Leonardo De La Torre Cruz

Professor Matta

Algorithms and Data Structures, CS 340

6 September 2023

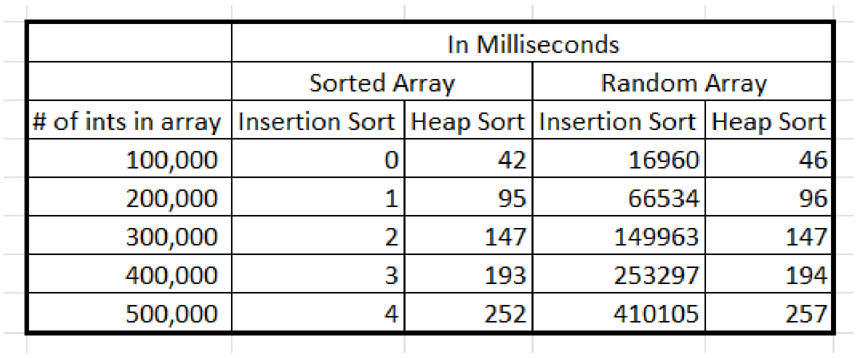
Insertion Sort Vs. Heap Sort

In this project I created two sorting algorithms of different time complexities. I made an insertion sort algorithm and a heap sort algorithm. The time complexity for heap sort is supposed to be θ (n log n), while insertion sort is n^2 in the worst case and n in the best case. I hope to see if having these two sorting algorithms sort large arrays of many sizes we will see that they follow their expected time complexities.

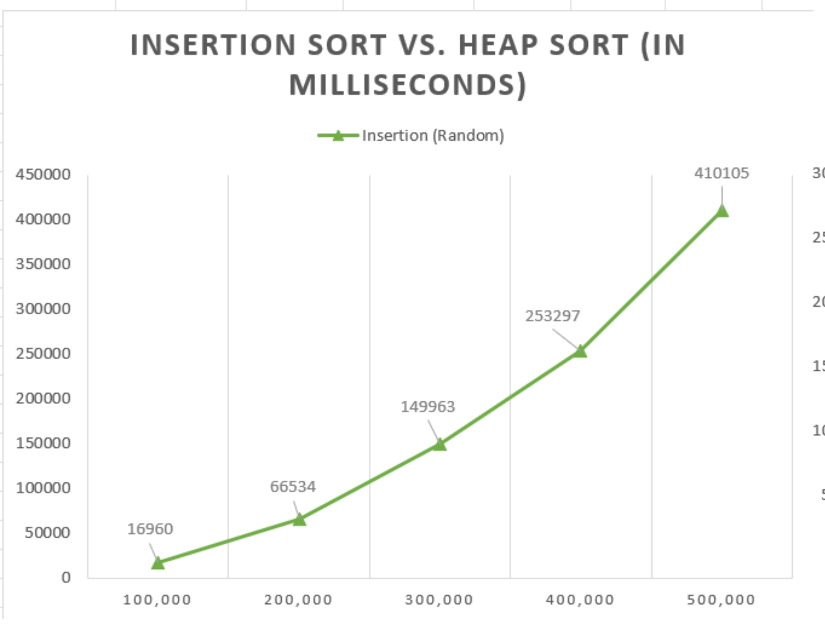
Using C++ to create the algorithms I was able to use chrono::high_resolution_clock to track the time the algorithm would take to execute their sorting. The time that is gathered in the table below is in milliseconds. I created two methods in the project.cpp file that will create vector<int> one that is filled with random numbers and one that is already sorted. The two methods will take an input from the user for the size the vectors should be by using cin. The sorting algorithms will be in their own .h file and will be created in a class. I use <fstream> to output the times of the sorting algorithms (in milliseconds) to a text file called data so I can keep the results and look at them easily.

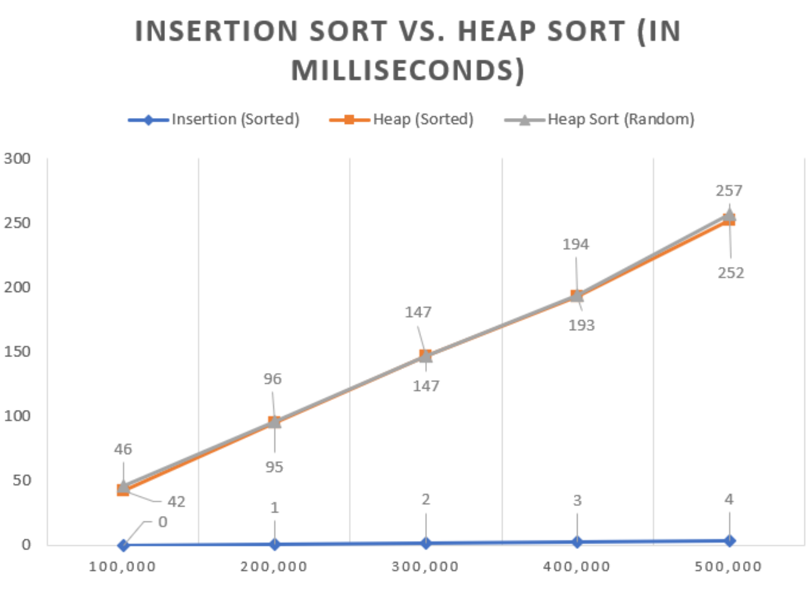
The way the program is set up is that the terminal will ask for a size for the all the vectors, it will enter a for loop and create the vectors. Two that will be randomly generated and another two that will be sorted already with the size equal to what the user inputs. As the program goes through the for loop using different sorting algorithms to sort the two different vectors the sizes of the vectors being sorted will be written to the data.txt as well as the times for how long it takes for each sorting algorithm to sort through the vector. Once we reach the end of the for loop the size variable will increase by 100,000 and the for loop will repeat again five times. So, when a user input 100,000 into the terminal the program will increase the size of the vectors. Each sorting algorithm will sort vectors of many sizes like 100,000, 200,000, 300,000, 400000, and 500,000. Once the for loop finishes it will close data.txt and finish the program.

Results:



	In Milliseconds			
	Sorted Array		Random Array	
# of ints in array	Insertion Sort	Heap Sort	Insertion Sort	Heap Sort
100,000	0	42	16960	46
200,000	1	95	66534	96
300,000	2	147	149963	147
400,000	3	193	253297	194
500,000	4	252	410105	257





Before starting this project, I already had an idea of how these sorting algorithms would run. I knew that heap sort no matter what would run n log n time complexity. Insertion sort is different because if the vector is not sorted it would have a time complexity of n^2, but when sorting and already sorted array the time complexity would change to n. So, looking at the data we can see that when heap sort sorts a random vector and sorted vector they are about the same values and same graphical shape. Which does go with our predictions that no matter the type of vector we sort it will be the same time complexity every time. The biggest change is between insertion sort for a vector unsorted and a vector sorted. When insertion sort sorts the vector that is already sorted it has a perfect linear shape on the graph. As the size of the vector increased by 100,000 the time for insertion sort to sort the already sorted vector took another millisecond. When sorting the unsorted vector, we see in the graph that insertion sort has a quadratic shape to it. As well the time that I take for sorting is significantly higher than the other cases seen on the graph. Since the unsorted insertion sort graph is quadratic shaped, we can conclude that it follows the time complexity of n^2. The total amount of time it took for the program to run is 14.9 minutes, with most of the time being when insertion sort was sorting unsorted vectors.

After looking over the results I can say confidently that the two sorting algorithms do follow their time complexities in best- and worst-case scenarios. Heap sort is the fastest in sorting the worst case by miles compared to insertion sort. However, insertion sort does have an effective use with its n time complexity with sorted arrays make it the fastest compared to heap sort which is useful when you already have a sorted array and need to add another value to it.