Assignment 3 - Sets and Sorting

Chucheng Xie

CSE 13S – Spring 2023

Purpose

This assignment will implement a total of five sorting algorithms, including Insertion Sort, Shell Sort, Heap Sort, Quick Sort, and Batcher Sort. Each algorithm takes in a structure called stats, an array, and the size of the array as parameters, and will count the number of compares and moves that occur during the sorting process.

Program Design

Pseudocode:

- **heap.c**

```
max_child(A: list, first: int, last: int):
   left = 2 * first
   right = left + 1
   if right <= last and A[right - 1] > A[left - 1]:
      return right
   return left


fix_heap(A: list, first: int, last: int):
   found = False
   mother = first
   great = max_child(A, mother, last)

   while mother <= last // 2 and not found:
      if A[mother - 1] < A[great - 1]:
         A[mother - 1], A[great - 1] = A[great - 1], A[mother - 1]
         mother = great
         great = max_child(A, mother, last)
      else:
            found = True

build_heap(A: list, first: int, last: int):
    for father in range(last // 2, first - 1, -1):
        fix_heap(A, father, last)

heap_sort(A: list):
    first = 1
    last = len(A)
    build_heap(A, first, last)
    for leaf in range(last, first, -1):
        A[first - 1], A[leaf - 1] = A[leaf - 1], A[first - 1]
        fix_heap(A, first, leaf - 1)
```

- **batcher.c**

```
comparator (A: list, x: int, y: int):
    if A[x] > A[y]:
        A[x], A[y] = A[y], A[x]

(A: list):
    if len(A) == 0:
        return

    n = len(A)
    t = n.bit_length()
    p = 1 << (t - 1)

    while p > 0:
        q = 1 << (t - 1)
        r = 0
        d = p

        while d > 0:
            for i in range(0, n - d):
                if (i & p) == r:
                    comparator(A, i, i + d)
                d = q - p
                q >>= 1
                 r = p

        p >>= 1
```

- **shell.c**

```
(array, length)
    reset global variables
    for item in gaps:
        for (int i = item; i < length; i++):
            index = i;
            holder = array[i];

            while (index >= gap && array[index - gap] > holder):
                swap;
                index -= gap;
            array[index] = holder;

partition(A: list, lo: int, hi: int):
    i = lo - 1
    for j in range(lo, hi):
        if A[j - 1] < A[hi - 1]:
            i += 1
            A[i - 1], A[j - 1] = A[j - 1], A[i - 1]
    A[i], A[hi - 1] = A[hi - 1], A[i]
    return i + 1
```

- **quick.c**

```
(array, length)
```

```
    reset global variables
    i = 0, j = length - 1;
    push to stack: i and j;
    max_stack_size += 2;

    while (stack != empty):
        partition = partition(array, j, i)

        comparisons ++
        if (partition > i):
            push to stack: i and partition
            max_stack_size += 2
        comparisons ++
        if (partition + 1 < j):
            push to stack: partition + 1 and j
            max_stack_size += 2
```

- **insert.c**

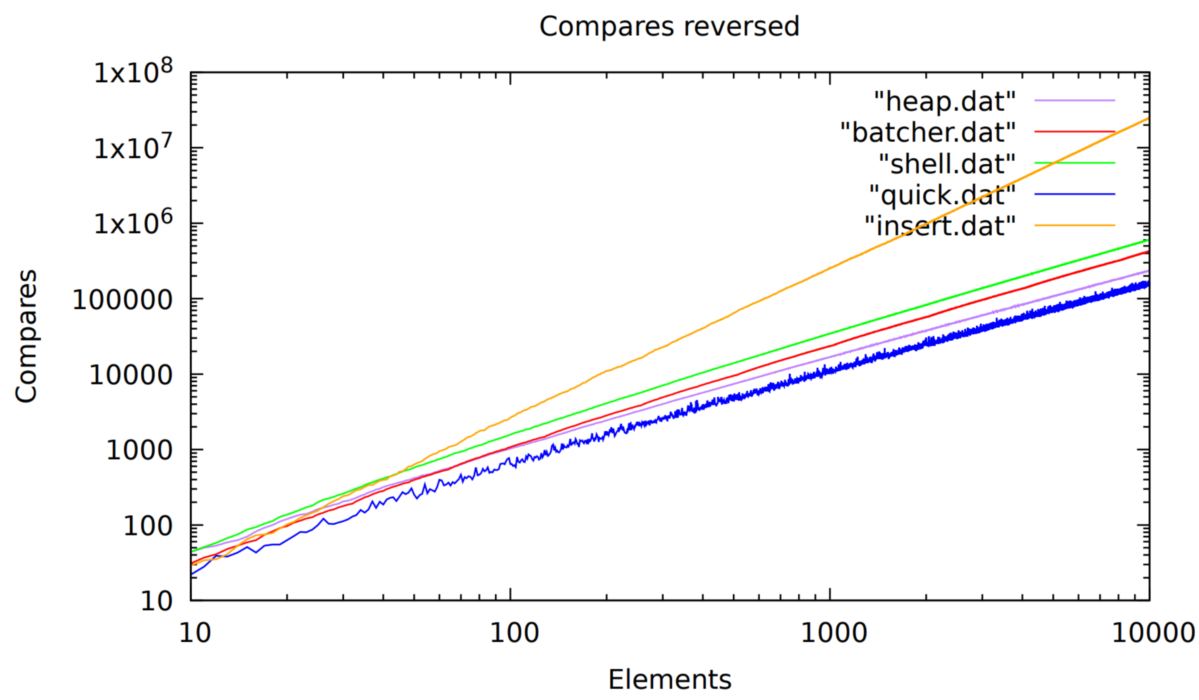
```
insertion_sort(A: list):
    for k in range(1, len(A)):
        j = k
        temp = A[k]
        while j > 0 and temp < A[j - 1]:
            A[j] = A[j - 1]
            j -= 1
        A[j] = temp
```

- **sorting.c**

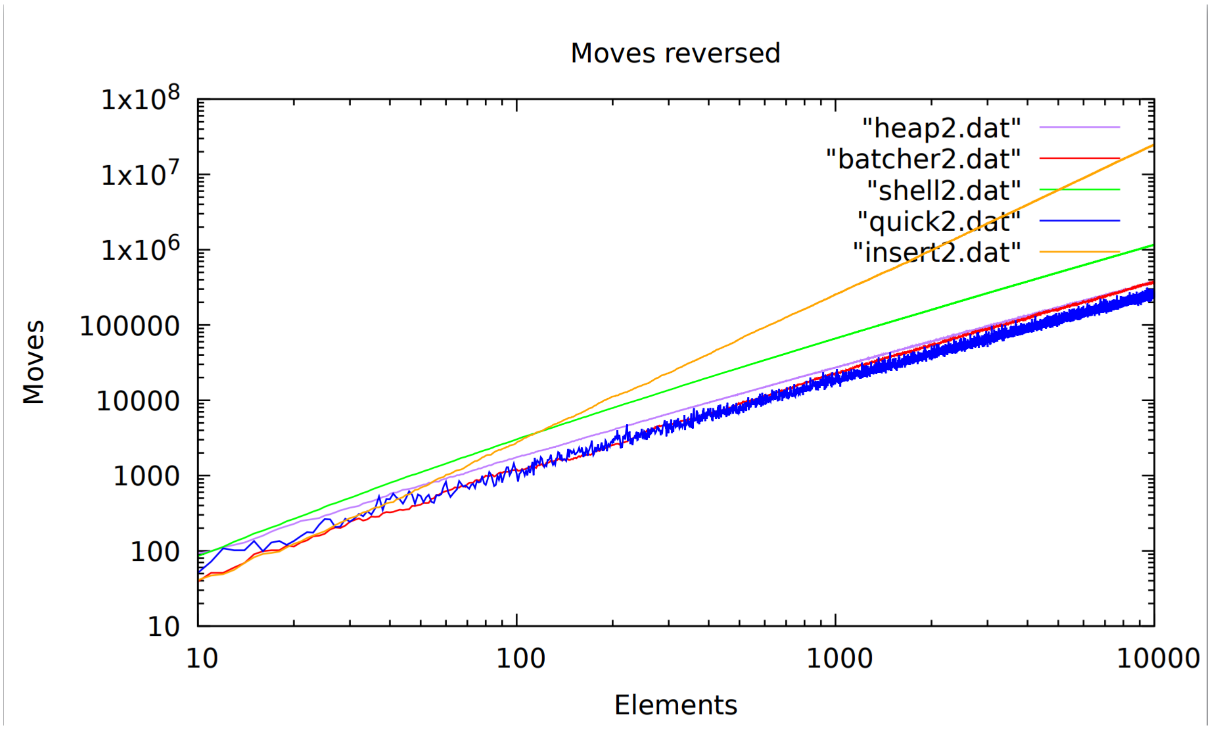
```
include all function header files
include inttypes.h, stdio.h, stdlib.h, and unistd.h

define OPTIONS
```

```
create enums for all the sorts and options
create an array to store the names of the sorts
set elements and size to default
set seed to default

create a function print_help() to print out all of the help message
return void

main
free memory allocated to arrays
return void
```

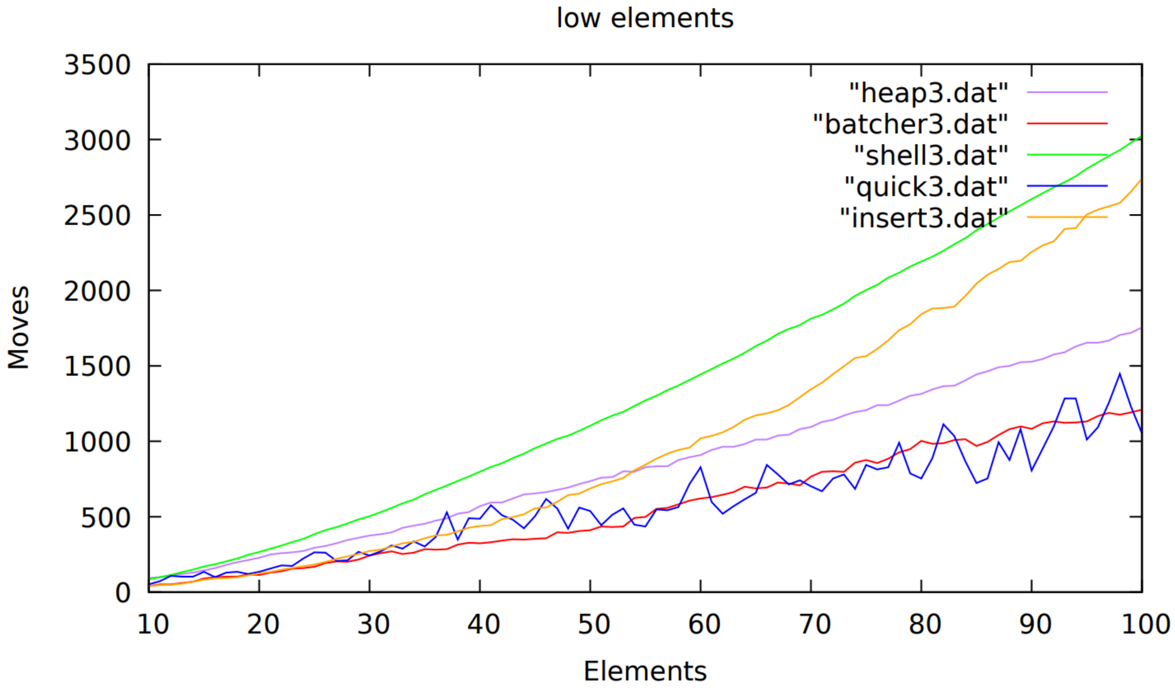
Result



In terms of how many compares each function utilizes, we can see some slight differences from the previous graph. Quick Sort uses the least amount of compares consistently throughout the entire range, apart from a few spikes, and Insertion Sort uses the most compares.

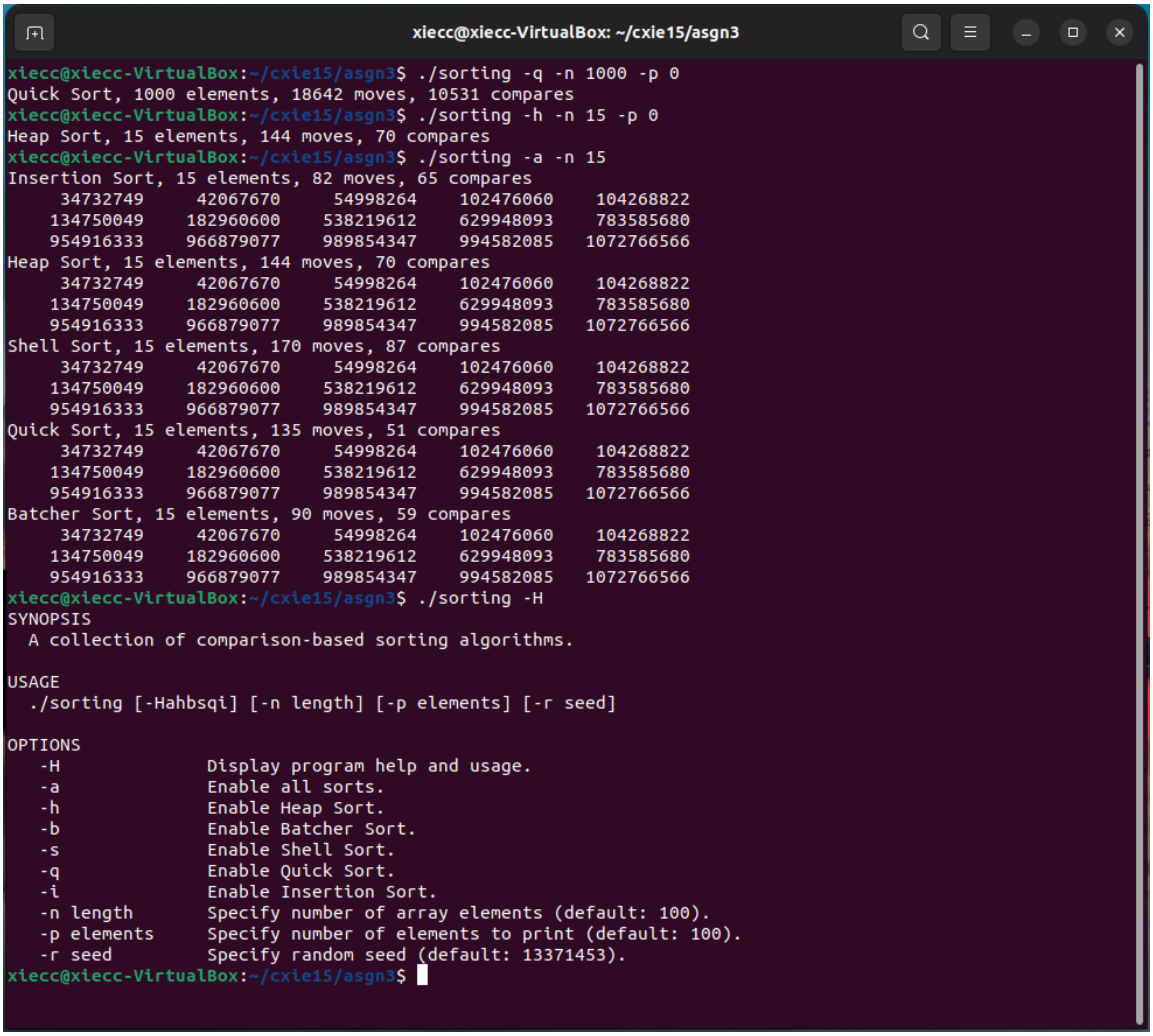


In terms of how many moves it takes for each function in this range, we can see that Insertion Sort becomes exponentially worse than the other functions. Quick Sort still spikes quite a bit. Heap Sort remains behind Shell Sort, but seems to get nearer to it as the number of elements increase. From this, we can conclude that Quick Sort is the best for this range. Insertion Sort should not be used to many elements without it taking a very long time.



In terms of how many moves it takes each function, we can see that Shell Sort takes the most moves through 10 to 100 elements. When Quick Sort handles from 10 to 100 elements, it is very unstable. Batcher Sort is very stable when processing from 10 to 100 elements, and moves are relatively small.

Sample Output:



```
xiecc@xiecc-VirtualBox:~/cxie15/asgn3$ ./sorting -q -n 1000 -p 0
Quick Sort, 1000 elements, 18642 moves, 10531 compares
xiecc@xiecc-VirtualBox:~/cxie15/asgn3$ ./sorting -h -n 15 -p 0
Heap Sort, 15 elements, 144 moves, 70 compares
xiecc@xiecc-VirtualBox:~/cxie15/asgn3$ ./sorting -a -n 15
Insertion Sort, 15 elements, 82 moves, 65 compares
     34732749     42067670     54998264    102476060    104268822
    134750049    182960600    538219612    629948093    783585680
    954916333    966879077    989854347    994582085   1072766566
Heap Sort, 15 elements, 144 moves, 70 compares
     34732749     42067670     54998264    102476060    104268822
    134750049    182960600    538219612    629948093    783585680
    954916333    966879077    989854347    994582085   1072766566
Shell Sort, 15 elements, 170 moves, 87 compares
     34732749     42067670     54998264    102476060    104268822
    134750049    182960600    538219612    629948093    783585680
    954916333    966879077    989854347    994582085   1072766566
Quick Sort, 15 elements, 135 moves, 51 compares
     34732749     42067670     54998264    102476060    104268822
    134750049    182960600    538219612    629948093    783585680
    954916333    966879077    989854347    994582085   1072766566
Batcher Sort, 15 elements, 90 moves, 59 compares
     34732749     42067670     54998264    102476060    104268822
    134750049    182960600    538219612    629948093    783585680
    954916333    966879077    989854347    994582085   1072766566
xiecc@xiecc-VirtualBox:~/cxie15/asgn3$ ./sorting -H
SYNOPSIS
  A collection of comparison-based sorting algorithms.

USAGE
  ./sorting [-Hahbsqi] [-n length] [-p elements] [-r seed]

OPTIONS
   -H              Display program help and usage.
   -a              Enable all sorts.
   -h              Enable Heap Sort.
   -b              Enable Batcher Sort.
   -s              Enable Shell Sort.
   -q              Enable Quick Sort.
   -i              Enable Insertion Sort.
   -n length       Specify number of array elements (default: 100).
   -p elements     Specify number of elements to print (default: 100).
   -r seed         Specify random seed (default: 13371453).
xiecc@xiecc-VirtualBox:~/cxie15/asgn3$
```