Quiz/Homework 3

The questions are designed by the TAs or me. For information on how to contact the TAs, please refer to the course outline.

Due Date: **From** Monday March **18th** to March **29th** during your labs sessions.

Instructions

- Please ensure that you complete all questions and present your answers to your TAs during **one** of your **assigned** weekly lab sessions between Due Date. TAs may ask you questions and request you to work with your codes or present it. TAs will provide feedback and your grade right away. Grades will also be released on Avenue a few days later.
- The purpose of these Quiz/Homework assignments is not solely to test your programming skills. Instead, they are designed to provide you with simple and basic examples to better prepare you for your assignments. Consequently, you can work on the questions outside of the lab sessions. You may ask questions from your TAs at any point over Teams or during the lab sessions.
- Please attend only the labs for which you are enrolled to prevent classes from becoming overcrowded.
- You don't need to submit anything. Just keep your files organized to be presentable if the TAs asked you to work with your code.

1. **Reverse an Array using Pointers (Renjie, 1 point)**

   Given an integer array `arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9}` of size n, write a function `reverseArray(int *arr, int n)` that reverses the array using pointers. **Do not** use indexing to access the array elements within this function.

2. **Toggle Case in a String using Pointers (Renjie, 0.5 points)**

   Given a string `s[] = "Hello, World!"`, write a function `toggleCase(char *s)` that changes each uppercase letter to lowercase and each lowercase letter to uppercase. The function should directly modify the string passed to it using pointer arithmetic.

3. **Mini Math Module using Pointers and Make (Patrick, 1 point)**

   Create a module called `pmath.h` comprised of three functions for adding, multiplying, and taking the absolute value of a number.

- `void add(int* num1, int* num2, int* res)`
- `void multiply(int* num1, int* num2, int* res)`
- `void absolute(int* num)`

Have these functions called in `main.c` and create a `Makefile` that compiles the `pmath.c` and `main.c` files and produces an output file. All of the object files should be automatically stored in a build folder. The `make clean` command should remove the build folder and the output file. Below is the main function you can use to support your answer.

```c
#include "pmath.h"

int main(){

    int a = 2;
    int b = 5;
    int c = 0;
    int d = -10;

    add(&a, &b, &c);
    printf("The value of c after adding is %d\n", c);

    multiply(&a, &b, &c);
    printf("The value of c after multiplying is %d\n", c);

    printf("The value of  d before taking the absolute value %d
       \n", d);
    absolute(&d);
    printf("The value of d after taking the absolute value %d\n
       ", d);

    return 0;
}
```

4. **Matrix Transpose (Pedram, 1 point)**

Create a program to receive two inputs `n` and `m`, using Command-line Arguments, like :

```c
 int main(int argc, char *argv[])
```

Create a `matrix` with `n` number of rows and `m` number of columns all with random double precision numbers between `-10` to `+10`. Expand the code to compute the Transpose of `matrix`. You can keep the codes for this question in `q4.c`.

5. **Sorting Algorithms Efficiency (Pedram, 1 point)**

   The following program sorts an array of integers using the bubble sort algorithm:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void bubbleSort(int arr[], int n) {
 int i, j, temp;
 for (i = 0; i < n - 1; i++) {
  for (j = 0; j < n - i - 1; j++) {
   if (arr[j] > arr[j + 1]) {
    // Swap arr[j] and arr[j+1]
    temp = arr[j];
    arr[j] = arr[j + 1];
    arr[j + 1] = temp;
   }
  }
 }
}

int main() {
 const int ARRAY_SIZE = 200000;
 int arr[ARRAY_SIZE];

 // Seed the random number generator with current time
 srand(time(NULL));

 // Fill the array with random integers
 for (int i = 0; i < ARRAY_SIZE; i++) {
  arr[i] = rand();
 }

 // Display some of the elements (optional)
 printf("First 10 elements of the array:\n");
 for (int i = 0; i < 10; i++) {
  printf("%d ", arr[i]);
 }
 printf("\n");
```

```
  clock_t start = clock();
  // Sort the array
  bubbleSort(arr, ARRAY_SIZE);
  clock_t end = clock();

  double taken_time = (double)(end-start)/CLOCKS_PER_SEC;
  printf("Time taken by bubbleSort: %f seconds\n", taken_time);

  // Display some of the sorted elements (optional)
  printf("First 10 elements of the sorted array:\n");
  for (int i = 0; i < 10; i++) {
   printf("%d ", arr[i]);
  }
  printf("\n");
}
```

(a) In my machine, it takes about 115 seconds to sort an array with the size `200000`, in ascending order. Run the code and write down how long it takes in your machine.

(b) Change the implementation of function `bubbleSort(int arr[], int n)` to use pointers. You don't need to change anything in `int main()`, but the function input arguments, `bubbleSort(int *arr, int n)`, and codes within the function must be changed to user pointer. Now run the program, and compare the time taken. Which one is faster?

(c) In C, there is a standard library function called `qsort` for sorting arrays. It's part of the `stdlib.h` header file. The qsort function is a standard library function that implements a highly optimized sorting algorithm. Write the code using `qsort()` function and compare the speed-up with the previous ones.

Keep the time taken by:

1. Bubble Sort without pointers,
2. Bubble Sort with pointers, and
3. `qsort()`,

and show them to your TA. You can keep all your codes for this question in `q5.c`.

6. **Debugging with GDB or LLDB (Pedram, 1 point)**

The following function is another algorithm sorting an array:

```
void merge(int arr[], int left, int mid, int right) {
 int n1 = mid - left + 1;
 int n2 = right - mid;

 // Create temporary arrays
 int L[n1], R[n2];

 // Copy data to temporary arrays L[] and R[]
 for (int i = 0; i < n1; i++)
 L[i] = arr[left + i];
 for (int j = 0; j < n2; j++)
 R[j] = arr[mid + 1 + j];

 // Merge the temporary arrays back into arr[left..right]
 int i = 0, j = 0, k = left;
 while (i < n1 && j < n2) {
  if (L[i] <= R[j]) {
   arr[k] = L[i];
   i++;
  } else {
   arr[k] = R[j];
   j++;
  }
  k++;
 }

 // Copy the remaining elements of L[], if any
 while (i < n1) {
  arr[k] = L[i];
  i++;
  k++;
 }

 // Copy the remaining elements of R[], if any
 while (j < n2) {
  arr[k] = R[j];
  j++;
  k++;
 }
}

// Merge Sort function
void mergeSort(int arr[], int left, int right) {
 if (left < right) {
  int mid = left + (right - left) / 2;

  // Sort first and second halves
  mergeSort(arr, left, mid);
  mergeSort(arr, mid + 1, right);

  // Merge the sorted halves
  merge(arr, left, mid, right);
 }
}
```

And it can be called by:

```
clock_t start = clock();
mergeSort(arr, 0, ARRAY_SIZE - 1);
clock_t end = clock();
```

Run the code with Merge algorithm and compare the time with previous ones. (a) **Which** one runs faster, Bubble or Merge?

Use `mergeSort()` function and increase the array size from `200000` to `1500000`. You should see `Segmentation fault (core dumped)` error in the terminal. Using GDB\LLDB, setup some break points and find (b) **which** line of the code is causing this issue. (c) **Why** is this happening? Note: the same array size using `bubbleSort()` will not create this issue. You can keep the codes for this question in `q6.c`.