ASSIGNMENT 2: SORTING

CS3D5A, Trinity College Dublin

Deadline: 13:00 03/11/2022
Grading: The assignment will be graded on Submitty based on the results of the tests, your code and your report.
Questions: You will be able to ask questions during the lab hours on 21/10/2022
Submission: Submit via Submitty, see instructions for each task marked with ☁

Goals:

- Implement some simple sorting algorithms
- Learn how to implement quicksort
- Learn how to evaluate the performance of a sorting algorithm
- Use sorting in a practical application
- Learn to use header files in c

Task 1 – Set up (2 marks)

In this assignment, we want to evaluate sorting algorithms on different types of arrays. In this first task, you will write functions to generate these arrays. Edit the `t1_skeleton.c` file to generate arrays of the following types:

- An ascending sorted array e.g. [0, 1, 2, 3, 4, 5]
- A descending sorted array [5, 4, 3, 2, 1, 0]
- An array where every value is the same (uniform) e.g. [3, 3, 3, 3, 3, 3]
- A randomly shuffled array with no duplicate values e.g. [4, 3, 5, 1, 0, 2]
- A randomly shuffled array with duplicate values e.g. [3, 3, 2, 1, 1, 4]

Only edit the function were indicated (do not change their signature), but you can add functions, variables etc.

You can use the `t1_test_skeleton.c` file to your implementation locally. Note that the functions are defined in `t1_skeleton.c` but used in `t1_test_skeleton.c`, thanks to the header file `t1.h` which contains the signatures of the functions implemented in t1_skeleton.c. Note how `t1_test_skeleton.c` includes the line #include "t1.h". To compile this, you can simply write:

```
gcc -Wall t1_skeleton.c  t1_test_skeleton.c -o t1
```

and then run the executable t1.

Submit the edited `t1_skeleton.c` and (un-edited) `t1.h` on Submitty for task 1 (do NOT submit the `t1_test_skeleton.c`)

Task 1 - mark allocations	
Write a program to generate arrays of n values for each of the 5 types of data given above	2 marks

Task 2- Sorting algorithms (5 marks)

In this task, you will implement some sorting algorithms. Edit the t2_skeleton.c file to implement the following sorting algorithms:

- Insertion sort

- Selection sort

- Quicksort – you can choose any pivot selection and partitioning, mention and justify the design choices made (pivot selection and partitioning system chosen) in your report

Test your algorithms first on small arrays, then extend to bigger arrays (you can use task 1 to generate arrays). You can use `t2_test_skeleton.c` for this.

Task 2 - mark allocations	
Correct implementation of insertion sort.	1 mark
Correct implementation of selection sort.	1 mark
Correct implementation of quicksort.	3 marks

Submit the edited `t2_skeleton.c`, and the unedited `t1.h` and `t2.h` for "task 2 & 3" to Submitty. (do NOT submit the `t2_test_skeleton.c`)

Task 3 – Algorithm comparisons (4 marks)

Update your code for task 2 to count the number of swaps and counts for each of them (using the global variables `number_comparisons` and `number_swaps`). Run `t3_test.c` to profile your implementations of the sort functions. (eg `gcc -Wall t1_skeleton.c t2_skeleton.c t3_test.c -o t3`).

Copy the output in your report and discuss whether your results correspond to what you expected and why.

Update the `t2_skeleton.c` for "task 2 & 3" to Submitty.

Task 3 - mark allocations	
Printing the number of swaps and the number of comparisons the algorithms perform when sorting an arbitrary array	2 mark
Include your results and comment on them in the report	2 marks

```
// Sample Output

Arrays of size 10000:
  Selection sort
                    TEST |      SORTED |       SWAPS |       COMPS |
               Ascending |         YES |        9999 |    49995000 |
              Descending |         YES |        9999 |    49995000 |
                 Uniform |         YES |        9999 |    49995000 |
     Random w duplicates |         YES |        9999 |    49995000 |
    Random w/o  duplicates |         YES |        9999 |    49995000 |

  Insertion sort
                    TEST |      SORTED |       SWAPS |       COMPS |
               Ascending |         YES |           0 |        9999 |
              Descending |         YES |    49995000 |    50004999 |
                 Uniform |         YES |           0 |        9999 |
     Random w duplicates |         YES |    25096072 |    25106071 |
   Random w/o duplicates |         YES |    23993371 |    24003370 |


  Quick sort
                    TEST |      SORTED |       SWAPS |       COMPS |
               Ascending |         YES |        9999 |    50014998 |
              Descending |         YES |        9999 |    50014998 |
                 Uniform |         YES |       60517 |      121034 |
     Random w duplicates |         YES |       32174 |      163052 |
   Random w/o duplicates |         YES |       31597 |      162690 |
```

Task 4 – Most popular games (4 marks)

- You have been provided with a dataset of game reviews which have been gathered from IGN over the last 20 years. Write a program that takes the game reviews as an argument and sorts the reviews on the basis of game scores and finds out what the most popular games of the last 20 years are.

  (You may need to make use of the atoi function in order to convert the scores from strings to ints. You can see examples of this in the Pokemon solution from assignment 0.)

Submit your solution to "task 4" to Submitty.

- How would you get the top 5 games for each of the last 20 years (i.e. top ranked games for 2012, top ranked games for 2011 etc.)? (you might want to remember what we discussed about combining sorts!). **Write your approach in your report** (no need to implement it)

Task 4 - mark allocations	
Load and sort IGN reviews and print the top 10 most popular games of the last 20 years – include a short description of your approach and your results **in the report**	3 marks
Discuss how you would get the top 5 games for each of the last 20 years	1 mark

Submit your report to Submitty