Homework - 4

# Problem 1 (5 points)

Write a C program that sorts the ***real numbers*** provided through user inputs (***not from the command line***). Your program can assume that the count of the real numbers is not greater than 100. The real numbers can be processed as 4-byte float point numbers. Note that the numbers may be a mixture of positive numbers and negative numbers. **Your program must use ONLY radix sort. It will NOT receive any points if it uses other sorting algorithms.**

# Instructions

The program should ask the user for a count before it reads into the numbers. For saving the numbers and for the buckets, your program can use arrays, and the size of each array or bucket is 100. Then, in a loop, the program asks the user for the numbers (*scanf("%d", …*) or *scanf("%d", …)*), and save the numbers in an array or one piece of the memory space.

The program prints out the sorted numbers *with one number on each line*.

Applying bitwise operations directly to floating point numbers is not allowed in C. To apply bitwise operations on floating point numbers, your program needs to “instruct” the system to interpret the data as unsigned integers, which have the same length as floating point numbers. You can do this with typecasting using a pointer. This is also illustrated using the following example.

float f=123.45;

unsigned int \*p = (unsigned int \*) &f, value\_of\_bits4to7 = (\*p & 0xF0)>>4;

Analyzing the following program and its output will also help you understand how to radix-sort float point values.

#include <stdlib.h>

<stdio.h>

int main(){

int i;

float value, f[20];

/\* typecasting using a pointer \*/ unsigned int \*p = (unsigned int \*) f;

value = -10.5;

for( i = 0; i < 20; i++) { f[i] = value;

value = value + 1;

} /\* f has 20 float point numbers in ascending order \*/

for( i = 0; i < 20; i++)

printf("%.2f\t%u\n", f[i], p[i], (p[i]&0xFFFF0000)>>16);

/\* show the float point numbers (column 1) and the corresponding

* unsigned int values (column 2). Bit-wise operations are used to
* obtain the last column, showing that bit-wise operations can be
* used after typecasting float point numbers to unsigned integers.

\*/

}

Check the output of the program. The first column shows float point numbers in ascending order. Check column 2, which shows that sorting the corresponding unsigned int values in ascending order is not enough to ensure that the float point values are also in ascending order. By checking column 2, you can understand better the two methods for sorting float point numbers in the slides.

# Testing

Test your programs manually first. Manually type in inputs, and check output.

To fully test your programs with more numbers, modify and use the following scripts. $1 of the script is the count.

Take screenshot when you test your programs manually, such that we can see the numbers provided to the programs and the output of the programs.

Bash script for testing the program radix-sorting float point numbers:

#!/bin/bash count=$1 rm input

rm your\_output

rm standard\_output

echo "================== input ======================"

echo ${count} | tee input

for (( i=0; i<${count}; i++ )); do

printf "%d.%d\n" $((($RANDOM-$RANDOM)%1000)) $RANDOM

done | tee -a input

echo "============= execution result ================"

cat input | YOUR\_PROGRAM | xargs printf "%.2f\n" | tee your\_output tail -n +2 input | sort -n | xargs printf "%.2f\n" > standard\_output echo "====== differences from correct result ======="

diff your\_output standard\_output

# Problem 2 (5 points)

Your task is to write a C program that utilizes radix sort to sort a list of 4-byte integers. The integers can range from negative to positive values.

The program should not retrieve integer values from the command line. Instead, it should read in a count of integers to be entered/sorted; after this, based on the count, it should read in the corresponding number of integers.

Assume that the number of integers to be sorted will not exceed 100. Therefore, both your primary array (the list of integers to be sorted) and your buffer array (the buckets used in radix sort) should be initialized with a size of 100.

The radix sort algorithm implemented should be a **hexadecimal radix sort**. This means the program should process 4 bits at a time and use 16 buckets for sorting. **Submissions that do not adhere to this specification will not receive credit.**

Once sorted, the program should print each integer in the sorted array on a separate line. The output should contain no text other than these integers. The following Bash script will be used to test your program. The script assumes that your output only includes the sorted integers, with no additional text. Make sure to align your program's output with this assumption to ensure accurate testing.

**Testing**:

Begin by manually testing your program. Input numbers directly and verify the output. For more rigorous testing with larger amounts of data, utilize the following scripts. The script variable $1 will serve as the count. Since all the output of your program will be used in the comparison, you want to comment out the printf() calls that are not printing out the sorted integers (such as hints, error messages, debug messages, etc) before you test your program using the script.

Remember to capture screenshots while manually testing your program. These screenshots should show the numbers being inputted into your program and the resultant output. This will help in verifying the functionality of your program.

#!/bin/bash

count=$1

rm input

rm your\_output

rm standard\_output

echo "================== input ======================"

echo ${count} | tee input

for (( i=0; i<${count}; i++ )); do

echo $((RANDOM-16384))

done | tee -a input

echo "============= execution result ================"

cat input | PATH\_NAME\_OF\_YOUR\_PROGRAM | tee your\_output

tail -n +2 input | sort -n > standard\_output

echo "====== differences from correct result ======="

diff your\_output standard\_output