给你输入一个数组nums和一个正整数k，请你判断nums是否能够被平分为元素和相同的k个子集。

分析：

1. 将数组等分为k份，首先计算数组和是否能够被k平分，如果不行则返回；
2. 开始回溯，回溯的对象为bucket和target = sum / k;
3. 为方便剪枝，将数组递减排序；
4. 对指定index将其尝试放入k个桶中，放入后开始回溯，当回溯为true时直接返回。

```
class Solution {
public:
  bool canPartitionKSubsets(std::vector<int> &nums, int k) {
    int len = nums.size();
    if (len < k) {
      return false;
    }

    std::sort(nums.begin(), nums.end(), [](int a, int b) {
      return a > b;
    });

    int sum = 0;
    for (auto iter : nums) {
      sum += iter;
    }
    if (sum % k != 0) {
      return false;  // 不能等分成k份
    }

    // 桶的集合
    std::vector<int> bucket = std::vector<int>(k, 0);
    // 每个桶中的和
    int target = sum / k;

    return backtrack(nums, nums.size(), 0, target, bucket);
  }

private:
  // 递归枚举nums中的每个数字
  bool backtrack(std::vector<int> &nums,
                 int               len,
                 int               index,
                 int               target,
                 std::vector<int> &bucket) {
    if (index == len) {
      // 数组遍历完成，判断每个bucket中值是否为target
      for (auto iter : bucket) {
        if (iter != target) {
          return false;
```

```
          }
        }

        return true;
      }

      // 遍历桶
      for (int i = 0; i < bucket.size(); i++) {
        // 桶装满了, continue
        if (nums[index] + bucket[i] > target) {
          continue;
        }

        // 放入桶中
        bucket[i] += nums[index];
        if (backtrack(nums, len, index + 1, target, bucket)) {
          return true;
        }
        // 回溯
        bucket[i] -= nums[index];
      }

      // nums[index]放入哪个桶中都不行
      return false;
    }
};
```