CS2303_HW04

Author: Ryan Mechery

For this assignment, 2 programs were created.

1. sorttest – This program takes in numbers from the command line, puts them into an array, sorts them, and prints out time taken to sort.
2. sorttest2 – This program does much of the same as sorrtest except it uses randomly generated values and lets user select sorting function.

Compilation

To compile the program, open the directory into any unix environment and enter make to compile all the source files.

```
 $ cd <path to directory>/cs2303_hw04
 ~/<path to directory>/cs2303_hw04$ make
```

Note: To compile individual testers enter make `make sorttest` or `make sorttest2`.

To get doxygen output, configure the target folder in the makefile and enter make docs.

```
$ make docs
```

To remove object files and executables, run this command:

```
$ make clean
```

sorttest usage

```
$ ./sorttest num1 num2 nums3
```

Parameter	Type	Description
`num1`	int	**Required**. Valid integer to be put in array.
`num2`, `num3`, ...	int	Optional. Valid integer to also put in array

sorttest2 usage

```
$ ./sorttest2 array_size sort_type
```

Parameter	Type	Description
`array_size`	int	**Required**. Size of random array.
`sort_type`	string	**Required**. Type of sorting arrays. Either `sort_descending` or `alt_sort_descending`.

`sort_type` options	Description
`sort_descending`	Regular bubble sort using array with indexes.
`alt_sort_descending`	Optimized bubble sort using pointer access.

tests

```
 $ ./sorttest 0 1 2 3
Unsorted array:
0
1
2
3
Sorted array (descending order):
3
2
1
0
Timestamp before sorting: Seconds 1675917181,   Microseconds:515155
Timestamp after sorting: Seconds 1675917181,    Microseconds:515155
Time spent sorting: Seconds 0,  Microseconds:0
```

Test #1a: Random Range [0,10]

```
 $ ./sorttest2 5000 sort_descending
(list of numbers) ...
Timestamp before sorting: Seconds 1676122056,   Microseconds:14497
Timestamp after sorting: Seconds 1676122056,    Microseconds:54418
Time spent sorting: Seconds 0,  Microseconds:39921
```

Test #1b: Random Range [0,10]

```
 $ ./sorttest2 5000 alt_sort_descending
(list of numbers) ...
Timestamp before sorting: Seconds 1676122914,   Microseconds:680067
Timestamp after sorting: Seconds 1676122914,    Microseconds:714086
Time spent sorting: Seconds 0,  Microseconds:34019
```

Running test one shows that alt_sort is 1.17 times faster than running the regular bubble sort.

Test #2a: Random Range [0,1000]

```
 $ ./sorttest2 10000 sort_descending
(list of numbers) ...
Timestamp before sorting: Seconds 1676122207,   Microseconds:988680
Timestamp after sorting: Seconds 1676122208,    Microseconds:156638
Time spent sorting: Seconds 0,  Microseconds:167958
```

Test #2b: Random Range [0,1000]

```
 $ ./sorttest2 10000 alt_sort_descending
(list of numbers) ...
Timestamp before sorting: Seconds 1676122766,   Microseconds:735068
Timestamp after sorting: Seconds 1676122766,    Microseconds:886740
Time spent sorting: Seconds 0,  Microseconds:167958
```

Running this test again with a different max and more numbers shows that the optimized sort is around 1.11 times faster.

My reasoning for this 10% performance gain is that the time cost of accessing array elements with an index multiplies in an O(n^2) method such as bubble sort. Allowing the computer act more closely as a Turing computer with just one pointer prevents the computer from calculating unnecessarily calculations to swap a little bit faster. But ultimately, swapping takes more time than calculating memory addresses.

Test #3: Invalid `sort_type`

```
 $ ./sorttest2 10 new_sort
"new_sort" is not a valid sort type!
Only "sort_descending" and "alt_sort_descending" are allowed.
```