```c
/**
 * 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().
 */
#define SOLUTION_FIRST          (1)
#define SOLUTION_SECOND         (0)


#if(SOLUTION_FIRST)
void _subsetsWithDup(int* nums, int numsSize, int* returnSize, int** returnColumnSizes, int pos, int tmp_index, int** result, int* tmp, bool select) {

    int index;

    if(tmp_index >= numsSize)
    {
        printf("Push, %d, %d\n", tmp_index, numsSize);
        result[*returnSize] = (int*)malloc(sizeof(int)*(pos+1));
        memcpy(result[*returnSize], tmp, sizeof(int)*(pos+1));
        (*returnColumnSizes)[*returnSize] = pos+1;
        (*returnSize)++;
    }else
    {

  if( ( (tmp_index == 0) || (tmp_index > 0 && (nums[tmp_index] != nums[tmp_index-1]) && select == false) || select == true) )
  {

   tmp[pos+1] = nums[tmp_index];
   _subsetsWithDup(nums, numsSize, returnSize, returnColumnSizes, pos+1, tmp_index+1, result, tmp, true);
  }

  _subsetsWithDup(nums, numsSize, returnSize, returnColumnSizes, pos, tmp_index+1, result, tmp, false);
    }
}

int** subsetsWithDup(int* nums, int numsSize, int* returnSize, int** returnColumnSizes) {
    int** result;
    int* nums_map;
    int* tmp;
    int min;
    int max;
    int index;
    int nums_index;

    tmp = (int*)malloc(sizeof(int)*numsSize);
    result = (int**)malloc(sizeof(int*) * (1<<numsSize));
    *returnColumnSizes = (int*)malloc(sizeof(int) * (1<<numsSize));
    *returnSize = 0;

    min = INT_MAX;
    max = INT_MIN;


    for(index = 0; index < numsSize; index++)
    {
        if(nums[index] > max)
        {
            max = nums[index];
        }

        if(nums[index] < min)
        {
            min = nums[index];
        }
    }

    nums_map = (int*)calloc(sizeof(int), (max - min + 1));

    for(index = 0; index < numsSize; index++)
    {
        nums_map[nums[index] - min]++;
    }


    for(index = 0, nums_index = 0; index < (max - min + 1) && (nums_index < numsSize); index++)
    {
        while(nums_map[index])
        {
            nums[nums_index] = index + min;
            nums_index++;
            nums_map[index]--;
        }
    }

    _subsetsWithDup(nums, numsSize, returnSize, returnColumnSizes, -1, 0, result, tmp, false);

    return result;
}
#elif(SOLUTION_SECOND)
int** subsetsWithDup(int* nums, int numsSize, int* returnSize, int** returnColumnSizes) {
    int** result;
    int* tmp;
    int* nums_map;
    int min;
    int max;
    int nums_index;
    int index;
    int bit_index;
    int tmp_index;
    bool found;


    tmp = (int*)malloc(sizeof(int)*numsSize);
    result = (int**)malloc(sizeof(int*) * (1<<numsSize));
    *returnColumnSizes = (int*)malloc(sizeof(int) * (1<<numsSize));
    *returnSize = 0;

    min = INT_MAX;
    max = INT_MIN;


    for(index = 0; index < numsSize; index++)
    {
        if(nums[index] > max)
        {
            max = nums[index];
        }

        if(nums[index] < min)
        {
            min = nums[index];
        }
    }

    nums_map = (int*)calloc(sizeof(int), (max - min + 1));

    for(index = 0; index < numsSize; index++)
    {
        nums_map[nums[index] - min]++;
    }


    for(index = 0, nums_index = 0; index < (max - min + 1) && (nums_index < numsSize); index++)
    {
        while(nums_map[index])
        {
            nums[nums_index] = index + min;
            nums_index++;
            nums_map[index]--;
        }
    }

    for(index = 0; index < (1<<numsSize); index++)
    {
        tmp_index = 0;
        found = true;

        for(bit_index = 0; bit_index < numsSize; bit_index++)
        {
            if(  (bit_index > 0) &&
                 (nums[bit_index] == nums[bit_index-1]) &&
                 (index & (1 << bit_index )) &&
                 ( (index & (1 << (bit_index-1)) ) == 0)
              )
            {
                found = false;
                break;
            }

            if(index & (1 << bit_index))
            {
                 tmp[tmp_index] = nums[bit_index];
                 tmp_index++;
            }

        }

        if(found)
        {
            result[*returnSize] = (int*)malloc(sizeof(int)*(tmp_index));
            memcpy(result[*returnSize], tmp, sizeof(int)*(tmp_index));
            (*returnColumnSizes)[*returnSize] = tmp_index;
            (*returnSize)++;
        }
    }

    return result;
}
#endif
```