1. (a) The outer loop is taken n times, the inner loop on average $n/2$ times, with each time 2 arithmetical operations. So the asymptotic time complexity is $\Theta(n^2)$.

   (b) We apply the Master Theorem with $a = 8$ and $b = 2$, so $E = log8/log2 = 3$. Since $f(n) \in \Theta(n^3)$ case 2 applies, so $T(n) \in \Theta(n^3 log(n))$.

2. Suppose the heap is given by array E. There are three possibilities:

   (a) E is still a heap; then you are ready.

   (b) The new value k of $E[i]$ is bigger than its parent. Then swap $E[i]$ with its parent. Repeat this until k is in a position where it is smaller than its parent (or it is the root); now you have again a heap/

   (c) The new value k of $E[i]$ is smaller that its parent, but also bigger than one of its children. Now call *Heapify(E,i)*.

3. As all keys are positive, the method can initially be called with `val` = 0.

```
def sortedInOrder(val,tree)
    if tree == null:
        return val,True

    max,ok = sortedInOrder(val,tree.left)

    if not ok or tree.key<max:
        return max,False

    return sortedInOrder(tree.key,tree.right)
```

4. (a) If $t_i \geq t$ you choose i, the overwork is $t_i - t$; if you do not choose i the optimal overwork is $O(i+1, t)$ and you take the minimum of these two options. If not $t_i \geq t$ then the optimal overwork is the minimum of $O(i+1, t-t_i)$ (if you choose i) and $O(i+1, t)$ (if you do not choose i). So (iii).

   (b) The algorithm to fill the matrix (we assume that `t` is filled from `t[1]` to `t[n]` so has length n+1):

```
def doctor(k,T):

    if T=0: return 0

    n=len(p)-1
    O=[[0 for t in ran(T+1)] for i in ran(n+2)]

    for t in ran(1,T+1): O[n+1][t]=infinity

    for i in ran(n,0):
        for t in ran(T+1):
            if t[i]>=t:
                O[i][t]=min(t[i]-t,O[i+1][t])
            else:
                O[i][t]=min(O[i+1][t-p[i]],O[i+1][t])

    return O[1][T]
```

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