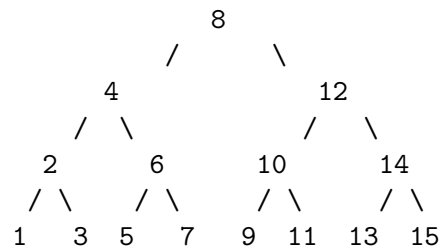


1. (a) This is actually bubble sort. The outer loop iterates n times, the inner loop on average $n/2$ times, so asymptotic complexity $\Theta(n^2)$. It is an in-place sorting algorithm.
- (b) Use the Master Theorem, with $a = 4$ and $b = 2$, so $E = \log 4 / \log 2 = 2$.
 - clearly, cases 1 and 2 do not apply since $E = 2$
 - since $f(n) = n^3 \in \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$, case 3 *might* apply
 - and as $4(n/2)^3 = 1/2 n^3 \leq c f(n)$ for $c = 1/2$, case 3 applies:
 $T(n) \in \Theta(n^3)$
2. (a) This array is already a maxheap, so you do not need to do anything.
- (b) The tree is



If you traverse this tree in a pre-order way you encounter the numbers in the following order:

8-4-2-1-3-6-5-7-12-10-9-11-14-13-15

3. (a) Either you don't put object i in the backpack, so the remainder weight is $R(i-1, g)$, or you do put object i in it, and then the remainder weight $R(i-1, g-w_i)$. You have to choose the minimum of those two choices, so
 - (ii) $R(i, g) = \min\{R(i-1, g), R(i-1, g-w_i)\}$.
- (b) The algorithm to fill the matrix R containing the remainder weights (we assume the indices in w range from 1 to n) :

```
def backpack(w,G):

    n=len(w)
    R=[[0 for g in range(G+1)] for i in range(n+1)]

    for g in range(0,G+1):
        R[0,g]=g          # restgewicht gelijk aan g

    for i in range(1,n+1):
        for g in range(0,G+1):
            if (g-w[i])<0:
                R[i,g]=R[i-1,g]
            else:
                R[i,g]=min(R[i-1,g],R[i-1,g-w[i]])

    return G-R[n,G];
```

The complexity of this algorithm is $\Theta(n^2)$.