Дополнительно задание(вариант 17):
Организация ввода двумерного целочисленного массива с последующей сортировкой его столбцов по возрастанию сумм двух средних элементов.

Краткие теоретические сведенья:

Алгоритмы поиска:
Существует много различных алгоритмов для поиска в массивах: линейный поиск, индексно-линейный поиск, бинарный поиск.
У каждого из них есть плюсы и минусы, например простота реализации линейного поиска. Но из-за простоты он имеет большУю временнУю сложность.

Сортировка массивов:
Алгоритмы сортировок очень широко применяются в программировании.

Виды: пузырьком(простыми обменами), сортировка выбором, быстрая сортировка, сортировка вставками, сортировка Шелла.

Производительность алгоритмов представляется временем и памятью, которые они занимают.

Так, сортировка пузырьком показывает хорошие результаты по памяти и времени на маленьких объемах данных. Однако на больших, она начинает уступать сортировки выбором.

Вывод:
Наша задача как программистов - четко понимать различия между алгоритмами, и умение их применять в различных ситуациях с максимальной выгодой.

Код:

```c
//
//  main.c
//  AdditionalTask
//
//  Created by Vlados iOS on 2/3/19.
//  Copyright © 2019 Vladislav Shilov. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/* TASK:
 Organizing the input of a two-dimensional integer array,
 followed by sorting its columns in ascending order of sums of two middle elements
*/

/// Reads only integer numbers from keyboard stream
/// Returns: read integer number from keboard

int readElementFromKeyboard() {
    char inputElement[100];
    int element = -1;
    bool shouldShowMessage = false;
    do {
        if(shouldShowMessage == true) {
            printf("Try to input positive integer number:\n");
        }
        scanf("%s",inputElement);
        element = (int)atol(inputElement);
        shouldShowMessage = true;
    } while(element < 0);
    return element;
}

/// Allocates memory for an array
/// Params: rowSize - amount of lines in array
///         columnSize - amount of columns in array
/// Returns: New dynamic array

int **dynamicArrayAlloc(size_t rowSize, size_t columnSize)
{
    int **A = (int **)malloc(rowSize * sizeof(int *));
    for(int i = 0; i < rowSize; i++) {
        A[i] = (int *)malloc(columnSize * sizeof(int));
    }
    return A;
}

/// Fills input array with random numbers
/// Params: array - array which we send via refernce on memory
///         rowSize - amount of lines in array
///         columnSize - amount of columns in array

void populateArray(int **array, int numberOfRows, int numberOfColumns) {
    for (int i = 0; i < numberOfRows; i++) {
        for (int j = 0; j < numberOfColumns; j++) {
            printf("Input [%d][%d] element\n", i, j);
            array[i][j] = readElementFromKeyboard();
        }
    }
}
```

```c
/// Prints array on console in a convenient format
/// Params: array - array which we send via refernce on memory
///         rowSize - amount of lines in array
///         columnSize - amount of columns in array

void printArray(int *array[], int numberOfRows, int numberOfColumns) {
    for (int i = 0; i < numberOfRows; i++) {
        for (int j = 0; j < numberOfColumns; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }
}

/// Swaps two values
/// Params: a - first value
///         b - second value

void swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
}

/// Swaps two arrays columns
/// Params: col1 - number of first column to swap
///         col2 - number of second column to swap
void columnSwap(int **array, int col1, int col2, int numberOfRows, int numberOfColumns) {
    for(int i = 0; i < numberOfRows; i++) {
        for (int j = 0; j < numberOfColumns; j++) {
            if(j == col1) {
                int t = array[i][j];
                array[i][j] = array[i][col2];
                array[i][col2] = t;
            }
        }
    }
}
/// Sorts array by the sum of middle elements in every column
/// Params: middleElementsAmount - amount of elements in the middle of array
///         you want to add to compare them
void sortArrayByMiddleElementsSum(int *array[], int numberOfRows, int numberOfColumns, int
    middleElementsAmount) {
    const int startIndexForSum = numberOfRows - middleElementsAmount - 1;
    const int endIndexForSum = startIndexForSum + middleElementsAmount - 1;

    int *sums = (int*)malloc(numberOfColumns * sizeof(int));

    // because of some issues with memory allocation
    for (int i = 0; i < numberOfColumns; i++) {
        sums[i] = 0;
    }

    // calculate sums of middle elenets
    for (int i = 0; i < numberOfRows; i++) {
        for (int j = 0; j < numberOfColumns; j++) {
            if(i >= startIndexForSum && i <= endIndexForSum) {
                sums[j] += array[i][j];
            }
        }
    }

    printf("\nsum of middle: \n");
    for (int i = 0; i < numberOfColumns; i++) {
        printf("%d ", sums[i]);
    }
    printf("\n\n");
```

```
    // swap columns if needed
    for (int i = 0; i < numberOfColumns; i++) {
        for (int j = 0; j < numberOfColumns; j++) {
            if(i < j && sums[i] > sums[j]) {
                swap(sums[i], sums[j]);
                columnSwap(array, i, j, numberOfRows, numberOfColumns);
            }
        }
    }

    free(sums);
}
int main(int argc, const char * argv[]) {

    const int middleElementsAmount = 2;
    int numberOfRows, numberOfColumns;
    int **array;

    printf("Input the number of rows and columns for the new array:\n");
    numberOfRows = readElementFromKeyboard();
    numberOfColumns = readElementFromKeyboard();

    printf("Input an array:\n");
    array = dynamicArrayAlloc(numberOfRows, numberOfColumns);

    populateArray(array, numberOfRows, numberOfColumns);
    printf("\nYour array:\n");
    printArray(array, numberOfRows, numberOfColumns);

    sortArrayByMiddleElementsSum(array, numberOfRows, numberOfColumns, middleElementsAmount);
    printArray(array, numberOfRows, numberOfColumns);

    return 0;
}
```

Результат:

```
Input the number of rows and columns for the new array:
5
5
Input an array:
Input [0][0] element
1
Input [0][1] element
2
Input [0][2] element
3
```

...

```
Your array:
1 2 3 4 5
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
1 2 3 4 5

sum of middle:
10 8 6 4 2

5 4 3 2 1
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
5 4 3 2 1
```

```
Input the number of rows and columns for the new array:
5
5
Input an array:
Input [0][0] element
1
Input [0][1] element
2
Input [0][2] element
3
Input [0][3] element
```

...

```
Your array:
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

sum of middle:
2 4 6 8 10

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Также алгоритм работает для любого числа средних элементов, но для этого нужно зарефакторить старый неправильный код:

```
107     int k = numberOfRows - middleElementsAmount;
108     if(k == 0) { k = 1; }
109     const int startIndexForSum = (numberOfRows / 2)/(k);
110     const int endIndexForSum = startIndexForSum + middleElementsAmount - 1;
```

Пример для одного среднего элемента:

```
Your array:
1 2 3 4
4 3 2 1
1 2 4 3
1 2 1 2

sum of middle:
4 3 2 1

4 3 2 1
1 2 3 4
3 4 2 1
2 1 2 1
Your array:
1 2 3
1 3 2
1 1 1

sum of middle:
1 3 2

1 3 2
1 2 3
1 1 1
```

Пример для 3 элементов: