Dmitrius Agoston
Prof. Darrell Long
17 October 2021

CSE 13S Fall 2021
Assignment 3: Sorting
Design Document

Description of Program:
The purpose of this program is to implement four different sorts on an array of random numbers given a seed, size and number of elements. The sorts that are implemented are insertion sort, shell sort, heap sort, and quick sort. Once the tests for the sorts are run, the ordered array will be printed and statistics for each sort's moves and comparisons will be given. With the data given the most efficient sort for that particular array of numbers can be determined.

Layout/Pseudocode:

- insert.c

  This file contains the function for insertion sort. The insertion sort works by taking each element and comparing it to its predecessors. If the current element is smaller than its predecessors then it keeps getting moved until it is in the right position. This process repeats until the array is ordered.

  ```
  Define insertion sort with parameters for stats, array, and size
      Loop through array
          Set previous to current
          Move current element to temporary variable
          While previous is greater than 0 and compare temporary and prev
              Move previous element
              Previous decrement by 1
          Array in previous position set equal to temp
  ```
- shell.c

  The file contains the function for shell sort. The shell sort works by using Knuth's gap sequence. This sort sorts the array by first comparing elements that are far apart from each other. This space between the two is referred to as the gap and this gap is decremented through each sorted element. Through this process the gaps will get smaller until the gap is a size of 1. At this point the array will be fully sorted.

  ```
  Define shell sort with parameters for stats, array, and size
      Variable for gap
  ```

```
Generate the gap
    Loop through the gap and length of array
    Previous set to current
    Move current element to temporary variable
    While previous is less than or equal to gap and
    compare temp var and variable previous - gap
        Move current element
        Decrement previous by gap
    Set current element to temporary variable
```

- heap.c

  This file contains the function for heap sort and the helper functions for building and fixing the heap. This sort works by first building a heap from the given array. In this case, building the heap requires finding the largest element and putting that as the first element when built. After this the heap must be fixed and sorted. This is done by removing the largest element from the array and placing it at the end. The heap needs to be fixed after this every time and the process starts over again.

```
Define max child function with parameters for array, first, and last
    Left set equal to 2 * first
    Right set equal to left + 1
    Check if right is <= to the last and right is > left
        Return right if so
    If not return left

Define fix heap function with parameters for array, first, and last
    Set found = to false
    Set mother = to first
    Set great = to the max child with mother and last as parameters
    While mother <= last floor divide by 2 and found equals false
        If current is less than previous
            Current and previous swap
            Set mother equal to great
            Great = max child
        Else
            Set found = to true

Define build heap with parameters for array, first, and last
    Loop for father in range of last floor divided by 2 to first - 1 and decrement
    Fix heap
```

```
Define heap sort with parameter for stats, array, and size
    First equal to 1
    Last equal to size of array
    Build the heap
    Loop for the leaf in range of last and first
        Swap leaves
        Fix heap
```

- quick.c

  The file contains the function for quick sort, quick sorter, and the function partition. This sort works by partitioning the array it is sorting into two sub arrays and using pivot points to sort the arrays. The pivot point checks if the element is less than itself or greater and based on the answer will either put it to the left or right in the array. After the array is partitioned the quick sort is called recursively on the partitioned parts and sorts them.

```
Define partition with parameters for array, low, and high
Set i equal to low - 1
Loop for j from low to high
    If current element  < high element
        I increment 1
        Swap
Swap
Return i + 1
```

```
Define quick sorter with parameters for array, low, and high
    If the low is less than high
    Set p = to partition
    Partition array to two separate ones
```

```
Define quick sort with parameters for stats, array, and size
    Call quick sorter
```

- sorting.c

  This file contains the test harness for testing the sorts. The test harness accepts different command inputs for choosing different tests as well as different settings for the test. The commands accepted are “a” for all tests, “e” for heap sort “i” for insertion sort, “s” for shell sort, “q” for quick sort, and “n”, “p”, and “r” for the length, seed, and number of elements desired. There is also an “h” function for general help and usage purposes. The test harness uses sets to determine which tests to run as opposed to a bunch of booleans. When the desired test are run with the pseudo random array, the corresponding statistics will be given as well for data on the number of moves and comparisons made.

```
Create enumerations for sorts

Define help function
Print help text

Define make array function
Set random seed
Bitmask array with desired seed and length

Define main with parameters for argc and **argv
Variable for empty set
Variable for current optarg
Variable for opt
Variable for seed
Variable for size
Variable for elements
Variable for help test
Loop and parse through command line
    Cases for testing: insert case into set
    Cases for seed, size, and elements
    Case for all test, insert all cases
    Case for help, return 0
If size < elements, elements equals size
If help variable not disabled
    Run help function
    Return 0
Loop through set
    If test in set, run test and give stats
```

- stats

  The stats for each sort are tracked through stats.c and stats.h. In these files the typedef struct Stats is defined where the number of compares and moves can be stored. Using the functions in the actual code of the sorts helps accurately keep track of the moves and comparisons rather than substituting what they do and having a separate variable for each sort. There are four functions that are made use of. First being reset() where the statistics for both moves and comparisons will be wiped clean making them ready to count for a new sort and return data accurately representing its process. The function cmp() compares two different elements and increments for each comparison. The other two are move() and swap() which both account for the total number of moves that each sort has to complete in order to sort its list.

Error Handling:

The main error handling is making sure the input given can be used in the tests. If a number is too big or negative for the seed, size, or elements, the default values for these variables will be used instead. Another error that had to be handled was dynamically allocating the array to test with sizes up to the memory limit of the computer.

Credit:

When creating this code I largely used asgn3.pdf as my main source of reference for creating this program. I also viewed Eugene's section recording which I used to help come up with my test harness.