```
* Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
void swap(int** a, int** b)
    int *tmp;
    tmp = *a;
*a = *b;
*b = tmp;
}
int partition(int** nums, int 1, int r)
    int pivot;
   int index;
   int base;
    base = 1-1;
   pivot = nums[r][0];
    for(index = 1; index < r; index++)</pre>
        if (nums[index][0] < pivot)</pre>
            base++;
            swap(&nums[index],&nums[base]);
    }
    base++;
    swap(&nums[base], &nums[r]);
    return base;
}
void quickSort(int** nums, int 1, int r)
        int pivot_pos;
        pivot_pos = partition(nums, 1, r);
        quickSort(nums, 1, pivot_pos-1);
quickSort(nums, pivot_pos+1, r);
bool _merge(int** intervals, int pos, int merge_pos) {
   bool result;
    result = false;
    if( intervals[pos][1] >= intervals[merge_pos][0] )
        if(intervals[pos][1] < intervals[merge_pos][1])</pre>
            intervals[pos][1] = intervals[merge_pos][1];
        result = true;
    return result;
int** merge(int** intervals, int intervalsSize, int* intervalsColSize, int* returnSize, int** returnColumnSizes) {
    int index;
    int base = 0;
    *returnSize = 0;
    *returnColumnSizes = (int*) malloc(sizeof(int)*intervalsSize);
    quickSort(intervals, 0, intervalsSize-1);
    for(index = 1 ; index <= intervalsSize; index++)</pre>
        if( (index == intervalsSize) || (false == _merge(intervals, base, index)) )
             intervals[*returnSize][0] = intervals[base][0];
intervals[*returnSize][1] = intervals[base][1];
             (*returnColumnSizes)[*returnSize] = 2;
             (*returnSize)++;
             base = index;
        }
    return intervals;
```