Assignment 3 – Sets and Sorting

Aditya Bhaskar

CSE 13S – Spring 2023

Purpose

This program is a command-line tool that implements a collection of comparison-based sorting algorithms. Specifically, the program provides the options to sort an array of random integers using any combination of the following sorting algorithms: insertion sort, heap sort, shell sort, quicksort, and batcher sort. The program accepts command-line arguments to configure the behavior of the sorting algorithms. For example, users can specify the size of the array, the random seed used to generate the random integers, and the number of elements to print after the sorting process. Additionally, the program provides a help menu with a synopsis of its usage, descriptions of its options, and an overview of the supported algorithms. The program uses a Stats structure to keep track of the number of comparisons and moves made during the sorting process, and the print-stats function is called to output the statistics of each sorting algorithm.

How to Use the Program

Welcome to our mathematical library program! Here's a guide on how to use it effectively.

- **Compiling the program:** To compile the program, you'll need to have a C compiler installed on your system, such as GCC or Clang. Once you have a compiler installed, navigate to the directory where you've saved the program's source code and run the following command in your terminal:

  ```
  clang -lm program -o program.c
  ```

  This command will compile the program and create an executable file named program in the same directory. Or you can just use a makefile:

  ```
  make all
  ```

- **Running the program:** Once you have compiled the program, you can run it by typing the following command in your terminal:

  ```
  ./sorting.c
  ```

  This will run the program with its default options. The program will then prompt you for input.

- **Program options:** The program has several options that you can use to customize its behavior. Here is a list of the available options:
  - **-a:** implements all the sorts.
  - **-i:** implements the insertion sort.
  - **-s:** implements the selection sort.
  - **-q:** implements the quick sort.
  - **-b:** implements the batcher sort.
  - **-h:** implements the heap sort.
  - **-r SEED:** initializes the seed value.
  - **-n length:** initializes the value of the length of the array.

  - **-p elements:** initializes the value of elements to be printed/displayed.

  You can use these options by adding them to the command line when you run the program. For example, to specify an input file and an output file, you would run the following command:

  ```
  ./sorting -i -n 10 -p 10 -r 1000
  ./sorting -a -n 100 -p 0 -r 10000
  ./sorting -iqb -n 10 -p 10 -r 1000
  ```

- **Removing files created in compilation:** If you want to remove the executable file created during compilation, you can use the following command:

  ```
  rm sorting
  ```

  This will delete the program file from the directory where it was created.

Program Design

This program is a collection of comparison-based sorting algorithms and is designed to take command-line arguments to specify the type of sorting algorithm to use, the size of the array to be sorted, and other options. The program is organized into several C source code files, each containing a different sorting algorithm implementation, as well as a stats file that contains functions to keep track of the statistics of the sorting algorithms.

Main Function: The main function takes command-line arguments and sets variables to determine which sorting algorithm to use and how to sort the array. The program provides help to the user by displaying a usage message when the -H flag is specified. The main function then calls the specified sorting function, displays the statistics of the sort, and prints the sorted array to the console.

Data Structures: The program uses only a single data structure, a dynamical array of unsigned 32-bit integers, to represent the data to be sorted. The size of the array is specified by the user through the command-line argument.

Algorithms: The program contains several sorting algorithms, including insertion sort, heap sort, shell sort, quick sort, and batcher sort. Each sorting algorithm has its own implementation in a separate source code file. The main function calls the appropriate sorting function based on the command-line arguments specified by the user.

Stats: The stats file contains functions to keep track of the statistics of the sorting algorithms, including the number of comparisons, the number of swaps, and the time taken to sort the array. These statistics are used to measure the efficiency of the sorting algorithms and to compare them with one another.

Overall, the program is well-organized, with separate files for each sorting algorithm and a stats file to keep track of the statistics. The main function provides a user-friendly interface for the user to specify which sorting algorithm to use and how to sort the array, and it displays the statistics of the sort and the sorted array to the console.

Data Structures

The program implements several sorting algorithms and allows the user to select which algorithms to run, as well as providing options for controlling the length of the array to sort and how much of it to print. The sorting algorithms themselves are defined in separate files, with each file containing the implementation of a different sorting algorithm.

The program uses several data structures to keep track of the state of the sorting algorithms and the options passed into the program. These data structures include:

- **Integer variables for each option:** The program uses several integer variables (a, i, s, h, q, b, p, H) to keep track of which sorting algorithms to run and which options to enable. These variables are set to 1 if the corresponding option is passed in as a command line argument, and 0 otherwise.
- **A Stats structure:** The program defines a Stats structure that is used to keep track of various statistics about the sorting algorithms, such as the number of comparisons and swaps performed. The Stats structure contains several integer fields, including compares, swaps, and moves.
- **An array to sort:** The program generates a random array of integers of a user-defined size using the rand() function and stores it in a dynamically allocated array, arr.
- **Integer variables to control array size and printing:** The program defines three integer variables, seed, size, and print, to control the size of the array to sort, the number of elements to print, and the random seed used to generate the array. These variables are set to default values if not provided by the user.
- **A while loop to parse options:** The program uses a while loop and the getopt() function to parse the command line options passed in by the user. The options are processed in a switch statement, with each option setting its corresponding integer variable to 1 and modifying the appropriate control variable if necessary.
- **A reset() function:** The program defines a reset() function that resets the Stats struct by setting all of its fields to 0.
- **Sorting algorithm functions:** The program defines separate functions for each of the sorting algorithms implemented, such as insertion-sort(), heap-sort(), etc. Each of these functions takes in the Stats structure and the array to sort as arguments and modifies the array in place to be sorted, while also updating the statistics in the Stats structure.
- **A print-stats() function:** The program defines a print-stats() function that takes in the Stats structure, the name of the sorting algorithm being used, and the size of the array as arguments. The function prints out the statistics in a formatted manner.

Overall, the program uses a combination of integer variables, data structures, and control variables to manage the state of the sorting algorithms and the options passed in by the user. The sorting algorithms themselves modify the array to be sorted in place while updating the Stats structure, and the print-stats() function is used to print out the final statistics in a readable format.

Algorithms

The following snippet contains the pseudocode of the **main()** of **sorting.c**:

```
main():
    opt = 0;
    aflag = False, iflag = False, qflag = False, bflag = False, sflag = False
    hflag = False, Hflag = False

    while (opt = getopt(argc, argv, COMMAND != -1):
        switch (opt):
            case ’a’:
                aflag = True
                break
            case ’i’:
                iflag = True
                break
            case ’q’:
                qflag = True
                break
            case 'b':
                bflag = True
                break
            case 's':
                sflag = True
                break
            case 'h':
                hflag = True
                break
            case 'H':
                Hflag = True
                break
            case 'n':
                length = input
                break
            case 'p':
                pflag = True
                elements = input
                break
            case 'r':
                seed = input
                break
            default:
                hflag = True
                break

     if (no commands are passed):
        Hflag = 1

    srandom(seed)

    dynamically initialize an array with random values using mallloc

    if (Hflag is True):
        printf("SYNOPSIS\n");
        printf("   A collection of comparison-based sorting algorithms.\n\n");

        printf("USAGE\n");
        printf("   ./sorting-x86 [-Hahbsqi] [-n length] [-p elements] [-r seed]\n\n");

        printf("OPTIONS\n");
        printf("   -H              Display program help and usage.\n");
        printf("   -a              Enable all sorts.\n");
        printf("   -h              Enable Heap Sort.\n");
        printf("   -b              Enable Batcher Sort.\n");
        printf("   -s              Enable Shell Sort.\n");
        printf("   -q              Enable Quick Sort.\n");
        printf("   -i              Enable Insertion Sort.\n");
        printf("   -n length       Specify number of array elements (default: 100).\n");
        printf("   -p elements     Specify number of elements to print (default: 100).\n");
        printf("   -r seed         Specify random seed (default: 13371453).\n");

    if (aflag || iflag):
        insert_sort()
        print_stats()
        if (pflag):
            print(array)

    if (aflag || hflag):
        heap_sort()
        print_stats()
        if (pflag):
            print(array)

    if (aflag || qflag):
        quick_sort()
        print_stats()
        if (pflag):
            print(array)

    if (aflag || sflag):
        shell_sort()
        print_stats()
        if (pflag):
            print(array)

    if (aflag || bflag):
        batcher_sort()
        print_stats()
        if (pflag):
            print(array)

    free(array)
    return 0
```

Function Descriptions

- The main() function:
  - Parses command-line arguments using getopt()
  - Initializes variables based on command-line options or default values
  - Generates an array of random integers based on the provided seed and size
  - Calls various sorting algorithms based on the command-line options provided, passing in the array of integers and a Stats structure
  - Prints out sorting statistics and optionally a portion of the sorted array based on the -p option
  - If no options or invalid options are provided, displays program help and usage information
- The insertion-sort() function:
  - Sorts an array of integers using the insertion sort algorithm
  - Modifies the input array in place
  - Takes a Stats struct to record sorting statistics
- The heap-sort() function:

  - Sorts an array of integers using the heap sort algorithm
  - Modifies the input array in place
  - Takes a Stats struct to record sorting statistics

- The shell-sort() function:
  - Sorts an array of integers using the shell sort algorithm
  - Modifies the input array in place
  - Takes a Stats struct to record sorting statistics

- The quick-sort() function:
  - Sorts an array of integers using the quick sort algorithm
  - Modifies the input array in place
  - Takes a Stats struct to record sorting statistics

- The batcher-sort() function:
  - Sorts an array of integers using the batcher sort algorithm
  - Modifies the input array in place
  - Takes a Stats struct to record sorting statistics

- The print-stats() function:
  - Prints out sorting statistics recorded in a Stats struct

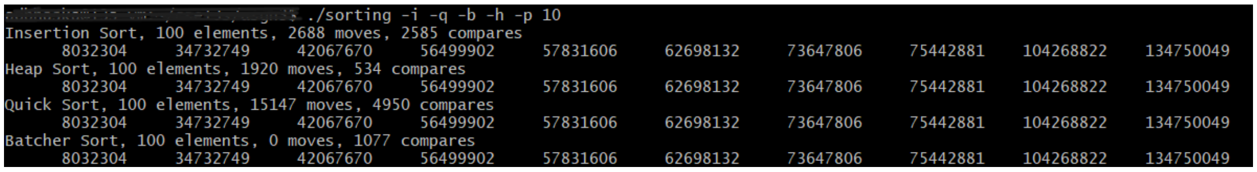
- The reset() function:
  - Resets the values in a Stats struct to 0.

Results

It accurately sorts randomly generated arrays using the different algorithms.

Numeric results

Fig. 1.



```
[redacted] ./sorting -i -q -b -h -p 10
Insertion Sort, 100 elements, 2688 moves, 2585 compares
     8032304     34732749     42067670     56499902     57831606     62698132     73647806     75442881    104268822    134750049
Heap Sort, 100 elements, 1920 moves, 534 compares
     8032304     34732749     42067670     56499902     57831606     62698132     73647806     75442881    104268822    134750049
Quick Sort, 100 elements, 15147 moves, 4950 compares
     8032304     34732749     42067670     56499902     57831606     62698132     73647806     75442881    104268822    134750049
Batcher Sort, 100 elements, 0 moves, 1077 compares
     8032304     34732749     42067670     56499902     57831606     62698132     73647806     75442881    104268822    134750049
```

Figure 1: Screenshot of the program running.

Error Handling

There are fairly small differences between some values.

References