CSE13s Winter 2022
Assignment 3: Sorting - Putting your affairs in order

Theodore Ikehara, tikehara@ucsc.edu

CSE 13S - Winter 2022

Due: January 26th at 11:59 pm

Writeup

What I learned:

During this assignment I was able to gain more knowledge on more than just sorting routines. We learned much about the time complexity of the objects. In computer science many of the things that we write can take many different forms. This is to say that many of the functions and methods we write can do the same thing as another function that is written completely differently. In other words there can be many functions that do the same thing but do not resemble each other at all. This assignment we performed on sorting is exactly this. We completed four sorting routines that look completely different and work differently fundamental but all do the exact same thing. The output would be the same no matter what you pass it. The difference between these sorts is the speed. And largely this is the biggest thing that we care about when writing functions like these. We want to be able to choose the most optimal algorithm for the job. Even though perfect does not exist in this case we can perform the fastest known algorithm. However, it is important to understand the methods in what makes a function slow or fast. This is where calculating time complexity comes into play. We know that the programs run at different speeds but we don't know how to visualize that or understand and make sense of the information we are given. We are able to calculate them by analysing the code that we wrote and see why and how these sorting routines perform better or worse than one another. Overall, in this assignment we learned the pros and cons of using different algorithms and learned how to analyze the written code and conclude the performance of the program based on how it was written.

What I learned on the individual sorting algorithms:

1. Insertion sort:

   Insertion sort was amongst the simplest of the sorts. This was a simple sort with a nested loop one for traversing the array and one for traversing the location to move the element to. This had one of the worst if not the worst time complexity out of the sorting algorithms. However, this algorithm was the easiest to understand, and write. This was also one of the shortest to write and took the least amount of code. Given this, even though it takes the least amount of code to execute this algorithm runs the slowest out of all the algorithms here.

2. Quick sort:
   Quick sort was one of the fastest sorts; this was also one of the only sorts that required recursion. Given this nature of recursion and how the algorithm uses recursion quicksort is a divide and conquer algorithm and given this nature is much faster than many other sorts this algorithm even though much longer and harder to understand is always going to be faster than insert sort.
3. Heap sort:
   In this sort we use a heap or nodes with parent child relations in order to sort a list. Inorder to perform this algorithm we needed to create two functions one was in charge of creating the heap and one was in charge of maintaining the heap. Creating the heap was just to make the array in a form of a heap so that our heap sort functions can sort this properly and the other function was making sure our heap stays in the form that we want. This was a comparatively pretty fast sorting algorithm.
4. Batcher sort:
   This is the only algorithm in this assignment that uses a sorting network. A sorting network is a circuit with the sole intention of it to sort incoming traffic, thus in this sorting routine we used operators such as bit shifts that we did not use in any of the other sorting routines.

The process, how the sorts were experimented with:
Writing these algorithms took a couple of tries and the result was not always as expected. The pseudo code was provided and I went through to see if I understood all the algorithms that I had to execute using count variables and other methods in python. First I was able to understand what the methods are doing. Here there were many print statements involved in order to see how the sorting method is working. We can then figure out how to find the moves and compare. This we just find the locations that either compares or moves are done to the array that we pass in. Then

we use the stats functions given to us and track the moves and compares of the algorithms.

Analysis of the graph:
As we see in the graph below we see the time complexity and the performance of the sorting algorithms. I put all the sorting algorithms on one graph as this way we can properly see how they compare to each other. We see that insertion sorts performance deteriorates quickly and faster than any other algorithm. Then we see that quick sort and heap sort have the same average time complexity. These two sorting routines have different optimal and least optimal time complexity, however for the sake of this assignment we are only going to consider the time complexity of only the average case. Then we see an interesting occurrence when we compare batcher sort with these two algorithms. We see that batcher sort actually performs better in cases where we are dealing with less elements, and as we start to get more elements the other two sorts will continually be better than batcher sort. This is interesting as the performance of the algorithms depends on the amount of elements and the position of these elements in the arrays. In most if not all cases either heap sort or quick sort will be more optimal to utilize over batcher sort or insertion sort. In all cases where we can observe the time complexity of a function we can immediately determine the efficiency of the function and how well it will perform. And we observe that in this assignment.

Graph:

Insertion Sort: $O(n^2)$ red
Heapsort: $O(nlog(n))$ blue
Quicksort: $O(nlog(n))$ blue
Batcher’s Odd-Even Merge Sort: $O(nlog^2(n))$ purple