Assignment 3
Sets and Sorting

By Jiancheng (Jason) Xiong
CSE 13S Spring 2023

Due May 10th 2023, 11:59 PM

1 Purpose

The purpose of this assignment is to create a program that uses sets in order to store called command line arguments. We will then use a program to call different commands, including;

- Running all sorts
- Insertion sort
- Batcher's merge sort
- Quick sort
- Shell sort
- A command to choose a custom random seed, default is 13371453
- A command to allow different number of elements, default is 100
- A command to choose how many elements to print for the final sorted arrays, default is 100
- A command to print a help message

After each function, a print message with the name of the sort, the number of elements, and the number of comparisons and merges will be printed, with the help of the file stats.c.

2 Using the Program

In order to use the program, compile and run the file sorting.c. In order to run the program, run ./sorting and specify the commands in which you want to run. The available options are;

-a; running all sorting algorithms
-i; runs insertion sort
-b; runs Batcher's merge sort
-q; runs quick sort
-s; runs shell sort
-r seed; sets the srandom seed to seed, default is 13371453
-n size; sets the array size to size, default is 100

-p size; sets the number of printed elements to size, default is 100
-h; displays a help message detailing program usage.

3 Program Design

The main program will be in the file sorting.c, while each of the separate sorting files will be in;

- Insert.c and header file insert.h for insertion sort
- Batcher.c and header file batcher.h for batcher sort
- Quick.c and header file quick.h for quick sort
- Shell.c and head file shell.h for shell sort
- Gaps.h for an array of numbers to use for the gaps in shell sort
- Set.c and header file set.h for an implementation of the set struct to be used for storing of command line options
- Main file sorting.c to be the test file that runs all other functions
- Stats.c and stats.h to keep track of the number of comparisons and swaps each function uses

4 Data Structures

The main data structure is the struct set, which implements a variety of helpful functions that allow us to create and manipulate sets. The various functions allow us to change specific elements in a set, as well as check if a certain element is indeed a member of a set. By giving each argument a specific placement in a set, we can tell the program that they were called if we replace the location in that set with a 1, to indicate membership.

Another main structure are arrays, which are made with random 30 bit long numbers and passed into the sorting algorithms to be sorted.

5 Algorithms

The main function;

creates an empty set 8 elements long, with all numbers set to 0.
Each argument is naturally assigned to a certain index within that set
Parses through the command line with getopt() from stdlib.h, and if an element is present its index is set to 1 to indicate that it has been asked for.

After parsing the entire argument, the set is iterated through, and for elements that have a 1, indicating membership, the corresponding algorithm or function is run.

For the sorting algorithms;

Insertion sort

1. Start by iterating each element in the unsorted array, starting with the second element
2. For each element in the loop, compare it to the elements before it, starting with the previous element
3. If the current element is smaller than the previous element, swap their positions
4. Repeat step 3 until the current element is larger than the previous element
5. Repeat for all unsorted elements

Batcher sort

1. First divide the unsorted array into 2 parts, and recursively sort each part
2. Once the two halves are sorted, merge them using;
    a. Compare the elements at the same indice of the two halves
    b. If the element in the second half is smaller than the first, swap their positions
    c. Repeat the comparison for every element in the two halves

3. Repeat the merges recursively until the entire array is remerged, until which it is sorted

Quick sort

1. Choose a pivot element, which can be done randomly or simply selecting the middle element
2. Partition the array into two subarrays: one containing all the elements smaller than the pivot, and the other will all greater elements
3. Recursively apply the quicksort algorithm to the subarrays

Shell sort

1. Start with a gap value of length of array/2

2. Divide the array into smaller subarrays of the length gap
3. Repeat step 2 for values of len/4, len/8, etc.. until gap is equal to 1
4. Perform a final insertion sort on the array

For the set implementation:
All arrays mentioned are of length 8
Set_empty(void):
Create an empty array of 0s

Set_universal(void):
Create an array of 1s

Set_insert(set, x):
Takes the input set and logic ors it with 0x01 shifted left x times

Set_remove(set,x):
Takes the input set and logic ands it with not 0x01 shifted left x times

Set_member(set, x):
Returns true if (set & 0x01 shifted left x times) is a 1, otherwise returns false

Set_union(set1, set2):
Returns all elements in set1 or set2

Set_intersect(set1,set2):
Returns all elements in both set1 and set2

Set_difference(set1,set2):
Returns all elements in set1 but not in set2

Set_complement(set):
Returns the opposite of set (not set)

6 Function Descriptions

Main function: takes its arguments from the command line, using getopt to know what sorting functions to run. Takes one (command line) input and outputs the sorted arrays requested along with the stats of the sorts

All Sorting Functions: takes the arguments (stats, input array, length of array). The sorting functions then sort the input array and output nothing.

7 Results

TBA

8 References

TBA