ascending order. Before the start of the next iteration i updates to $(i+1)$ and it's easy to see that the loop invariant is preserved.

Termination: The outer loop terminates when $i = a[0].col$. We already know that $a[0].col$ stores the no. of columns in $a[\,]$.

By the loop invariant, the transpose of all non-zero elements in columns 0 to $a[0].col - 1$ (i.e. all the columns of matrix a) have been correctly placed in $b[\,]$ as triples $\langle row, col, value \rangle$, where $row = a[j].col$, $col = a[j].row$, and $value = a[j].value$. These entries occupies the first currentb posns. of b and are arranged such that: 1) The row fields in $b[1]$ to $b[currentb - 1]$ are in ascending order. 2) For entries with the same row, the col fields are in ascending order.

$\therefore$ matrix $b[\,]$ stores the transpose of matrix $a[\,]$, follows the criterion of sparse matrix representation. Hence, the function is correct $\square$

Time Complexity: $O(\text{columns} \times \text{elements})$ (Easy to see)

- This time is a little disturbing since we know that if we represented our matrices as two-dimensional arrays of size rows x columns, we could obtain the transpose in $O(\text{rows} \cdot \text{columns})$ time. The algorithm to accomplish this has the simple form:

```
for(j=0; j<columns; j++)
  for(i=0; i<rows; i++)
    b[j][i] = a[i][j];
```

The $O(\text{columns} \cdot \text{elements})$ time for our transpose function becomes $O(\text{columns}^2 \cdot \text{row})$ when the number of elements is of the order columns · rows.