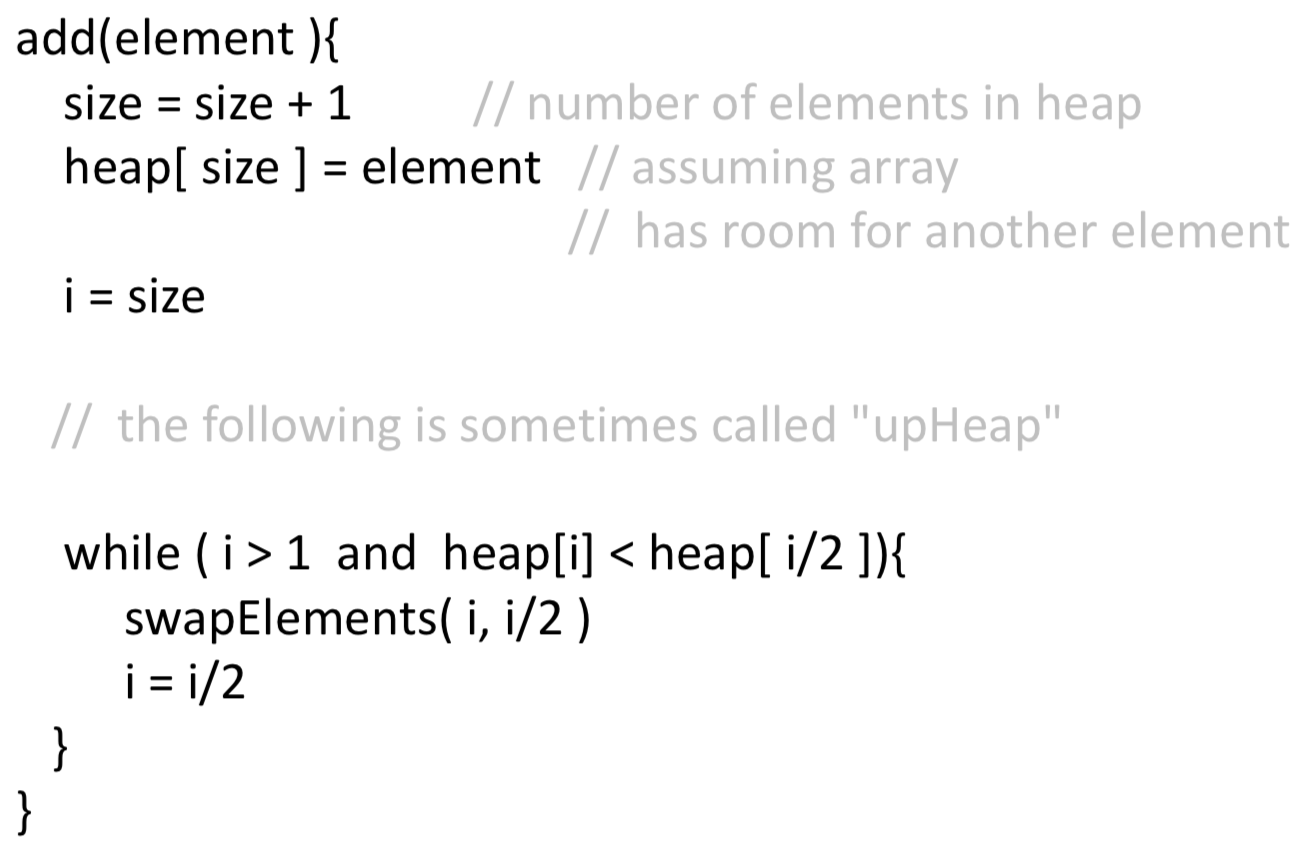
***Heap(array implementation)***

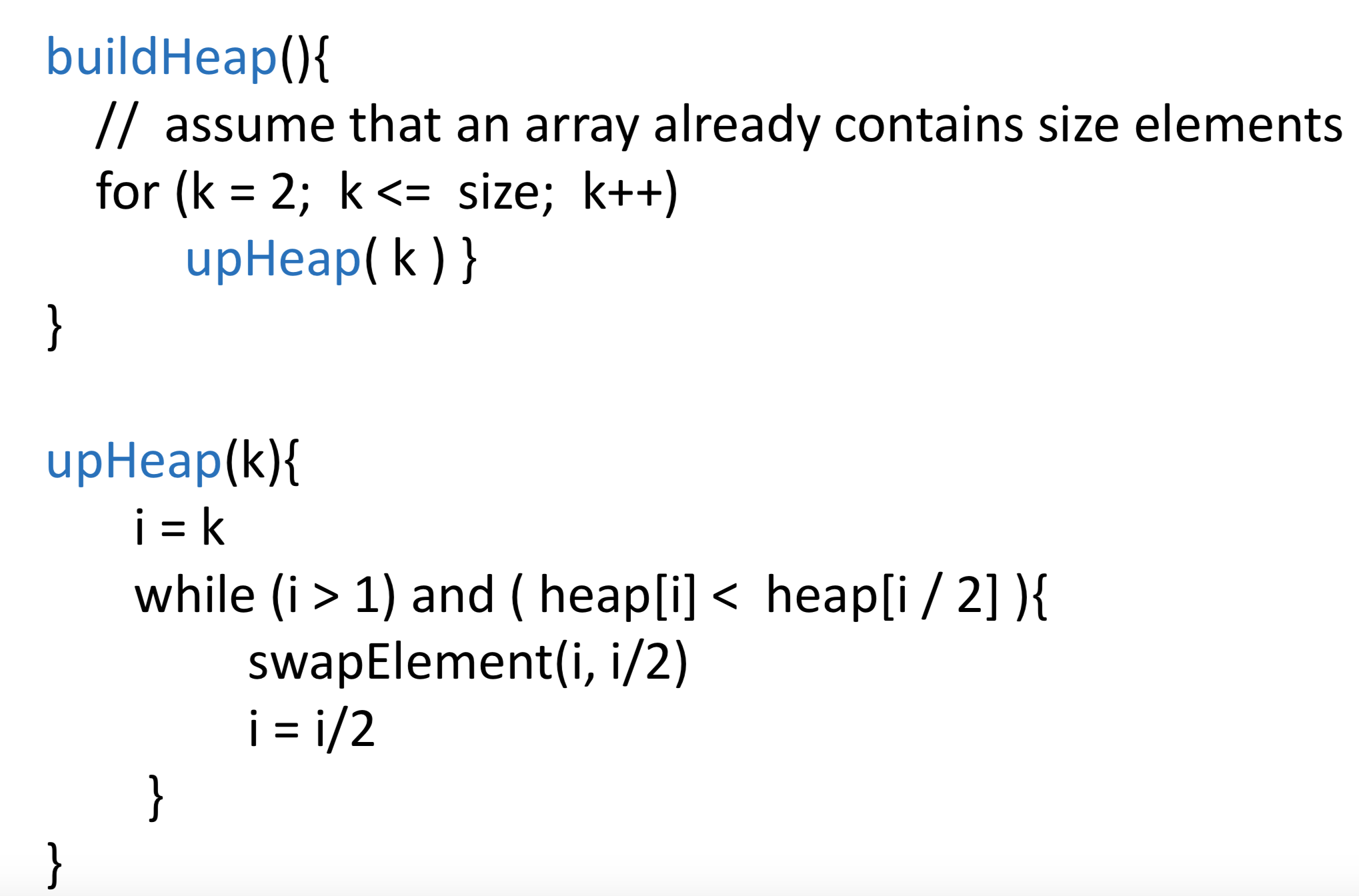


parent = child / 2

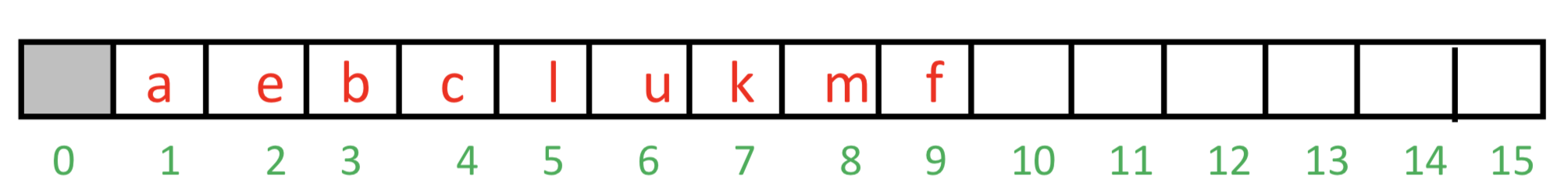
left = 2\*parent

right = 2\*parent + 1



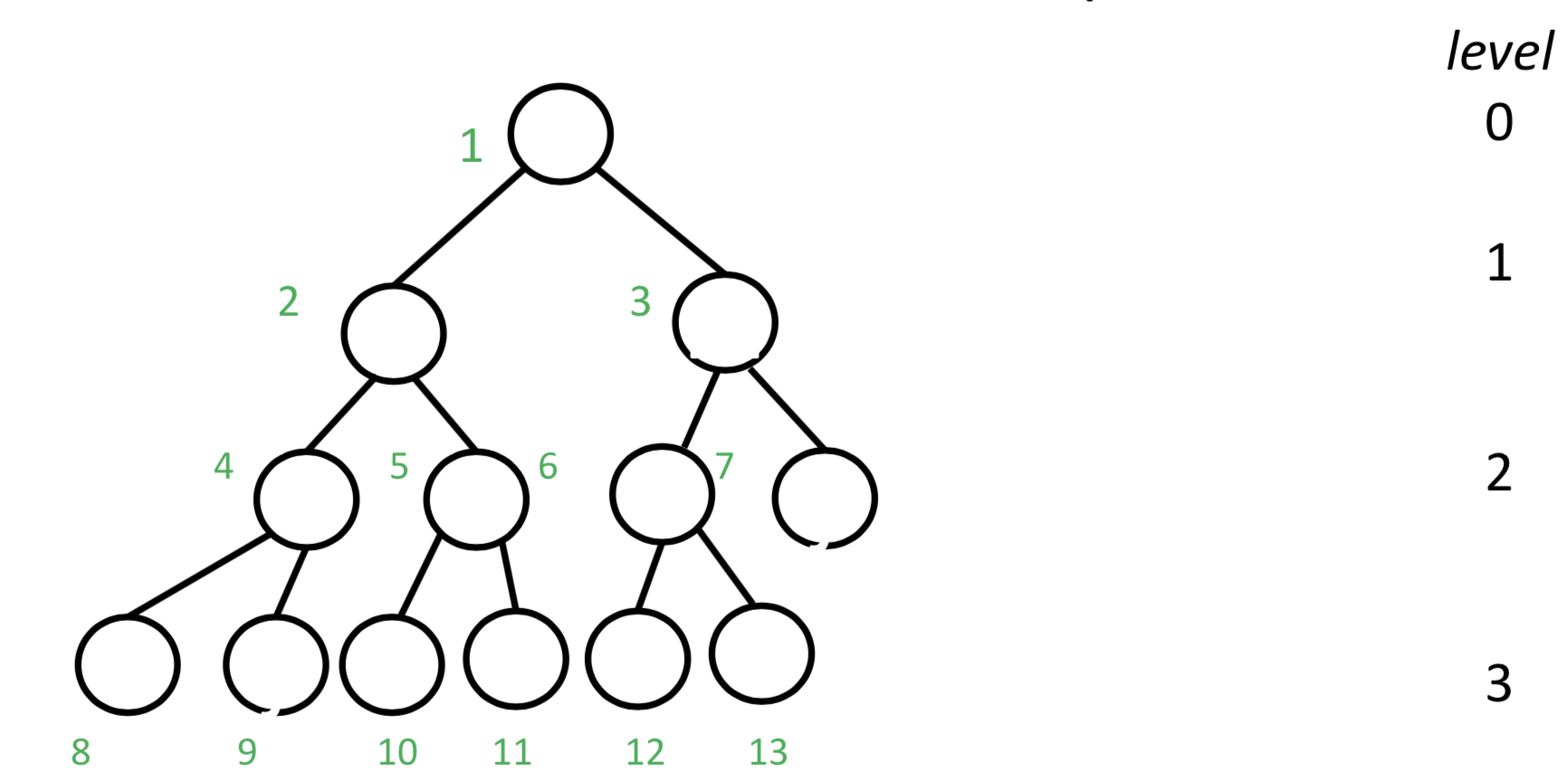


Best case of buildHeap is O(𝑛)



In the best case, the elements happen to already satisfy the heap parent-child ordering constraint, and no swaps are necessary.

Worse case of buildHeap is

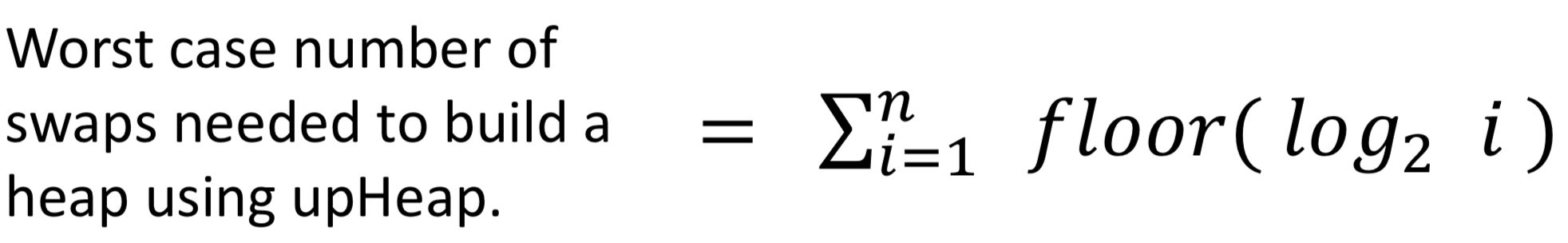


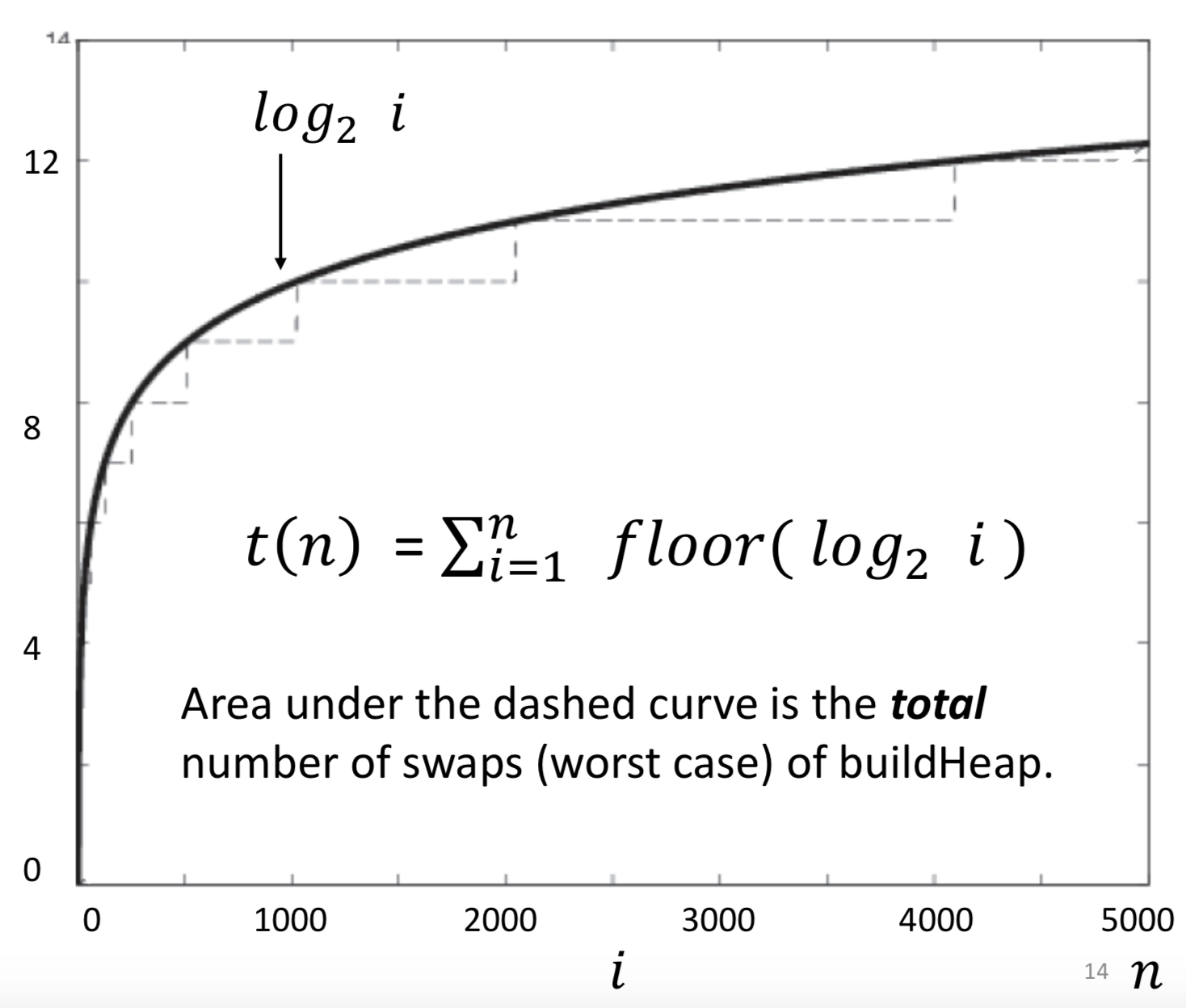
How many upHeap swaps do we need for element 𝑖 ? Element 𝑖 is at level, such that:

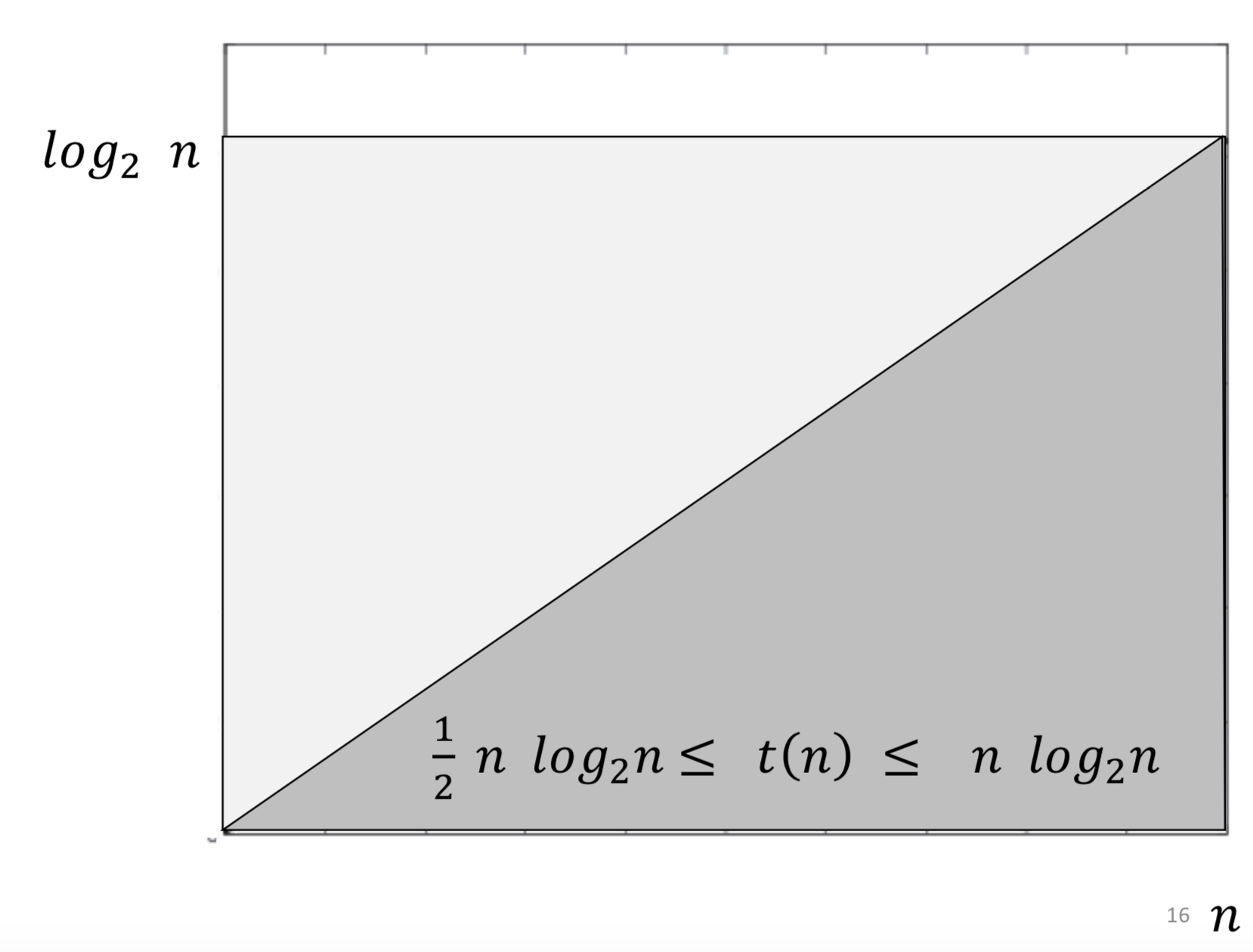
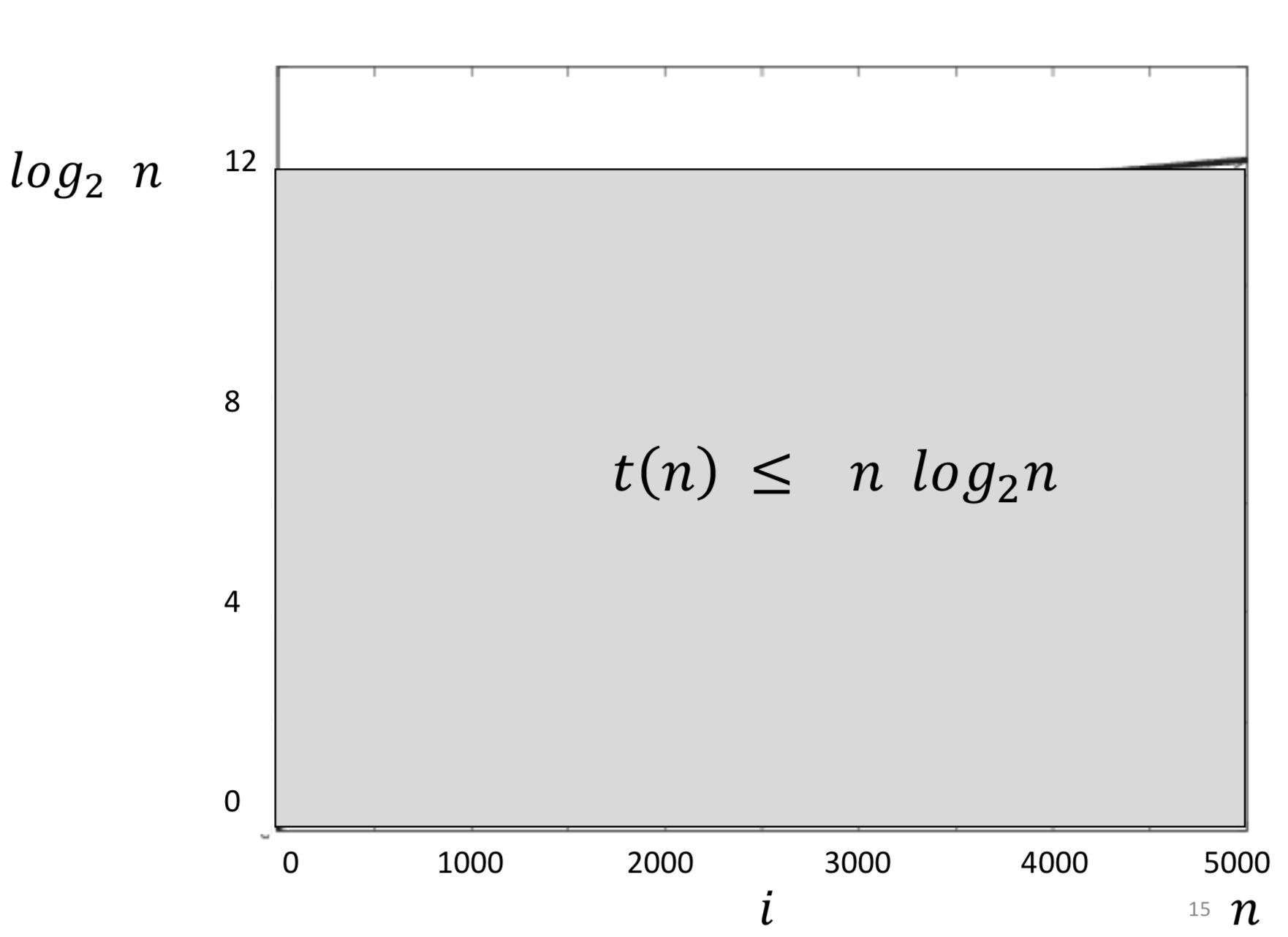
2𝑙𝑒𝑣𝑒𝑙 ≤ 𝑖 < 2𝑙𝑒𝑣𝑒𝑙 +1

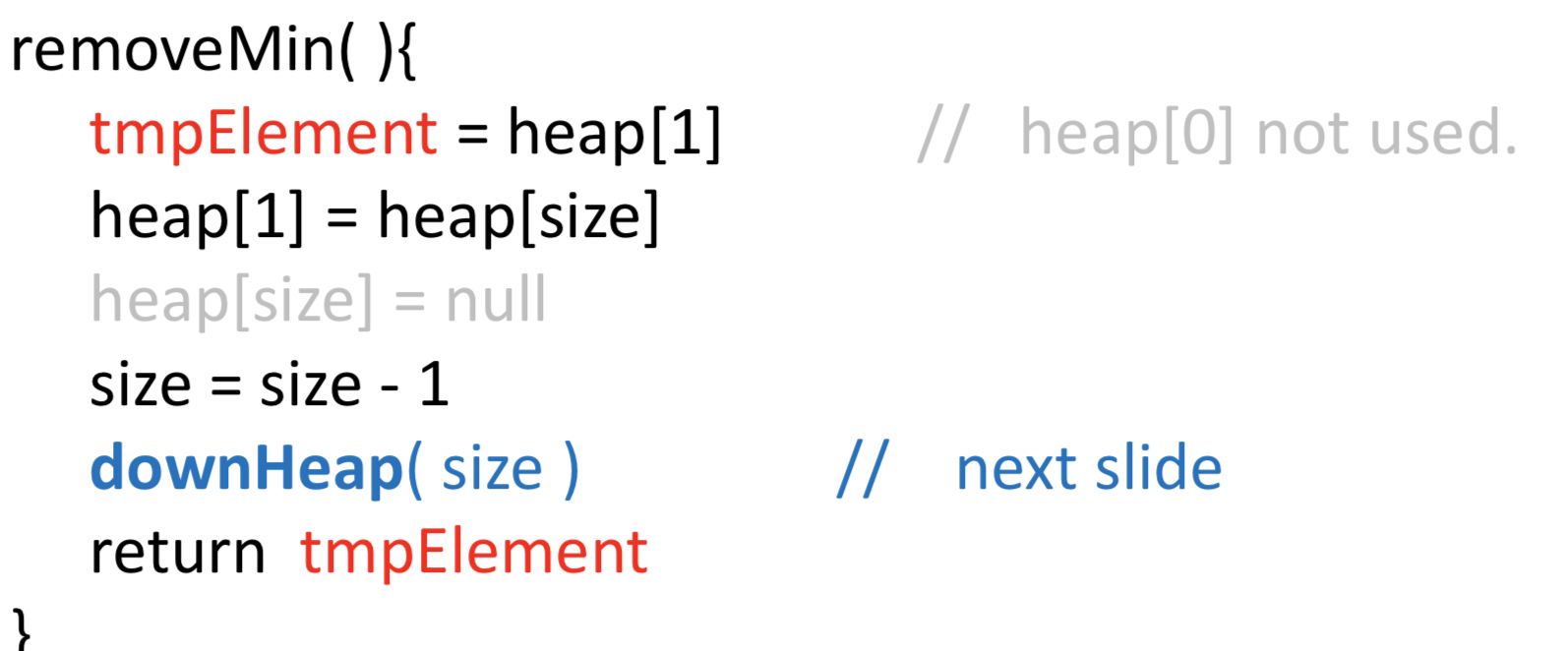
𝑙𝑒𝑣𝑒𝑙≤ 𝑙𝑜𝑔2𝑖 <𝑙𝑒𝑣𝑒𝑙+1

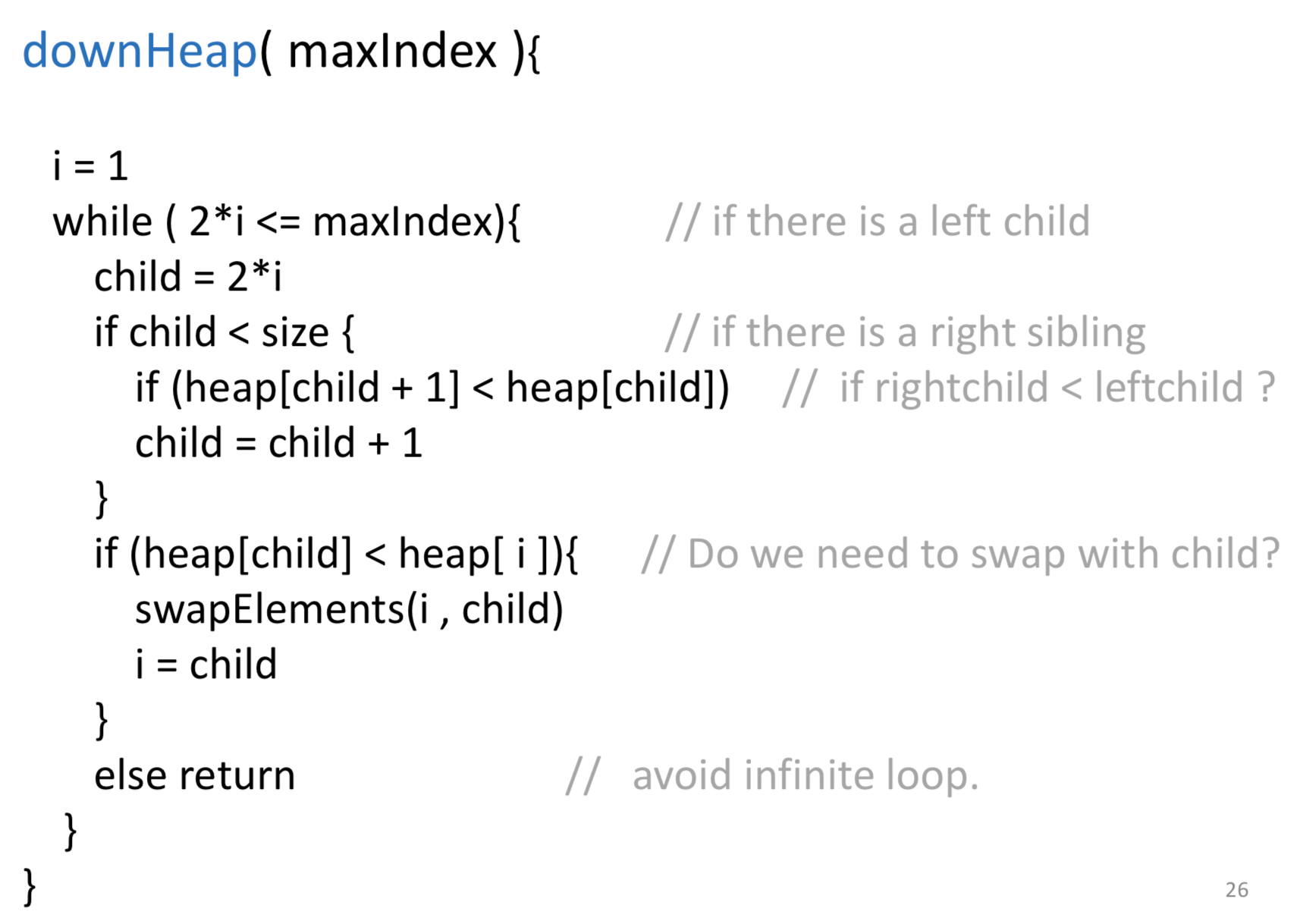
Thus, 𝑙𝑒𝑣𝑒𝑙 = 𝑓𝑙𝑜𝑜𝑟( 𝑙𝑜𝑔2 𝑖 )







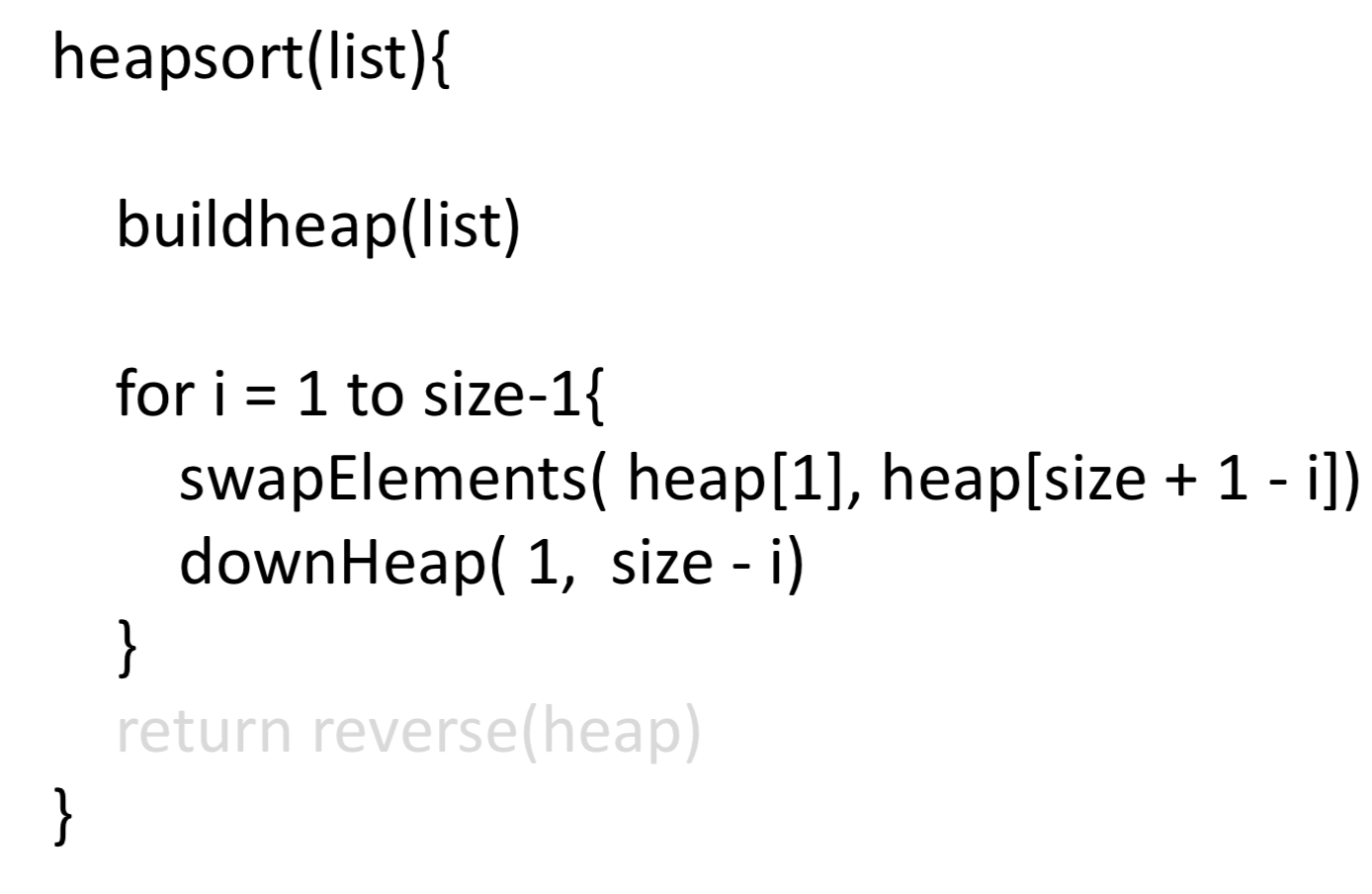


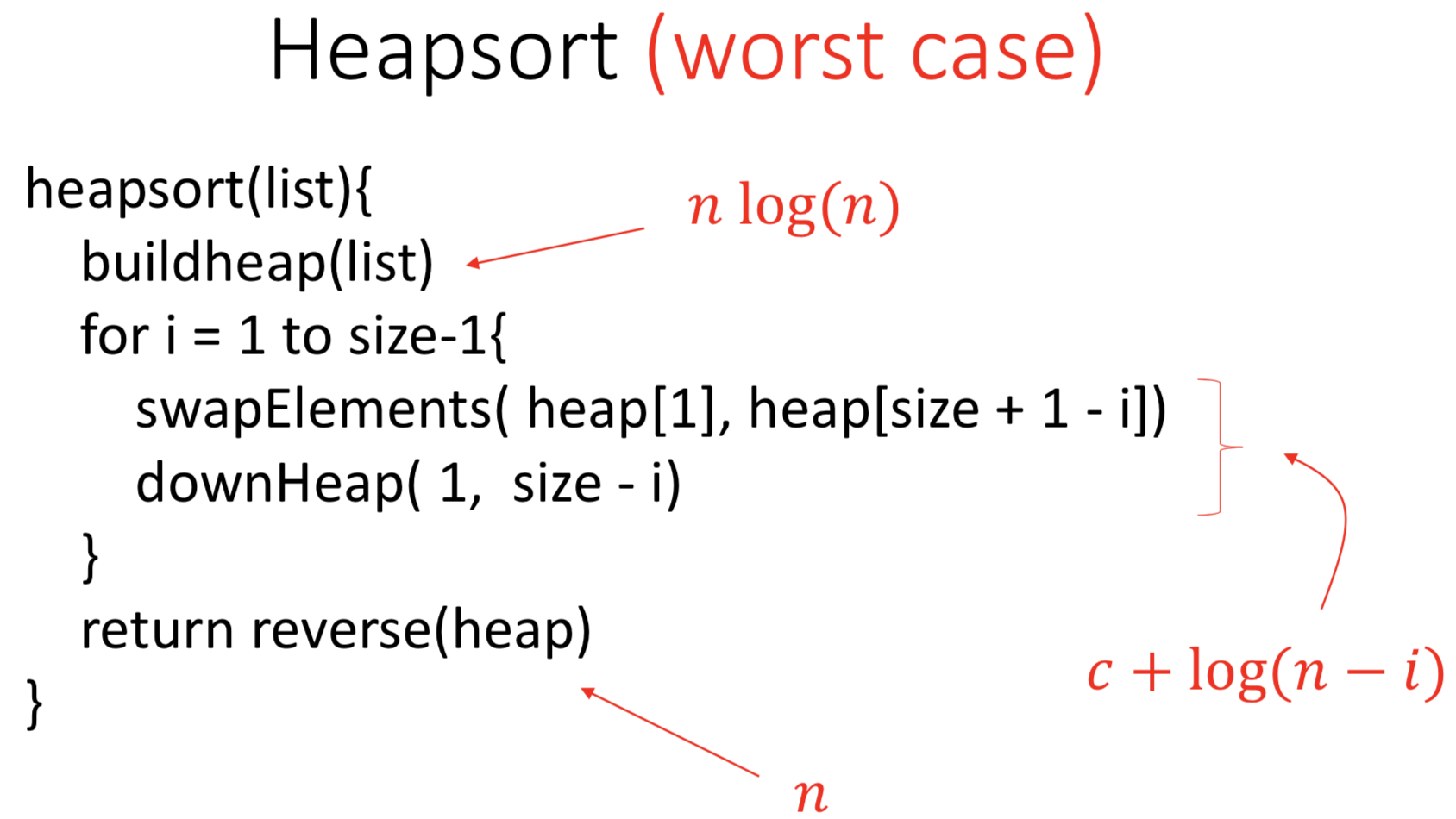


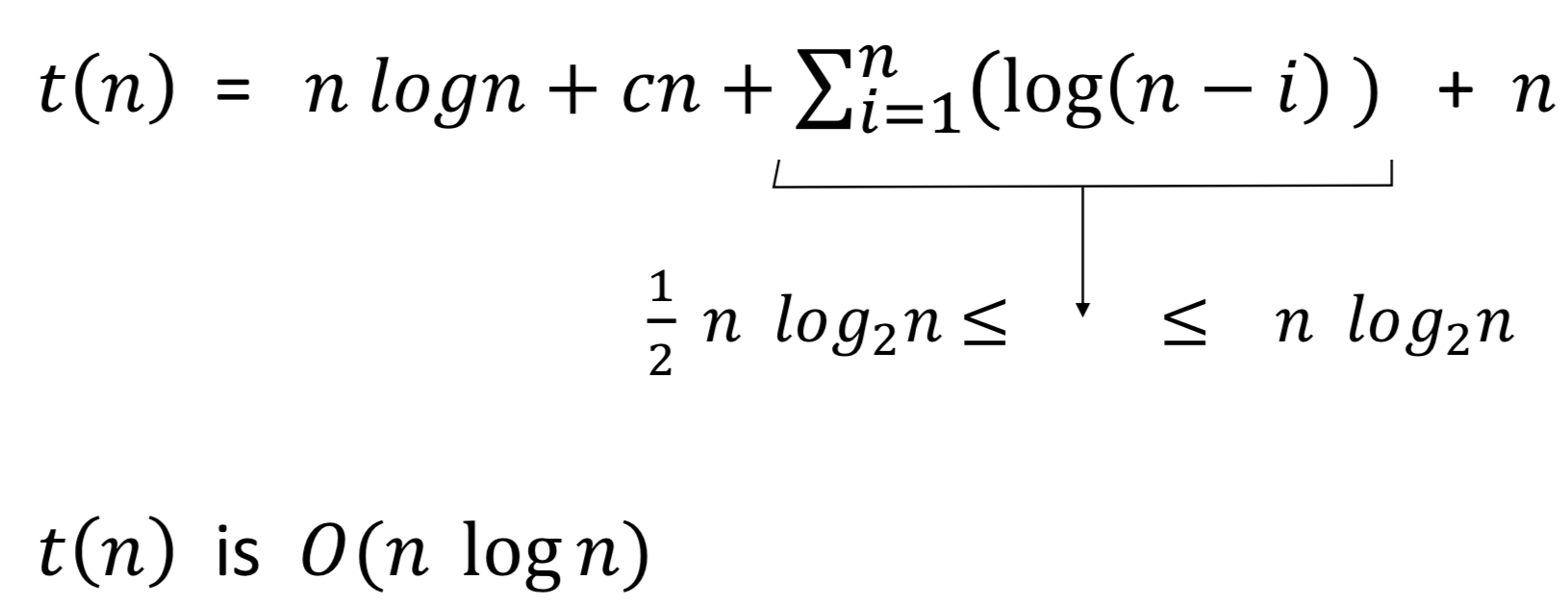
Heapsort

Given a list with 𝑛 = size elements: Build a heap.

Call removeMin() 𝑛 times.







don’t about base(constant)