1. (a) This is actually selection sort. The outer loop iterates n times, the inner loop on average $n/2$ times, with one comparison in the inner loop, so asymptotic compexity $\Theta(n^2)$. It is an in–place sorting algorithm.

   (b) Use the Master Theorem, with $a = 4$ and $b = 2$, so $E = log4/log2 = 2$.

   - 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 $4(n/2)^3 = 1/2n^3 \leq cf(n)$ for $c = 1/2$, case 3 applies: $T(n) \in \Theta(n^3)$

2. (a) Changing item $E[k]$ into K might cause two possible problems in the heap:

   - K might be bigger than the value for the parent of k: then K needs to be swapped upwards until this is no longer the case
   - K might be bigger than the value for the child of k: this can be solved by calling *Heapify*.

```
def update(E,K,k):      # assume k<len(E), K not in E

if E[k] < K :   # K possibly bigger than parent of k
    E[k] = K
    parent = (k-1)//2
    heap = (parent<0 or E[k]>E[parent])
    while not heap:   #move K upwards until E is a heap
        swap(E[k],E[parent])
        k=parent
        parent = (k-1)//2
        heap = = (parent<0 or E[k]>E[parent])
else:           # K possibly smaller than a child of k
    E[k]=K
    heapify(E,k)
```

(b) The tree is

```
              15
            /     \
       7              14
     /  \           /   \
   3      6      10      13
  / \    / \     / \     / \
 1   2  4   5   8   9  11  12
```

3. (a) You get the maximum value by looking at all possible blocks, and adding the price for a block to the value of what remains; and then take the maximum of all these possiblilities. So option a.

   (b)
```
def airtimevalue(price, n):
    val = [0 for x in range(n+1)]

    for m in range(1, n+1):
        max = -1
        for j in range(1, i+1):
             max = max(max, price[j] + value[i-j])
        value[m] = max
    return value[n]
```