Assignment Three Design.pdf

By James Yim

Description of Program:

This program will showcase different types of sorts. We will learn how different sorts work, why we sort things differently, and why some sorts take longer than others. We will be looking at insertion sort, heap sort, batcher sort, and quicksort.

Files to be included in asgn2

- batcher.c
  - includes the code to run batcher sort
- batcher.h
  - the header file for batcher sort
- insert.c
  - includes the code to run insertion sort
- insert.h
  - the header file for insertion sort
- heap.c
  - includes the code to run heap sort
-

- heap.h
  - the header file for heap sort
- quick.c
  - includes the code for quicksort
- quick.h
  - the header file for quicksort
- sets.h
  - implements and specifies the interface for the set ADT
- stats.c
  - implements the statistics modules
- stats.h
  - the header file for stats.c
- sorting.c
  - contains main() and allows us to pick between the different sorting algorithms
- Makefile
  - Tells the compiler how to compile the program
- README.md
  - Describes how to use the program, and how to deal with possible errors
- DESIGN.pdf
  - Describes the design of the program and the process of creating the program
- WRITEUP.pdf
  - This will show what was learned in the assignment. Showing which sorting algorithms worked faster. It will include graphs on algorithm performance.

Design & Pseudocode

insertionSort()

```
insertionSort(array listToSort)
{
    iterate through each index of the listToSort
    {
        compare the current index with the previous index
        {
            if current is greater than previous do nothing
            else swap them
        }
    }
}
```

heapSort()

```
heapSort(listToSort)
{
    first = 1
    last = last element in listToSort
    create a heap w/ listToSort, first, and last
    for each leaf from last to first
    {
        swap listToSort[first - 1] w/ listToSort[leaf - 1]
        if needed fix the heap
    }
}
```

quickSort()

```
quickSort(listToSort, low, high)
{
    if low < high
    {
        split the list into two parts
        sort each of the parts
    }
}
```

./sorting

```
#include neededLibraries

make ./sorting -H help page

arrayGenerator(*A, sizeOfArray, seed)
{
    set random to seed
    for loop to create a random array of size(sizeOfArray)
}

main()
opt = 0 //for getOpt()
sizeOfArray = 100; //100 is default
seed = 13371453 // this is default
elementsToPrint = 100; this is default
s = emptySet() //used later to “call” our sorting algorithms
batcherStats, insertStats, heapStats, quickStats = {0, 0} // this is declaring our stats struct

//Organizing user input
while (loop to get user input using getOpt())
{
    switch(opt)
    {
        cases available for user
        default case for errors
    }
}

initialize array with malloc

//knowing how many elements to print
if (elementsToPrint > sizeOfArray)
{
    set elementsToPrint to sizeOf array
}

//choosing which sort to use
for (loop through the set to see which sort type has been added)
{
    if the sort is in the set then run the sort
}

free allocated memory
```

Notes about code

- When running insertion sort and looking at moves. It is off by about two moves.
- we track the amount of moves and comparisons each sorting algorithm makes in order to compare algorithms. We use the Stats.h provided by the professor to keep track of these statistics.

Error Handling

- When an error happens as a result of a user they are directed to the -H page of ./sorting.

Credit

- Attended Eugene's Section on 1/21/22. Helped with how to use Set.h and other general tips on the assignment.
- Discord user *Toomai* helped with the bitlength function in batcher.c (bitLengths())