# 题目

给你一个下标从0开始的整数数组nums和一个整数k。你需要执行以下操作恰好k次，最大化你的得分：

从nums中选择一个元素m。

将选中的元素m从数组中删除。

将新元素m + 1添加到数组中。

你的得分增加m。

请你返回执行以上操作恰好k次后的最大得分。

示例 1：

输入：nums = [1,2,3,4,5], k = 3

输出：18

解释：我们需要从 nums 中恰好选择 3 个元素并最大化得分。

第一次选择 5 。和为 5 ，nums = [1,2,3,4,6] 。

第二次选择 6 。和为 6 ，nums = [1,2,3,4,7] 。

第三次选择 7 。和为 5 + 6 + 7 = 18 ，nums = [1,2,3,4,8] 。

所以我们返回 18 。

18 是可以得到的最大答案。

示例 2：

输入：nums = [5,5,5], k = 2

输出：11

解释：我们需要从 nums 中恰好选择 2 个元素并最大化得分。

第一次选择 5 。和为 5 ，nums = [5,5,6] 。

第二次选择 6 。和为 6 ，nums = [5,5,7] 。

所以我们返回 11 。

11 是可以得到的最大答案。

提示：

1 <= nums.length <= 100

1 <= nums[i] <= 100

1 <= k <= 100

# 分析

## 方法一：贪心

思路：

我们可以使用贪心算法。基本思想是每次选择当前数组中的最大值，将其从数组中删除，并将增加1后的新值添加到数组中。这样，每次操作都能使得得分增加最多。

首先定义了一个变量score来记录得分。然后，它使用std::max\_element函数来找到数组nums中的最大值及其索引。找到最大值后，将其从数组中删除，并将增加1后的新值添加到数组的末尾。同时，将最大值的原始值加到score中。这个过程重复执行k次。

注意，在删除和添加元素时，我们需要更新数组的大小。删除元素时，我们使用nums.erase函数，它接受一个迭代器指向要删除的元素。添加元素时，我们使用nums.push\_back函数。

最后，返回score作为结果，即执行k次操作后的最大得分。

代码：

class Solution {

public:

int maximizeSum(std::vector<int>& nums, int k) {

int score = 0; // 记录得分

// 执行k次操作

for (int i = 0; i < k; ++i) {

// 找到当前数组中的最大值

int maxIndex = std::max\_element(nums.begin(), nums.end()) - nums.begin();

int maxVal = nums[maxIndex];

// 将最大值从数组中删除

nums.erase(nums.begin() + maxIndex);

// 将新元素添加到数组中

nums.push\_back(maxVal + 1);

// 更新得分

score += maxVal;

}

return score;

}

};

或：

class Solution {

public:

int maximizeSum(std::vector<int>& nums, int k) {

int score = 0; // 记录得分

// 执行k次操作

for (int i = 0; i < k; ++i) {

int maxVal = nums[0]; // 假设第一个元素是最大值

int maxIndex = 0; // 最大值的索引

// 遍历数组找到最大值

for (int j = 1; j < nums.size(); ++j) {

if (nums[j] > maxVal) {

maxVal = nums[j];

maxIndex = j;

}

}

// 将最大值从数组中删除

nums.erase(nums.begin() + maxIndex);

// 将新元素添加到数组中

nums.push\_back(maxVal + 1);

// 更新得分

score += maxVal;

}

return score;

}

};