Homework 6: CSE 3318

About the Assignment

In this assignment, you will complete the program provided to implement Bucket Sort. Starter code is provided, along with unit tests to ensure correct implementation. You must use the bucket sort algorithm as described in class to earn credit. The goal is to write code that reads numbers from a file as an array and sorts the array using Bucket Sort.

Provided Files

1. `list.c` and `list.h`: These provide the Linked List structure as well as utility functions.
2. `makefile`: Provides the build scripts.
    a. To compile, simply use the command: `make .`
        i. To force recompilation of all files use: `make -B`
    b. Run with `./unit_tests`
    c.
3. `main.c`: Contains unit tests to test each utility function (e.g. reading array from file or finding the min in an array). The final test functions test your code on each of the data files. E.g. `int test_sorting_data1()` will test that your code correctly opens the file, reads the data and sorts the array given in file `data1.txt`.
4. `bucket_sort.h`: Contains signatures for the functions you will implement.
5. `bucket_sort.c`: Code and helper functions implementing Bucket Sort. ONLY MODIFY THIS FILE. This is the ONLY .c file you will submit.
6. `data.txt`, `data1_rev.txt`, `data2.txt`, `neg1.txt`, `dex1.txt`: Test files to read numbers from and sort. These have Unix files (they have the Unix EOL symbols).

    File format:
    The first line of each input file contains a single integer *N*, denoting the size of the array.
    The second line of each input file contains *N* space-separated integers corresponding to the *N* elements of the array.

    Example:
    6
    9 1 7 5 8 2
    means the size of the array will be 6 and the array will be {9,1,7,5,8,2}.

Note: All your work will be done in `bucket_sort.c`.

Step – 1: Open the file.

In bucket_sort.c, complete the function `FILE *openFile(char *filename)`.

It takes the name of the file (a string called `fileName`) as input and returns the file handle of the opened file. Do not close the file. If the file does not open, the function must print the string `"Could not open file. Exiting.\n"`.

Step – 2: Read data from the file.

Complete the function called `int readSizeFromFile(FILE *fp)` that simply reads an integer from the file and returns it.

Then, complete the function called `int *readArrayFromFile(FILE *fp, int N)` that dynamically allocates an integer array of size `N`, fills it up by reading `N` integers from the file with the given file handle `fp`, and finally returns the array.

At this point, if done correctly, the first set of tests to open files should pass.

Step – 3: Compute minimum and maximum values from the array.

Now, implement functions `int min_element(int *arr, int N)` and `int max_element(int *arr, int N)` which return the minimum and maximum values respectively in the passed nonempty array `arr` of size `N`. For example for array {9,1,7,5,8,2} max is 9 and min is 1.

If correctly done, the next set of tests related to min and max values should pass.

Step – 4: Compute bucket indexes.

Complete the function called `int bucket_index(int element, int min, int max, int N)` that computes and returns the index of an element's bucket. This should be done using the way discussed in class with the formula that uses the min and max values in the array.

E.g. `bucket_index(7, 1, 9, 6)` returns ***4*** Here element = ***7*** , min = 1 , max = 9, N = 6 then the bucket index is computed using: round_down(((7-1)/(9-1+1))*6)

More examples:

`bucket_index(9, 1, 9, 6)` returns ***5***
`bucket_index(1, 1, 9, 6)` returns ***0***
`bucket_index(7, 1, 9, 6)` returns ***4***
`bucket_index(5, 1, 9, 6)` returns ***2***
`bucket_index(8, 1, 9, 6)` returns ***4***
`bucket_index(2, 1, 9, 6)` returns ***0***

At this point, the next set of tests related to computing bucket indexes should pass.

Step – 5: Insert elements into the right buckets.

Now, implement the function `nodePT *putIntoBuckets(int *arr, int N)` that uses our auxiliary functions defined so far, dynamically allocates an array of linked lists and inserts each element of our original array into its corresponding linked list bucket.

E.g. for {9,1,7,5,8,2}, we have: min=1, max = 9, N=6 and generate bucket indexes:

9 => bucket index : 5
1 => bucket index : 0
7 => bucket index : 4
5 => bucket index : 2
8 => bucket index : 4
2 => bucket index : 0

At this point, the next set of tests labelled buckets should pass.

Step – 6: Putting it all together.

Finally, implement `int *run_bucket_sort(char *fileName)` that uses our functions to read an array from a file, sort it and return it. It must free all the buckets and other heap allocations it makes at the end (except the array it returns).

Now, all tests should pass.

Specifications

1. Files will always have the above format.
2. The filename (entered by the user) will have at most 50 characters.
3. Let N be the number of elements in an array in a file. Then $1 \leq N \leq 10000$.
4. There is no limit for the length of the second line in the file (the line that contains the elements in the array).
5. The numbers from the file must be stored in an array of integers.
6. Do not make a copy array. The only arrays in your program should be: the array of int in which you load the data from the file, and the array of pointers to linked lists. There should not be any other array. (The nodes from the linked lists are fine. They are not considered a copy array.) - 10 points deducted otherwise.
7. The linked lists for the buckets must be maintained sorted: when a new node is inserted in a sorted list, it will be inserted in the correct location so that the list will be sorted after the insertion. Review our linked lists lectures. We solved this in class.
8. There are no errors in Valgrind.

Resources and Hints

1. See resources:
    1. See video in Canvas->Homework 6 for a brief explanation for using an array of linked lists and inserting in a sorted linked list.
    2. Review the lecture on bucket sort as we discussed details relevant for this specific homework (e.g. index calculation, inserting in a sorted linked list and using an array of linked lists, issues with overflow).
2. If your indexes are not correct, the main causes are:
    1. integer division in C. E.g. 6/9*6 = (6/9)*6 = 0*6 = 0 (b.c. 6/9 = 0).
    2. overflow. File dex1.txt covers a special case: it includes the minimum and maximum values possible for a 4 byte int, which may cause an overflow in the index calculation.
    3. Solution: when you perform, or store intermediate calculations, you can use any data type you want; it does not have to be the type of the input or the output. For example, the input are ints, but when you compute the index, you can cast them to double and/or use other values of type double. That should handle both issues 2.1 and 2.2 from above. Note that you should use casting and/or a few local variables of type double, not an entire array of double.
3. For a review of my linked lists implementation see slides Linked Lists in C REVIEW and the four Linked Lists videos (under Canvas-> Modules->Linked Lists). More resources on linked lists are posted in Canvas->Modules->Linked Lists->"Linked Lists page", including references to solved problems.
4. You can use this visualization for how bucket sort works, but in your solution do not use their formula for index calculation, use our formula with min and max.

Grading Rubric

8 pts - style
10 pts - all 23 tests passed. No partial credit. If a single test fails, all 10 points are lost.
9 pts = 3 pts * 3 tests for the test_open_XXXX tests (3 pts * 3 tests => 9pts)
6 pts = 3 pts * 2 tests for test_min, test_max (3 pts each)
15 pts = 3 pts * 5 tests for the test_min_max_XXXX tests
20 pts = 4 pts * 5 tests for the test_bucket_index_XXXX tests
12 pts = 4 pts * 3 tests for the test_buckets_XXXX tests
20 pts = 4 pts * 5 tests for the test_sorting_XXXX tests
See grading example at the end of the document.
The code will be inspected. The functions must use the method covered in class to get the credit.
If a test passed but it was hardcoded somehow or it used another method, that test receives 0 credit.

Final Submission

Submit only the bucket_sort.c file. Do not archive it in any way. It must contain your name and student ID at the top as a comment.

Sample runs

Sample run when all tests passed

Run command:

```
make -B
```

Run command:

```
./unit_tests
```

See output:

```
RUNNING TEST: test_open_data1
============================================================
RUNNING TEST: test_open_data2
============================================================
RUNNING TEST: test_open_dex1
============================================================
RUNNING TEST: test_min
============================================================
RUNNING TEST: test_max
============================================================
RUNNING TEST: test_min_max_data1
============================================================
RUNNING TEST: test_min_max_data1_rev
============================================================
RUNNING TEST: test_min_max_dex1
============================================================
RUNNING TEST: test_min_max_data2
============================================================
RUNNING TEST: test_min_max_neg1
============================================================
RUNNING TEST: test_bucket_index_data1
============================================================
RUNNING TEST: test_bucket_index_data1_rev
============================================================
RUNNING TEST: test_bucket_index_data2
============================================================
RUNNING TEST: test_bucket_index_dex1
============================================================
RUNNING TEST: test_bucket_index_neg1
============================================================
RUNNING TEST: test_buckets_data1
============================================================
RUNNING TEST: test_buckets_data1_rev
============================================================
RUNNING TEST: test_buckets_neg1
```

```
============================================================
RUNNING TEST: test_sorting_data1
============================================================
RUNNING TEST: test_sorting_data1_rev
============================================================
RUNNING TEST: test_sorting_dex1
============================================================
RUNNING TEST: test_sorting_neg1
============================================================
RUNNING TEST: test_sorting_data2
============================================================
.......................
OK: 23
```

Sample run when all tests failed. This is what you get for the starter code

Run command:

```
make -B
```

Run command:

```
./unit_tests
```

See output:

```
RUNNING TEST: test_open_data1
ERROR: main.c:24 assertion failed: "fp != NULL"
File failed to open.
============================================================
RUNNING TEST: test_open_data2
ERROR: main.c:42 assertion failed: "fp != NULL"
xFile failed to open.
============================================================
RUNNING TEST: test_open_dex1
ERROR: main.c:60 assertion failed: "fp != NULL"
xFile failed to open.
============================================================
RUNNING TEST: test_min
ERROR: main.c:79 expected -123, actual: min_computed
============================================================
RUNNING TEST: test_max
ERROR: main.c:90 expected 100, actual: max_computed
============================================================
RUNNING TEST: test_min_max_data1
ERROR: main.c:100 assertion failed: "fp != ((void *)0)"
============================================================
RUNNING TEST: test_min_max_data1_rev
ERROR: main.c:117 assertion failed: "fp != ((void *)0)"
============================================================
RUNNING TEST: test_min_max_dex1
ERROR: main.c:134 assertion failed: "fp != ((void *)0)"
============================================================
RUNNING TEST: test_min_max_data2
```

```
ERROR: main.c:151 assertion failed: "fp != ((void *)0)"
============================================================
RUNNING TEST: test_min_max_neg1
ERROR: main.c:168 assertion failed: "fp != ((void *)0)"
============================================================
RUNNING TEST: test_bucket_index_data1
ERROR: main.c:185 assertion failed: "fp"
============================================================
RUNNING TEST: test_bucket_index_data1_rev
ERROR: main.c:209 assertion failed: "fp"
============================================================
RUNNING TEST: test_bucket_index_data2
ERROR: main.c:233 assertion failed: "fp"
============================================================
RUNNING TEST: test_bucket_index_dex1
ERROR: main.c:257 assertion failed: "fp"
============================================================
RUNNING TEST: test_bucket_index_neg1
ERROR: main.c:281 assertion failed: "fp"
============================================================
RUNNING TEST: test_buckets_data1
ERROR: main.c:325 assertion failed: "fp"
============================================================
RUNNING TEST: test_buckets_data1_rev
ERROR: main.c:360 assertion failed: "fp"
============================================================
RUNNING TEST: test_buckets_neg1
ERROR: main.c:395 assertion failed: "fp"
============================================================
RUNNING TEST: test_sorting_data1
ERROR: main.c:444 assertion failed: "fp"
============================================================
RUNNING TEST: test_sorting_data1_rev
ERROR: main.c:465 assertion failed: "fp"
============================================================
RUNNING TEST: test_sorting_dex1
ERROR: main.c:486 assertion failed: "fp"
============================================================
RUNNING TEST: test_sorting_neg1
ERROR: main.c:508 assertion failed: "fp"
============================================================
RUNNING TEST: test_sorting_data2
ERROR: main.c:529 assertion failed: "fp"
============================================================
xxxxxxxxxxxxxxxxxxxxx
OK: 0 FAILED: 23
```

Example Grading

For example, the code below would lose 27 pts from the 92 pts for code correctness:
- 10 pts (because one or more tests failed)
- 17 pts from the 5 failed tests highlighted below

```
./unit_tests
RUNNING TEST: test_open_data1
============================================================
RUNNING TEST: test_open_data2
============================================================
RUNNING TEST: test_open_dex1
============================================================
RUNNING TEST: test_min
ERROR: main.c:79 expected -123, actual: min_computed       --------> 3 pts lost
============================================================
RUNNING TEST: test_max
============================================================
RUNNING TEST: test_min_max_data1
============================================================
RUNNING TEST: test_min_max_data1_rev
============================================================
RUNNING TEST: test_min_max_dex1
ERROR: main.c:142 expected -2147483648, actual: min_computed---------> 3 pts lost
============================================================
RUNNING TEST: test_min_max_data2
============================================================
RUNNING TEST: test_min_max_neg1
ERROR: main.c:176 expected -10, actual: min_computed       ---------> 3 pts lost
============================================================
RUNNING TEST: test_bucket_index_data1
============================================================
RUNNING TEST: test_bucket_index_data1_rev
============================================================
RUNNING TEST: test_bucket_index_data2
============================================================
RUNNING TEST: test_bucket_index_dex1
ERROR: main.c:272 expected testArr[i], actual: index_computed[i]
...x...x.x...Bucket index computed incorrectly.            ---------> 4 pts lost
============================================================
RUNNING TEST: test_bucket_index_neg1
============================================================
RUNNING TEST: test_buckets_data1
============================================================
```

```
RUNNING TEST: test_buckets_data1_rev
============================================================
RUNNING TEST: test_buckets_neg1
============================================================
RUNNING TEST: test_sorting_data1
============================================================
RUNNING TEST: test_sorting_data1_rev
============================================================
RUNNING TEST: test_sorting_dex1
ERROR: main.c:498 expected arr[i], actual: test_sorted[i]   ---------> 4 pts lost
============================================================
RUNNING TEST: test_sorting_neg1
============================================================
RUNNING TEST: test_sorting_data2
============================================================
x.....x..
OK: 18 FAILED: 5
```