# **Задача АK6. МУЛТИСОРТ**

*Хъмпти-Дъмпти седеше на стената,*

*Карта след карта падна на земята.*

*Всички царски хора и царските коне*

*не могат вече да ги сортират по местата им, не!*

Както стихотворението казва, Хъмпти-Дъмпти изтървал тестето си от с карти и сега никой не може да го нареди в правилния ред. Затова той помолил Алиса за помощ. Тя е чувала, че има бързи методи за сортиране, които се базират на сравняване на елементите по двойки, но това ѝ се струва доста неефикасно. Все пак тя може да хване до карти в ръката си и веднага да знае в какъв ред трябва да са те. Чуди се дали няма някой метод за сортиране, който може да се възползва от това нейно умение. Нейната цел е да минимизира броя такива -странни сравнения, които ще извърши, за да подреди тестето.

След дълго мислене, тя не се сетила за добър метод. Затова сега тя пита Вас за помощ. Помогнете ѝ като напишете програма **multisort.cpp**, която да комуникира с Алиса (програмата на журито), за да сортира картите с колкото се може по-малко сравнения.

## *Детайли по имплементацията*



Вашата функция multisort трябва да има следния прототип:

std::vector<int> multisort(int n, int k);

Тя ще бъде извикана точно веднъж и ще получи като аргументи броя карти за сортиране и броя карти, които Алиса може да сравни наведнъж. Функцията трябва да върне вектор, който съдържа пермутация на числата от до . Това е пермутацията, в която трябва да се наредят картите, така че те да са подредени в нарастващ ред. Картите определяме с техните индекси, т.е. позициите им в началното разбъркване.

Функция compare на журито има следния прототип:

void compare(std::vector<int>& elems);

Вашата програма може да я вика колкото си пъти иска. Като аргумент ѝ се дава референция към вектор съдържащ индексите на картите, които да бъдат сравнени от Алиса. Векторът трябва да съдържа до валидни индекса на карти, но е позволено да съдържа повторения. Функцията ще сортира вектора, така че указаните карти да са в нарастващ ред. Това ще се случи със сложност , където е дължината на вектора.

Вашата програма трябва да имплементира функцията multisort, но не трябва да съдържа функция main. Освен това, тя не трябва да чете от стандартния вход или да печата на стандартния изход. Програмата Ви също така трябва да включва хедър файла **multisort.h** чрез указание към препроцесора:

#include "multisort.h"

Стига да спазва тези условия, програмата Ви може да съдържа каквито и да е помощни функции, променливи, константи и прочие.

## *Ограничения*



във всички тестове

## *Оценяване*



Всеки тест се оценява поотделно. За даден тест Вашето решение ще получи точки, различни от 0, ако функцията multisort успешно приключи изпълнение и върне вектор с дължина , който съдържа правилно сортираната пермутация от индекси на карти. Точките, които ще получите на теста са равни на максималният брой точки за теста, умножени по:

Тук е броя сравнения, които е използвало вашето решение, а е броя сравнения, които е използвало авторовото решение в най-лошия от 10 опита. По-точно, авторовото решение е пуснато на 30 теста с различни пермутации, но с еднакви и (един от тези тестове е този, на който Вашето решение бива тествано). След това за е взет най-лошият (максималният) от десетте резултата, които авторовото решение е постигнало.

Очакваният брой сравнения, които авторовото решение ще използва за дадени и като сложност е равен на теоретичния минимум (получен, като се разглежда колко информация дава едно сравнение). По-конкретно, за не твърде малки стойности на авторовото решение използва около два пъти повече сравнения от теоретичния минимум.

## *Тестове*



|  |  |
| --- | --- |
| **Процент тестове** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*Пермутацията на картите в даден тест ще бъде произволно генерирана, така че всяка пермутация има равен шанс да се падне.*

## *Локално тестване*



Предоставени Ви са файловете **multisort.h** и **Lgrader.cpp**, които можете да компилирате заедно с Вашата програма, за да я тествате.

При стартиране на програмата трябва да се въведат както и сийд за генериране на пермутацията. Ако искате да я конфигурирате по друг начин, може да правите каквито си промени искате по предоставените Ви файлове.

Гарантирано е, че грейдърът в системата ще се държи както предоставения Ви локален грейдър. Най-съществено, имплементацията на функцията compare ще е същата.

## *Примерна комуникация*



|  |  |  |
| --- | --- | --- |
| № | Действия на multisort | Действия и отговори на журито |
| 1. |  | multisort(4, 3) |
| 2. | compare({0, 1, 2}) | elems = {1, 0, 2} |
| 3. | compare({1, 2, 3}) | elems = {1, 3, 2} |
| 4. | compare({0, 3}) | elems = {3, 0} |
| 5. | return {1, 3, 0, 2} |  |

## *Обяснение*



(с цел демонстрация) и .

Тук стойностите на четирите карти са: .

Първото сравнение открива, че: .

Второто сравнение открива, че: .

Третото сравнение открива, че: .

Единственото възможно решение е: .

Програмата на състезателя общо е използвала 3 сравнения.