struct hashtable

{

    int value;

    int cnt;

};

int cmp(const void \*a, const void \*b)

{

    return \*(const int \*)a - \*(const int \*)b;

}

int cmph(const void \*a, const void \*b)

{

    return ((struct hashtable \*)b)->cnt - ((struct hashtable \*)a)->cnt;

}

int\* topKFrequent(int\* nums, int size, int k, int\* returnSize) {

    if(k == 0 || size == 0){

        \*returnSize = k;

        return 0;

    }

    if (size < k){

        \*returnSize = 0;

        return 0;

    }

    if(size == k && k == 1){

        \*returnSize = 1;

        return nums;

    }

    int \*ans = calloc(k, sizeof(int));

    struct hashtable hash[size];

    memset(hash,0,sizeof(struct hashtable)\*size);

    int i;

    int count=1;

    \*returnSize = k;

    qsort(nums, size,sizeof(int),cmp);

    hash[0].value=nums[0];

    hash[0].cnt++;

    for (i=1;i<size;i++){

        if(hash[count-1].value == nums[i]){

            hash[count-1].cnt++;

        } else {

            count++;

            hash[count-1].value = nums[i];

            hash[count-1].cnt++;

        }

    }

    qsort(hash, count,sizeof(hash[0]),cmph);

    for(i=0;i<k;i++){

        ans[i] = hash[i].value;

    }

    return ans;

}

/\*\*

 \* Note: The returned array must be malloced, assume caller calls free().

 \*/

typedef struct {

    int val;

    int freq;

} Pair;

int compare(const void \*a, const void \*b) {

    return \*(int\*) a - \*(int\*) b;

}

int comparePairs(const void \*a, const void \*b) {

    Pair p1 = \*(Pair\*) a, p2 = \*(Pair\*) b;

    return p2.freq - p1.freq;

}

int\* topKFrequent(int\* nums, int numsSize, int k, int\* returnSize){

    int numBuckets[20001] = {0};

    Pair pairs[100001];

    int i, \*res, unique=0;

    for (i = 0; i < numsSize; ++i) {

        ++numBuckets[nums[i]+10000];

    }

  for (i = 0; i < 20001; ++i) {

    if (numBuckets[i]) {

        pairs[unique].val = i - 10000;

        pairs[unique].freq = numBuckets[i];

        ++unique;

    }

  }

    qsort(pairs, unique, sizeof(Pair), comparePairs);

    res = (int\*) malloc(k \* sizeof(int));

    for (i = 0; i < k; ++i) {

        res[i] = pairs[i].val;

    }

    \*returnSize = k;

    return res;

}