A：van游戏

根据题意可知，数组 `nums` 的长度为 `n`，此时设选中小朋友人数为 `k`，此时 `k∈[0,n]`，`k` 应满足如下：

- 所有满足 `nums[i]<k` 的小朋友被选中；
- 所有满足 `nums[i]>k` 的小朋友不应被选中；
- 不能存在 `nums[i]=k` 的小朋友，因为每个小朋友只有选或不选两种可能。

这意味着在选择小朋友人数固定的时候，选择方案是唯一的。把 `nums` 从小到大排序后，唯一性可以更明显地看出来：

- 以 `k` 为分界线，左边的都要选，右边的都不能选。
- 排序后：
  - 如果选了 `nums[i]`，那么比 `nums[i]` 更小的小朋友也要选。
  - 如果不选 `nums[i]`，那么比 `nums[i]` 更大的小朋友也不选。

具体地，如果选 `nums[i-1]` 而不选 `nums[i]`，由于数组已排序，我们必须要选下标为 0,1,2,⋯,i−1 的小朋友，一共 `i` 个，而下标 ≥i 的小朋友都不能选，所以需要满足 `nums[i-1]<i<nums[i]`。

枚举 `i=1,2,⋯,n-1`（枚举分界线的位置），如果上式成立，就意味着我们可以选 `i` 个小朋友，算作一种方案。

特殊情况：

- 如果 `nums[0]>0`，那么可以一个小朋友都不选。
- 如果 `nums[n-1]<n`，那么可以所有小朋友都选。由于数据范围保证 `nums[i]<n`，所以这种方案一定存在。

参考代码

```cpp
#include<bits/stdc++.h>
using namespace std;

int countWays(vector<int>& nums,int n) {
    sort(nums.begin(), nums.end());
    int ans = 0;
    ans=nums[0]>0;
    for (int i = 1; i < n; i++) {
        ans += nums[i - 1] < i && i < nums[i];
    }
    return ans + 1;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    cout << countWays(nums,n) << endl;
    return 0;
}
```

B：蒂蒂和鲨鲨的头脑博弈游戏

打表找规律，可以发现 `n=1` 到 `n=6` 均为鲨鲨获胜，胆大的同学可能就直接输出鲨鲨获胜了，那恭喜你，猜对了，推理发现， `n>6` 时，经过先手若干次操作后鲨鲨一定能拿到一个 `n=3~6` 的局面，此时蒂蒂之前操作的异或和为 `0~3` ，通过分析这（4*4）16 种情况，鲨鲨都能获胜，因此，鲨鲨必胜，蒂蒂必败。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int n;
        cin >> n;
        cout << "shasha" << '\n';
    }
}
```

C：生日蛋糕就是青春的墓碑

由题干中的条件不难看出，若赠送的礼物价值全为奇数或全为偶数时，答案即为礼物中价值的最大值。由题干中的条件分析，要使得最终的答案最大，就要将更多的价值利用起来，由于存在 `a_i -x` ， `a_j +x` 的操作，因此，我们能将原 `a_j` 变成下次操作所需的 `a_i` （将奇数变成偶数），因此最优情况就是 `x=a_i -1` ，即每次操作损失 1 点价值，那总共会进行多少次操作呢？答案是礼物中价值为奇数的个数，每次操作可以看成 `a_j` 损失 1 点价值， `a_i` 的价值全贡献给 `a_j` ，由于最后一次操作要使得答案最大，因此将 `a_i` 的价值贡献给 `a_j` ，同时 `a_j` 不需要损失价值，因此最终答案为 `sum(a_1 ,a_2 ...a_n )-cnt_奇 +1`

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int main()
{
    int n, cnt = 0;
    cin >> n;
    vector<int> a(n);
    ll sum = 0;
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
        if (a[i] & 1)
            cnt++;
        sum += a[i];
    }
    if (cnt == 0 || cnt == n)
        cout << *max_element(a.begin(), a.end()) << endl;
    else
        cout << sum - cnt + 1 << endl;
}
```

D:暴力求和

将题干的式子展开 $\sum_{i=1}^{k} b_i c_i - c_i + \sum a$，化简后变为 $\sum_{i=1}^{k} (b_i - 1)c_i - \sum a$，即找一个序列 `c` 使得答案最小，考虑动态规划，定义一个一维数组 `dp`，其中 `dp[j]` 表示前 `i` 个元素中选取 `j` 个元素时的最小和，对于每个元素 `a[i]`，从后往前更新 `dp` 数组，确保每个元素只能被选择一次。更新公式为：`dp[j]=min(dp[j],dp[j-1]+(b[j-1]-1)*a[i])`
更新结束后，`dp[j]` 即为 $\sum_{i=1}^{k} (b_i - 1)c_i$，最后加上 $\sum a$ 完成。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 0x7f7f7f7f7f7f7f7f;
int main()
{
    int n, k;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cin >> k;
    vector<int> b(k);
    for (int i = 0; i < k; i++)
        cin >> b[i];
    vector<ll> dp(k + 1, INF);
    dp[0] = 0;
    for (auto x : a)
    {
        for (int j = k; j >= 1; j--)
        {
            if (dp[j - 1] != INF)
            {
                dp[j] = min(dp[j - 1] + (b[j - 1] - 1) * x, dp[j]);
            }
        }
    }
    cout << dp[k] + accumulate(a.begin(), a.end(), 0LL) << endl;
}
```