1. (a) The outer while loop runs n times and the inner while loop an average of $n/2$ times, each with 2 arithmetic operations. So the time complexity is $\Theta(n^2)$.

   (b) We apply the Master theorem: $a = 8$, $b = 2$, $f(n) = n^2+4n+1/n$, and $E = \log a/\log b = 3$. Since $f(n) \in O(n^{E-\varepsilon})$ for e.g. $\varepsilon = \frac{1}{2}$, the first case applies, so

$$W(n) \in \Theta(n^E) \text{ so } W(n) \in \Theta(n^3)$$

2. (a) The smallest element in a minheap has index 0. Swap this with the last element, adapt the length of de heap, and restore the heap property using *heapify*; this last step has complexity $O(log\ n)$.

```
def delmin(E):
    i=E.heapsize-1
    E[0]=E[i]
    E.heapsize= E.heapsize-1
    heapify(E,0)
```

   (b) The node with the biggest element is the root node. For the node with the smallest element: go left if possible, else go right, until this is not possible anymore; the node where you end is the node with the smallest element.

```
def maxmin(N)
    max = N
    x = N
    children = true
    while children
        if x.left != null:
            x = x.left
        else:
            if x.right != null:
                x = x.right
            else:
                children = false
    min = x
    return max, min
```

3. (a)
   - Either you do not include integer a_i in the sum, then the possibilty of forming the sum g with the remaining numbers is given by $R(i-1, g)$

- or you do include a_i in the sum, and now the problem is whether it is possible to form the sum $g - a_i$ with the remaining numbers, which is given by $R(i-1, g-a_i)$
- so $R(i, g)$ is the "or" of these two possibilities

So the correct option is (ii): $R(i, g) = R(i-1, g) || R(i-1, g-a_i)$

(b) The algorithm to fill the boolean matrix `R` (we use the indices in `a` ranging from 1 to n, so we do not use element $a[0]$):

```
def subsum(a,G):

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

    for i in range(0,n+1):
        R[i,0]=1      #always possible to form sum 0

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

    return [n,G];
```

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