CS 137: Assignment #8

Due on Friday, Nov 22, 2024, at 11:59PM

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

Victoria Sakhnini

Fall 2024

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`

Create a 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.**

Here is a sample test file:

```
 1. #include…..

 2. long long int multiadd(long long int a[], int n);

 3. int main(void){
 4.     long long int a[8] = {1,4,3,6,5,3,2,8};
 5.     assert(748 == multiadd(a,8));
 6.     long long int b[16] = {0};
 7.     assert(0 == multiadd(b,16));
 8.     long long int c[16] = {1,6,5,8,6,7,5,4,8,9,10,2,12,14,1,1};
 9.     assert(27498 == multiadd(c,16));
10.     return 0;
11. }
12.
```

Problem 2

Create a 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, **<u>you must keep the required included libraries.</u>**

You must not create any additional arrays.

Here is a sample test file:

```
 1. #include ….
 2. int count_common(long long int a[], long long int n1, long long int b[], long
long int n2);
 3.
 4. int main(void){
 5.     long long int a1[4] = {1,1,1,1};
 6.     long long int b1[8] = {1,1,1,1,1,1,1,1};
 7.     assert(1 == count_common(a1,4,b1,8));
 8.     long long int a2[8] = {1,3,3,4,5,8,9,100};
 9.     long long int b2[7] = {4,4,8,15,20,40,100};
10.     assert(3 == count_common(a2,8,b2,7));
11.     return 0;
12. }
13.
```

Problem 3

You are given a string of lowercase characters representing a word. Your task is to complete a C program, `palindrome.c`, that reads the input string (only lower case letters between a to z) and determines whether it is possible to rearrange the characters to form a palindrome. A palindrome is a word that reads the same backward as forward.

Your Function `void canFormPalindrome(char *str)` takes a string and should output "YES" if it is possible to form a palindrome by rearranging the characters of the string; otherwise, it should output "NO". in Both cases, the output is followed by "\n".

For example, the string "aabb" can be rearranged to form the palindrome "abba," so the output would be "YES."

Complete the C program to solve this problem, and assume that the input string has at most 1000 characters (so you can define a fixed array). Your Function must run in ***O(n)*** run time, where `n` is the length of the string.

Submit `palindrome.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.**

Note: Assume all string functions from <string.h> we covered in lectures, each has O(n) running time.

Here is a sample main function for testing:

```c
#include …

// Function to check if it is possible to rearrange the characters to form a palindrome
void canFormPalindrome(char *str) {    }

int main() {
    char inputString[1001];  // you can make it bigger if you want to test longer strings
    // Read the input string
    printf("Enter a string of lowercase characters: ");
    scanf("%1000s", inputString);
    canFormPalindrome(inputString);
    return 0;
}19.
```

Examples of input and output:

1)
Enter a string of lowercase characters: abcdabcdabcd
NO

2)
Enter a string of lowercase characters: cs
NO

3)
Enter a string of lowercase characters: aabb
YES

4)
Enter a string of lowercase characters: reprepa
YES