1. Recursive algorithm for BubbleSort.

```
// sorts A
    BubbleSort (A) :
        n = |A|
        BS(n)

// BS(i) : sorts A[1...i] using bubble sort.
// assumes that A is global
    BS(i):
        if i = 0 : return
        PutLast (i)
        BS (i-1)

// assume that A[(i+1)...n] is sorted initially.
// puts the largest element of A[1...i] at A[i] so that
   A[i...n] becomes sorted.
    PutLast (j) :
        if j = 0 ; return
        PutLast (j-1)
        if A[j] > A[j+1]
            swap ( A[j], A[j+1])
```

2. Proof of correctness of BubbleSort (A).

// If $A[(i+1)...n]$ is sorted initially, when calling BS(i), then $A[i...n]$ is finally sorted. We can also say this as, if $A[1... (n-i)]$ (starting count from last index and moving backward) is sorted initially then $A[1... (n-i+1)]$ will be sorted finally.

Base case : If $A[(n+1)...n]$ is sorted inially, then $A[n...n]$ is finally sorted. LHS is trivially true.

Induction Hypothesis : Assume that PutLast(j) is correct whenever $j <= k$.

Induction claim : To show that PutLast($k+1$) acts correctly, i.e., if $A[1 \ldots (n-(k+1))]$ is sorted initially before calling PutLast($k+1$), then $A[1 \ldots (n-(k+1)+1)]$ will be sorted finally.
(here index counting is done starting from end and moving backward)

When PutLast($k+1$) is called :
Line 2 is no operation. In Line3 PutLast(k) will be called and by Induction Hypothesis PutLast(k) will work correctly.
When $j=0$ condition is reached by recursive PutLast(j) calls, after that the actual computation starts, for each PutLast(j) calls Line4 is executed (where $1 <= j <= k+1$) and if $A[j] > A[j+1]$ then the larger element moves toward right of the array (bubbles up) by the swapping operation of Line5. If $A[j] <= A[j+1]$ then, no swap takes place and the control returns to the previous recursive call. Thus when the computation of PutLast($k+1$) call takes place, then the largest element of $A[1 \ldots (k+1)]$ is already placed at $A[k+1]$, so line5 is not executed and the control returns to BS(i).
Now after Line3 of BS(i) method the largest element of $A[1 \ldots (k+1)]$ is placed at $A[k+1]$ or $A[n-(k+1)+1]$ (counting from the back). We know that the largest $(n-(k+1))$ elements were already present in $A[1 \ldots (n-(k+1))]$ (counting from the back). Now after Line3 $A[n-(k+1)+1]$ becomes the minimum element of the subarray $A[1 \ldots (n-(k+1)+1)]$, thus $A[1 \ldots (n-(k+1)+1)]$ becomes sorted finally.

3. Time complexity analysis of recursive BubbleSort().

Let $T(n)$ denote the time complexity of BubbleSort()

Let $B(n)$ denote the time complexity of BS(n)

Let $P(n)$ denote the time complexity of PutLast(n)

$$T(n) = B(n) + O(1)$$

$$B(n) = B(n-1) + P(n)$$

$$\begin{aligned} P(n) &= P(n-1) + c \\ &= P(n-2) + c + c \\ &= P(n-3) + c + c + c \\ &\vdots \\ &= P(n-k) + kc \end{aligned}$$

for base case, $n-k=0 \Rightarrow k=n$

$$\begin{aligned} P(n) &= P(0) + nc \\ &= 1 + nc \end{aligned}$$

$$P(n) \cong O(n)$$

Now,
$$\begin{aligned} B(n) &= B(n-1) + nc + 1 \\ &= B(n-2) + (n-1)c + nc + 1 + 1 \\ &= B(n-3) + (n-2)c + (n-1)c + nc + 1 + 1 + 1 \\ &= B(n-k) + c\,[n + (n-1) + \cdots + (n-k+1)] + k*1 \end{aligned}$$

for base case $n-k=0 \Rightarrow k=n$

then
$$\begin{aligned} B(n) &= B(0) + c\,[n + (n-1) + (n-2) + \cdots + 2 + 1] + n \\ &= 1 + c * \frac{n(n+1)}{2} + n \end{aligned}$$

$$B(n) \cong O(n^2)$$

Therefore, $T(n) \cong O(n^2)$

Time complexity of BubbleSort(A) is $O(n^2)$.