Sorting: Design

Ralph Miller

October 17th 2021

1 Objective

In this assignment I will be implementing four different sorting algorithms in C. Insertion sort, Shell sort, Quick sort, and Heap sort. A test harness will be created to facilitate running the sorting algorithms given, as well as facilitating data collection for later analysis.

2 Sorting Algorithms

2.1 Insertion Sort

This sorting algorithm, sorts an array in increasing order, placing them in their correct order. It compares the k-th element with each of the preceding elements in descending order until its position is found. Insertion Sort's time complexity is $O(n^2)$

Inputs: Array[] of 30 bit integers, length of array: n
Output: Sorted Array[] from least to greatest

```
for 1 ≤ k ≤ n do
    j = 1;
    compare A[k] with A[k-j];
    if A[k] > A[k-j] then
        A[k] is in the right place;
        increment k by 1;
    end
    if A[k] < A[k-j] then
        move A[k-j] to A[k];
        increment j by 1;
    end
end
```

Algorithm 1: Insertion Sort Pseudocode

2.2 Shell Sort

This sorting algorithm is a variation of the insertion sort algorithm. It first sorts pairs of elements which are far away from each other. The distance between these pairs is known as a gap. In each succeeding iteration of the shell sort the gap is decreased until a gap of 1 is used. For this assignment we will be using Knuth's gap sequence $[3^k - 1/2]$. Which gives us a time complexity of $O(n^{3/2})$

Inputs: Array[] of 30 bit integers, length of array: n
Output: Sorted Array[] from least to greatest

```
for each gap in gap sequence do
    compare A[k] with A[gap];
    if A[k] > A[gap] then
        A[k] is in the right place;
    end
    if A[k] < A[gap] then
        swap A[gap] to A[k];
    end
end
```

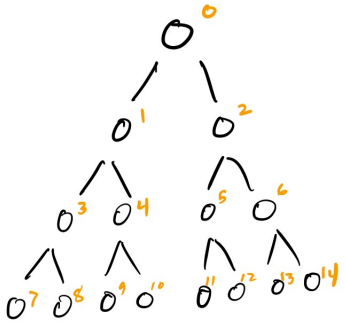
Algorithm 2: Shell Sort Pseudocode

2.3 Quick Sort

Quick Sort is an in place sorting algorithm, utilizing a pivot to partition two sub arrays around this pivot. Then recursively sorts the sub arrays. Its average time complexity is $O(nlog(n))$. With a worst case time complexity of $O(n^2)$.

2.4 Heap Sort

Heap Sort is an interesting sorting algorithm that uses the heap data structure. My implementation of Heap Sort uses a Max Heap in which the parent node is greater than its two children. The time complexity of Heap Sort is $O(nlog(n))$. The Pseudo Code is drawn below.



Start with index 6
do if statements
until index 0 is reached

index 6
left child = 6 x 2 + 1
right child = 6 x 2 + 2

if

- is lc > index 6
  swap LC + in 6
- is rc > index 6
  swap rc + in 6
- is the root different?
  do index 5

When index 0 is reached
this should be the biggest
number

We can swap index 0 with
the last index and
remove the last index
from the sort

And call heap sort again
until there is only
one node left

3 Test Harness

The test harness facilitates the functionality of all sorts using sets and getopt command line arguments. Collects statistical data through stats.c functions, and prints the statistics and sorted elements.

3.1 Command Line Options

```
-a: Employs all sorting algorithms.
-e: Enables Heap Sort.
-i: Enables Insertion Sort.
-s: Enables Shell Sort.
-q: Enables Quick sort.
-r seed: Set the random seed to seed. The default seed is 13371453.
-n size: Set the array size to size. The default size is 100.
-p elements: Print out elements number of elements from the array.
    The default number of elements to print is 100.
-h: Prints out program usage.
```

3.2 Data Collection

Data collection will be tracked with the included Stats.c file. For the sorting algorithms it will track moves, swaps, and comparisons. Printing out to the console the amount of moves, swaps, or comparisons the sorting algorithm took to sort the array.

- A move is when the array is shifted, this is mainly used for the Insertion Sort algorithm.
- A swap is when two elements of the array are swapped.
- A comparison is when two elements of the array are compared.