CS 137: Assignment #8

Due on Friday, Nov 21, 2025, at 11:59PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2025

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 11.
- `<math.h>` is not allowed
- Use Valgrind when testing.
- **For each question, you won't get any correctness marks if the solution doesn't meet the running time requirement.**
- **For all questions: All marmoset tests will only verify the correctness of your functions; however, they do not assess whether the solution compiles within the required running time. We will assess the runtime of your submission with the highest correctness score after the deadline (if multiple submissions have the same score, the most recent one will be assessed)**

Problem 1

For this question, you will operate on a sequence of natural numbers. For the remainder of this question, we will refer to this operation as *multi-add*.

multi-add involves alternating back and forth between two steps:

1) Multiply pairs of integers to reduce the number of integers to half. You should multiply the smallest integer by the second smallest, the third smallest by the fourth smallest, and so on.
2) Add pairs of integers to reduce the number of integers to half. You should add the smallest integer to the second smallest, the third smallest to the fourth smallest, and so on.

For example, consider the following integers:

`1 4 3 6 5 3 2 8`

Multiply step reduces the integers to: `2 9 20 48` Because (1*2 3*3 4*5 6*8)

Add step reduces the integers to: `11 68` Because (2+9 20+48)

Multiply step reduces the integers to: `748`

Complete the file `multiadd.c` that contains the definition of

```
long long int multiadd(long long int a[], int n);
```

that takes an array of length `n` of natural numbers. `n` is at least `2` and is a power of `2` (meaning that n could be `2,4,8,16,32,64`, etc.).

You must also **assume** that all values in the array are less or equal to `n` (***This is an important hint***). The Function returns the result of applying *multi-add* operation on the values in the array `a`.

Your Function must run **O(n)** run time where `n` is the array's length.

Note: You must **NOT** use VLA; however, you may allocate arrays on the heap. (Do not forget to free them before the end of the Function.

Submit `multiadd.c` file containing only your implemented Function (that is, you must delete the test cases portion/the `main` Function). However, **you must keep the required included libraries.**

Sample main function is provided in `multiadd.c` file

Problem 2

Complete the file `countcommon.c` that includes the definition of

```
    int count_common(long long int a[], long long int n1, long long
int b[], long long int n2);
```

that takes two arrays (`a` and `b`) of natural numbers sorted in nondecreasing order and the length of the two arrays (`n1` and `n2`, respectively).

The Function returns the number of common elements in both arrays. If a common value appears more than once in at least one of the arrays, it must be counted only once.

Your Function must run in ***O(n)*** run time where `n` is the maximum between `n1` and `n2`.

Submit `countcommon.c` file containing only your implemented Function (that is, you must delete the test cases portion/the `main` Function). However, **you must keep the required included libraries.**

You must not create any additional arrays.

Sample main function is provided in `countcommon.c` file

Problem 3

Given an array of length n of integers(positives, negatives, and zeros), we want to find the largest sum that can be calculated from a sequence of consecutive elements in the array. If the array has only negative values, the answer is 0.

For example, given the array [-2,-4,3,-1,5,6,-7,-2,4,-3,2], the answer is 13 (3+-1+5+6).

Write a function maxsequence(int a[], int n) that takes an array of integers of length n and returns the largest sum as defined above. Your Function must run in ***O(n)*** run time, where `n` is the array's length.

Note: You can not use VLA (or define an additional array in the function with length n)

Submit `maxseq.c` file containing only your implemented Function (that is, you must **delete** the test cases portion/the `main` Function). However, **you must keep the required included libraries.**

Here is a sample main function for testing:

```
1. int main()
2. {
3.     int a[] = {-2,-4,3,-1,5,6,-7,-2,4,-3,2};
4.     int n = sizeof(a)/sizeof(int);
5.     assert(maxsequence(a, n) == 13);
6.
7.     int b[] = {-2,-4,-3,-1};
8.     n = sizeof(b)/sizeof(int);
9.     assert(maxsequence(b, n) == 0);
10.
11.    int c[] = {1,2,3,4,5,6};
12.    n = sizeof(c)/sizeof(int);
13.    assert(maxsequence(c, n) == 21);
14.
15.    return 0;
16. }
17.
```

Problem 4

Create a C program `minTasks.c` that consists of a function:

```
int findTasks(int tasks[], int n, int numempl);
```

Given the number of minutes in `n` different tasks in array `tasks` and `numempl` employees. The tasks are arranged in non-descending order of the number of minutes required to complete each task. Every employee is assigned to complete some consecutive tasks. The goal is to assign tasks in such a way that the maximum number of minutes assigned to an employee is minimum (the function returns that value).

Example :

`tasks[] = {120, 340, 670, 900}  numempl = 2`

returned value : `1130`

Explanation:

There are `two` employees. Tasks can be distributed in `3` different ways:

1) `[120]` and `[340, 670, 900]`

   Max number of minutes is assigned to employee `2` with `340 + 670 + 900 = 1910` minutes

2) `[120, 340]` and `[670, 900]`

   Max number of minutes is assigned to employee `2` with `670 + 900 = 1570` minutes

3) `[120, 340, 670]` and `[900]`

Max number of minutes is assigned to employee `1` with `120 + 340 + 670 = 1130` minutes

Assumptions: the maximum total number of minutes is always `<=10000`. `n>=numempl`, `n>0`, `numempl>0`.

The program must run in **O(nlogm)** where n is the number of tasks and m is the sum of all task durations. Since m is limited to a maximum of 10000, **O(nlogm)** is **O(n)**.

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion/main function). However, **you must keep the required included libraries.**

Hint: The tasks are arranged in ascending order of the number of minutes. Take advantage of this fact 😊

Sample main function is provided in `minTasks.c` file