Solutions: Algorithms & Data Structures

1. (a) Sorting an array of length 1 or 2 costs 1 comparison. For arrays with more elements we have 3 recursive calls, each with an array of length $\frac{2}{3}$ of the original length. The non-recursive costs are 1 (since we have 1 comparison). This leads to the following recurrent equation for $W(n)$:

$$\begin{array}{lcll} W(n) & = & 1 & \text{for } n < 3 \\ W(n) & = & 3 \cdot W(\frac{2n}{3}) + 1 & \text{for } n \geq 3 \end{array}$$

   We apply the Master theorem: $a = 3$, $b = \frac{3}{2}$, $f(n) = 1$, and $E = \log a / \log b = \log 3 / \log \frac{3}{2}$. Since $f(n) \in O(n^{E-\epsilon})$for e.g. $\epsilon = \frac{1}{2}$ the first case of the Master theorem is applicable, so $W(n) \in \Theta(n^E)$.

   (b) Note that $E = \log 3 / \log \frac{3}{2} = {}^{1.5}\log 3 > 2$ because $1.5^2 = 2.25$. So the worst-case time complexity of *sort* is significantly worse than that of the other sorting algorithms; therefore we never prefer it.

2. (a) in one of the leafs

   (b) A is already a maxheap, so do nothing: complexity $\Theta(0)$.

3. preorder: MGDAHKLTRVUW
   inorder: ADHGKLMRUVTW
   postorder: AHDLKGUVRWTM

4. (a) You can only arrive at cell $(0, j)$ from $(0, j-1)$, and at $(i, 0)$ from $(i-1, 0)$. For $i, j > 0$ we take the minimum of both possibilities to arrive at (i, j), so
   $T(i, j) = \text{cost(i,j)}+\min(\text{T(i-1,j)},\text{T(i,j-1)})$

   (b)

```
def mintrav(cost,M,N):

    T=[[0 for j in range(N)] for i in range(M)]
    T[0][0]=cost[0][0]
    for i in range(1,M): T[i][0]=cost[i][0]+T[i-1][0]
    for j in range(1,N): T[0][j]=cost[0][j]+T[0][j-1]

    for i in range(1,M):
        for j in range(1,N):
            T[i][j] = cost[i][j]+min(T[i-1][j],T[i][j-1])

    return T[M-1][N-1]
```

   The complexity of this algorithm is $\Theta(MN)$.