МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**ФАКУЛЬТЕТ СИСТЕМ УПРАВЛЕНИЯ И РОБОТОТЕХНИКИ**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**(1005, 1296, 2025)**

Выполнили:

Хюинь Тан Куонг(336231)

Преподаватель:

Tropchenko Andrey Aleksandrovich

**A picture containing shape

Description automatically generated**

Санкт-Петербург, 2023

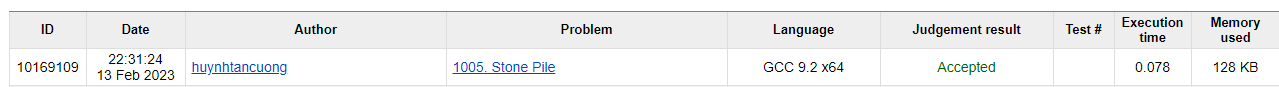
# 1005.

## Problem description

Timeline

Description automatically generated

## Timus system acceptance:



## Code:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

/// @brief This function return the different betweent two piles of stone. If the bit from bitMask is 1, the stone will be added to the first pile. And the second pile for the bit 0.

/// @param bitMask bitMask

/// @param n size of bitMask

/// @param a array of stones

/// @return long int different between 2 piles

long int getDifference(int bitMask, int n, int a[]) {

    long int first\_pile = 0, second\_pile = 0;

    for (int i = 0; i<n; i++) {

        if (bitMask&(1<<i)) {

            first\_pile += a[i];

        } else {

            second\_pile += a[i];

        }

    }

    return abs(first\_pile-second\_pile);

}

/// @brief This function print the bit by bit in the bitMask. Just for debugging

/// @param bitMask bitMask

/// @param size size of bitMask

void printBitMask(int bitMask, int size) {

    int bit;

    for (int i = 0; i<size; i++) {

        bit = bitMask & (1<<i);

        if (bit) bit = 1;

        printf("%d", bit);

    }

    printf("\n");

}

int main() {

    // Input n and array

    int n;

    scanf("%d", &n);

    int \*arr;

    arr = malloc(n\*sizeof(int));

    for (int i = 0; i< n; i++) {

        scanf("%d", &arr[i]);

    }

    long int min = 100000000; // set the min to a big int value

    int size = pow(2, n); // n bit represent the value of 2^n

    // find the minimum value of different between 2 piles

    for (int bitMask = 0; bitMask < size; bitMask++) {

        long int diff = getDifference(bitMask, n, arr);

        if (min > diff) min = diff;

        if (min == 0) break;

    }

    // Print the output

    printf("%ld", min);

    // Free the array

    free(arr);

    return 0;

}

## Output:

5

5 8 13 27 14

3

## Algorithm description:

- The main idea is we loop through all the possible ways that put stones into the 2 piles and find the minimum of difference.

- To loop through all the possible ways, we create all the subset of stone’s array.

- A bitmask was used to create the subsets. If the bit is 1, we add the stone to the first pile. If the bit is 2, we add the stone to the second pile.

- Because numbers are stored on the machine in binary format. So, we just need to loop from 0 to 2n to loop through all the subset of n bit length binary number.

- In the process of looping, we store the minimum number of difference between two piles.

# 1296.

## Problem description

Table

Description automatically generated

## Timus system acceptance:



## Code:

#include <stdio.h>

#include <stdlib.h>

/// @brief This function find the maximum continuous subarray

/// @param a array of weights

/// @param size size of the array

/// @return maximum sum of continuous subarray

int maxSubArraySum(int a[], int size) {

    int max\_so\_far = 0;

    int max\_ending\_here = 0;

    for (int i = 0; i<size; i++) {

        max\_ending\_here += a[i];

        if (max\_so\_far < max\_ending\_here) max\_so\_far = max\_ending\_here;

        if (max\_ending\_here < 0) max\_ending\_here=0;

    }

    return max\_so\_far;

}

int main() {

    // Input

    int n;

    scanf("%d", &n);

    int \*arr;

    arr = malloc(n\*sizeof(int));

    for (int i = 0; i< n; i++) {

        scanf("%d", &arr[i]);

    }

    // Find the maximum

    int max = maxSubArraySum(arr, n);

    // Output

    printf("%d", max);

    free(arr); // free memory

    return 0;

}

## Ouput:

10

31

-41

59

26

-53

58

97

-93

-23

84

// output

187

## Algorithm description:

- The idea of Kadane’s algorithm is to maintain a variable *max\_ending\_here* that stores the maximum sum contiguous subarray ending at current index and a variable *max\_so\_far* stores the maximum sum of contiguous subarray found so far, Everytime there is a positive-sum value in *max\_ending\_here* compare it with *max\_so\_far* and update *max\_so\_far* if it is greater than *max\_so\_far*.

- So the main Intuition behind Kadane’s algorithm is, the subarray with negative sum is discarded (by assigning max\_ending\_here = 0 in code). And we carry subarray till it gives positive sum.

Pseudo code:

Initialize:

max\_so\_far = INT\_MIN

max\_ending\_here = 0

Loop for each element of the array

(a) max\_ending\_here = max\_ending\_here + a[i]

(b) if(max\_so\_far < max\_ending\_here)

max\_so\_far = max\_ending\_here

(c) if(max\_ending\_here < 0)

max\_ending\_here = 0

return max\_so\_far

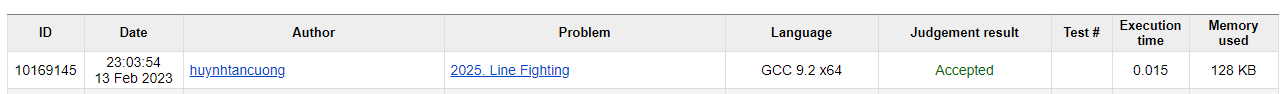
# 2025.

## Problem description

Graphical user interface, text, application

Description automatically generated

## Timus system acceptance:



## Code:

#include <stdio.h>

/// @brief This function calculate the matches between fighters in diffent teams.

/// @param b amount of basic team

/// @param a amount of advanced team

/// @param va amount of team member in advanced team

/// @param vb amount of team member in basic team

/// @return matches between fighters in diffent teams

int f(int b, int a, int va, int vb) {

    if (b > 0) {

        return (b-1)\*(vb\*vb) + a\*(vb\*va) + f(b-1, a, va, vb);

    }

    if (a > 0) {

        return (a-1)\*(va\*va) + b\*(va\*vb) + f(b, a-1, va, vb);

    }

    return 0;

}

int main() {

    // Input

    int T;

    scanf("%d", &T);

    int n, k;

    // Find answer for every (n, k), where n - number of fighters, k - number of teams.

    for (int i = 0; i<T; i++) {

        // Input n and k

        scanf("%d %d", &n, &k);

        int a, b, va, vb;

        /\*\*

         \* a - number of advanced team, which is the team with more member (n/k+1)

         \* b - number of basic team, which is the team with basic member (n/k)

         \* va - the amount of team member in advanced team.

         \* vb - the amount of team member in basic team.

        \*/

        a = n%k;

        b = k - a;

        va = n/k +1;

        vb = n/k;

        int max = f(b, a, va, vb);

        printf("%d\n", max);

    }

    return 0;

}

## Ouput:

1

6 3

12

## Algorithm description:

* In order to maximize the matches between different teams, we must distribute fighters into teams equally.
* But if fighters can not distributed equally (n%k != 0), then we have 2 type of teams:
  + Basic team, where team member vb = n/k. There are n – n%k basic team.
  + Advanced team, where team member va = n/k + 1. There are n%k advanced team.

Chart

Description automatically generated with low confidence

* Consider the above case, where
  + Amount of basic team *a* = 2
  + Amount of advanced team *b* = 2
  + Member in basic team *vb* = 2
  + Member in advanced team *va* = 3
* We create a ***f(a, b, va, vb)*** function, which calculate the matches between fighters in different teams
* We can see that the amount of matches is the sum of all connection between different teams
* So we can calculate the ***f*** function according to the following **recursion** formular
* ***f(a, b, va, vb) =*** 
  + if amount of basic team b > 0:
    - f = (b-1)\*(vb\*vb) + a\*(va\*vb) + ***f(a, b-1, va, vb)***
  + if amount of advanced team a > 0:
    - f = (a-1)\*(va\*va) + b\*(va\*vb) + ***f(a-1, b, va, vb)***