Ahmed Iftiak

Algorithm 22000

04/15/2022

N	Insertion sort	Merge sort	Heapsort	Quicksort	Quicksort (random)	Radix sort
10	2.75E-07	1.71E-06	1.11E-06	6.67E-07	1.10E-05	1.86E+06
100	5.95E-06	1.34E-05	1.48E-05	1.75E-05	1.01E-04	1.00E-05
1000	4.86E-04	1.46E-04	2.25E-04	1.33E-04	1.24E-03	8.09E-05
10000	3.52E-02	1.74E-03	2.56E-03	1.36E-03	1.14E-02	8.65E-04
100000	2.26E+00	1.67E-02	2.91E-02	1.25E-02	9.47E-02	6.11E-03
1000000	251.563	1.12E-01	2.95E-01	1.10E-01	9.02E-01	4.54E-02

This is the first time I have done this kind of project and every aspect was kind of new. I have not done any time analysis programming before or used sorting algorithms at this scale. There was some confusion regarding how to do this project since it's about getting only the compiler time for each sorting algorithm. After doing some research I decided to use the *Chrono high-resolution clock.* Since I am measuring the duration of algorithms. I was able to adjust the unit up to nanoseconds. Although I took two different outputs, one in nanoseconds and one in seconds, just to get the accurate result across array size and make sure that it is working right.

For the random number generator at first, I used the rand() function, while doing the research I found that the rand() function is not so uniform when given a range. So, I decided to go with one that is easy to implement, the random device, the mt19937 generator. For the range I was considering getting them as input, so I can adjust as the array size change, but it did not have any significant effect on time, so I decided to set the range from 0 to 50000.

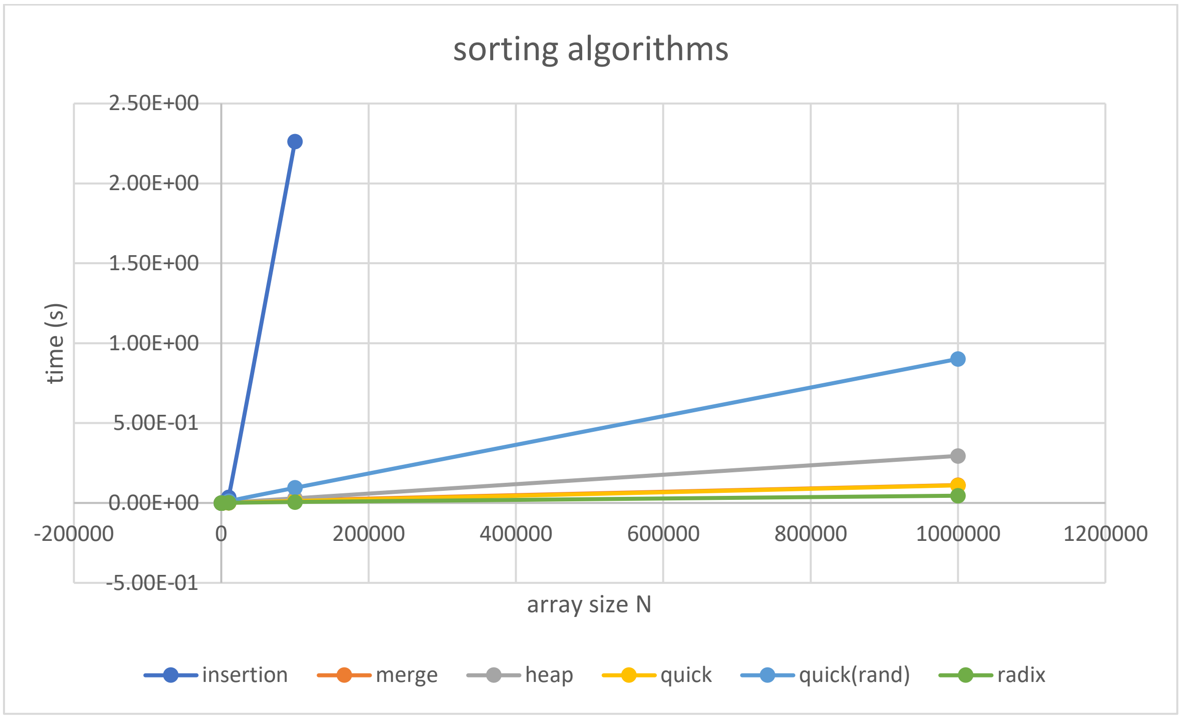
The implementation of the algorithms was not as bad as I expected. I had no problem implementing the insertion sort algorithm. It was simple and easy to follow. For merge sort, I had some trouble, first few tries implementing the pseudo-code I did not get the right result, I decided to use the while loop instead of for loop, which worked, and made the changes accordingly. I did not face much problem with the heapsort and quicksort either, it was easy to implement using the pseudo-code from the book. For radix sort, I had to do the same research to implement the pseudocode. My code ended a little differently than the pseudocode, but it works as expected.

The main part of this project was to get the compiler time of the sorting algorithms. Initially, I wrote my code on VS Code editor on windows and the compiler I used was MinGW. The driver code of each program is similar, and array size was taken as input. First, I run the programs using a test array to make sure that the program is working currently. I was getting different times with the same size array for every algorithm. The variation was not minor sometimes I was getting almost double to time, but after a couple of runs, I started to get similar results. I did change the clock function to make sure it's not the clock. The result was the same then I decided to take an average time of 5 runs. Everything worked fine but whenever I enter an array size of 1M my program would crash. I thought there might be a limit to the random generator but turns out it wasn't the problem. I tried to use an online compiler which worked but the variations were significant sometimes. The problem was solved after changing the operating system. I did not have Linux previously so, I had to set up Linux on Virtual Box. Everything worked perfectly. The variation in time was also minor.

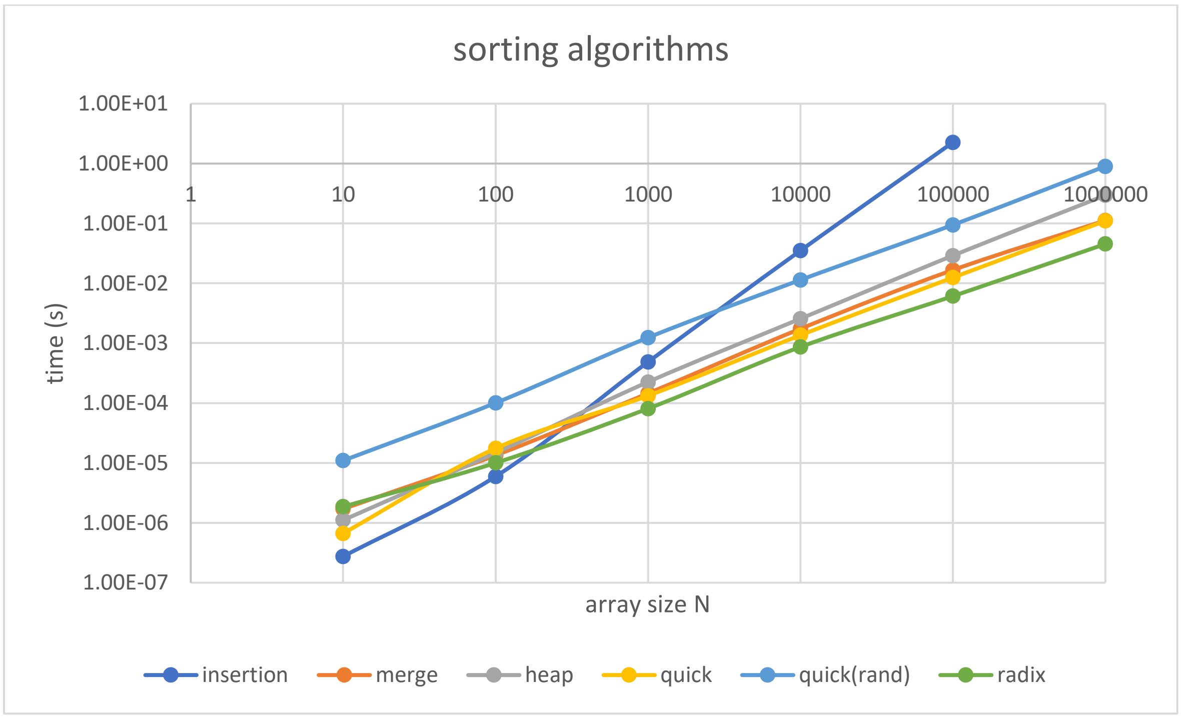
As for time complexity, insertion sort was vastly different than other sorting algorithms, although it took less time to sort array sizes up to 1k but after that difference was multiple orders of magnitude. First few runs I had to quit the program because it was taking so long, and I thought my system froze. As with other algorithms, the difference was not that different, as we can see in the graphs below. They follow the same trend as the theoretical time complexity. radix sort being the fastest. Time for Quicksort Random was unique. Each run time differs with higher variation than other algorithms. This is expected as each run the pivot was randomized so depending on the location of the pivot time differs.

I attached the screenshot of the problem I was having with size 1M and graphs of the above data and a table for the first attempt in windows.

Normal scale



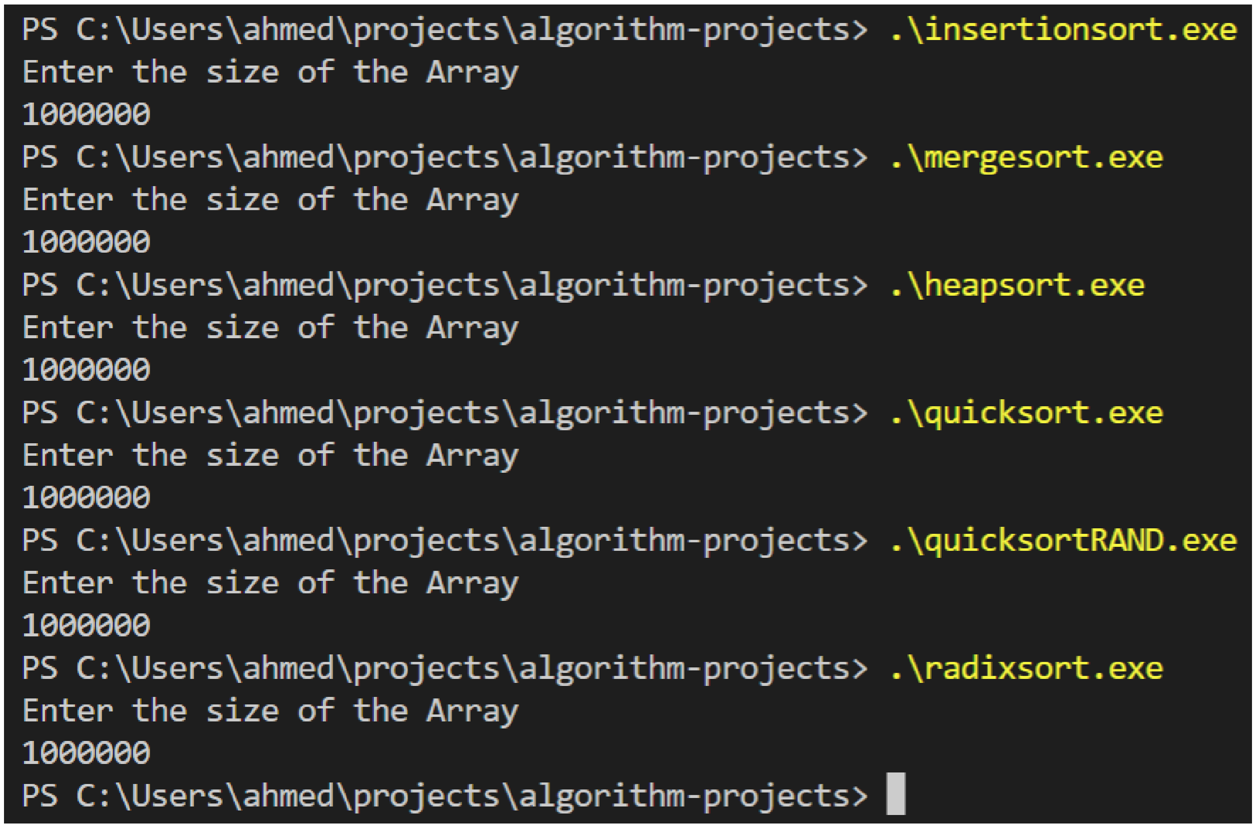
Logarithmic scale



First attempt

N	Insertion sort	Merge sort	Heapsort	Quicksort	Quicksort (random)	Radix sort
10	4.40E-07	1.70E-06	1.10E-06	7.00E-07	4.00E-06	1.86E-06
100	6.96E-06	1.29E-05	1.29E-05	6.70E-06	1.29E-05	9.34E-06
1000	4.65E-04	1.32E-04	1.69E-04	9.34E-05	1.18E-04	8.66E-05
10000	3.60E-02	1.64E-03	2.39E-03	1.16E-03	1.45E-03	7.44E-04
100000	2.13E+00	1.52E-02	2.22E-02	1.14E-02	1.48E-02	6.70E-03
500000	54.8605		9.81E-02	5.08E-02	6.00E-02	
1000000						

Error from the complier



```
PS C:\Users\ahmed\projects\algorithm-projects> .\insertionsort.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects> .\mergesort.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects> .\heapsort.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects> .\quicksort.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects> .\quicksortRAND.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects> .\radixsort.exe
Enter the size of the Array
1000000
PS C:\Users\ahmed\projects\algorithm-projects>
```