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 $W(n)$:

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

   (b) We apply the Master theorem with $b = 3$ and $c = 3$, so $E = log3/log3 = 1$. Now $3 \in O(n^{1-\epsilon})$ for some ϵ, so we have case 1, so $T(n) \in \Theta(n)$.

2. (a) Then you could sort n elements by first creating a priority queue by n repeated insertions, and then n times selecting (and deleting) the minimum. This would have complexity $\Theta(n)$, which is impossible as the optimal complexity for sorting is $\Theta(n \log n)$.

   (b) The tree is

```
              A
          /       \
      B               I
    /   \           /   \
  C       F       J       M
 / \     / \     / \     / \
D   E   G   H   K   L   N   O
```

   If you traverse this tree in an in–order way you encounter the letters in the order DCEBGFHAKJLINMO

3. (a)
   - If $v[1..j]$ is empty, then you need i delete's to turn $u[1..i]$ into the empty string, so $D[i, j] = i$ if $j = 0$
   - If $u[1..i]$ is empty, then you need to insert the j elements of $v[1..j]$ into the empty string, so $D[i, j] = j$ if $i = 0$
   - If $u[i] = v[j]$ then you still need to turn $u[1..i-1]$ into $v[1..j-1]$, so so $D[i, j] = D[i-1, j-1]$ if $u[i] = v[j]$
   - Otherwise, you could delete $u[i]$ (and then you still need to turn $u[1..i-1]$ into $v[1..j]$), or you could insert $v[j]$ at the end of $u[1..i]$ (and then you would still need to turn $u[1..i]$ into $v[1..j-1]$). Now you take the minimum of the number of these two possibilities, so $D[i, j] = 1 + min\{D[i-1, j], D[i, j-1]\}$ otherwise

(b)

```
def distance(u,v):
    n=len(u)-1
    m=len(v)-1

    D=[[0 for j in range(m+1)] for i in range(n+1)]

    for i in range(0,n+1):
        D[i,0]=i

    for j in range(0,m+1):
        D[0,j]=j

    for i in range(1,n+1):
        for j in range(1,m+1):
            if u[i]=v[j]:
                D[i,j]=D[i-1,j-1]
            else:
                 D[i,j]=1+min(D[i-1,j],D[i,j-1])

    return D[n,m];
```

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