Runtime of the sorting algorithms (A.D project)

Name: SIDDHARTH SAURABH

Aim	To compute the runtime of the basic algorithms of Sorting i.e Merge-sort, Quick-sort and Heap-sort using C programming language.
Resources. used	Vim editor, knowledge of C programming, Excel sheet.

Abstract

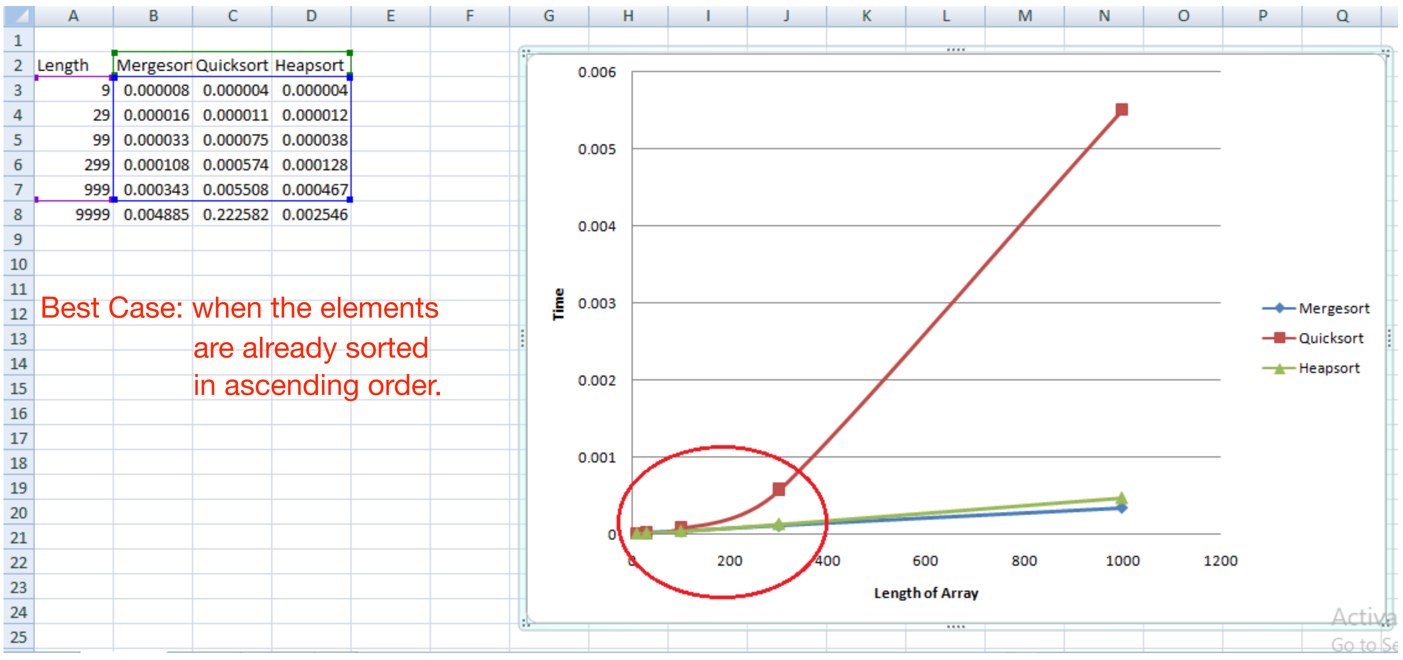
The program is built using C programming . This program includes the concepts of different ways of sorting. The header files included in the project are <stdio.h>, <time.h> and <stdlib.h>. The <time.h> header file is used for including time functions to compare the time taken for the execution of each algorithm. This time taken for each algorithm is then compared with each other and thus the result is generated.
The program starts by clearing the terminal screen using the command "system("clear");" , then the user is asked to enter the size of the array. After the size is entered , the program has two options you can go for, first is to ask the user to input the number of elements required to fill up the array to be sorted or else one can select the range from 0 to the size of the array to be sorted. The program then calls the Merge-sort, Quick-sort and Heap-sort functions. These functions have embedded functions to calculate the start time and the termination time for each of the sorting to take place.the start time and end time are calculated using the <time.h> header file. Then the total time of execution is calculated by the difference of start time and end time and dividing the difference with number of clocks per second. The time taken by each algorithm for sorting is then compared. The one taking the least time for execution is the fastest.

Limitations

The project is built with a limitation of inputting the array size upto which the system can hold. If a very large value for size of array is given then the program crashes. Also the time difference is very less and thus very hard to plot

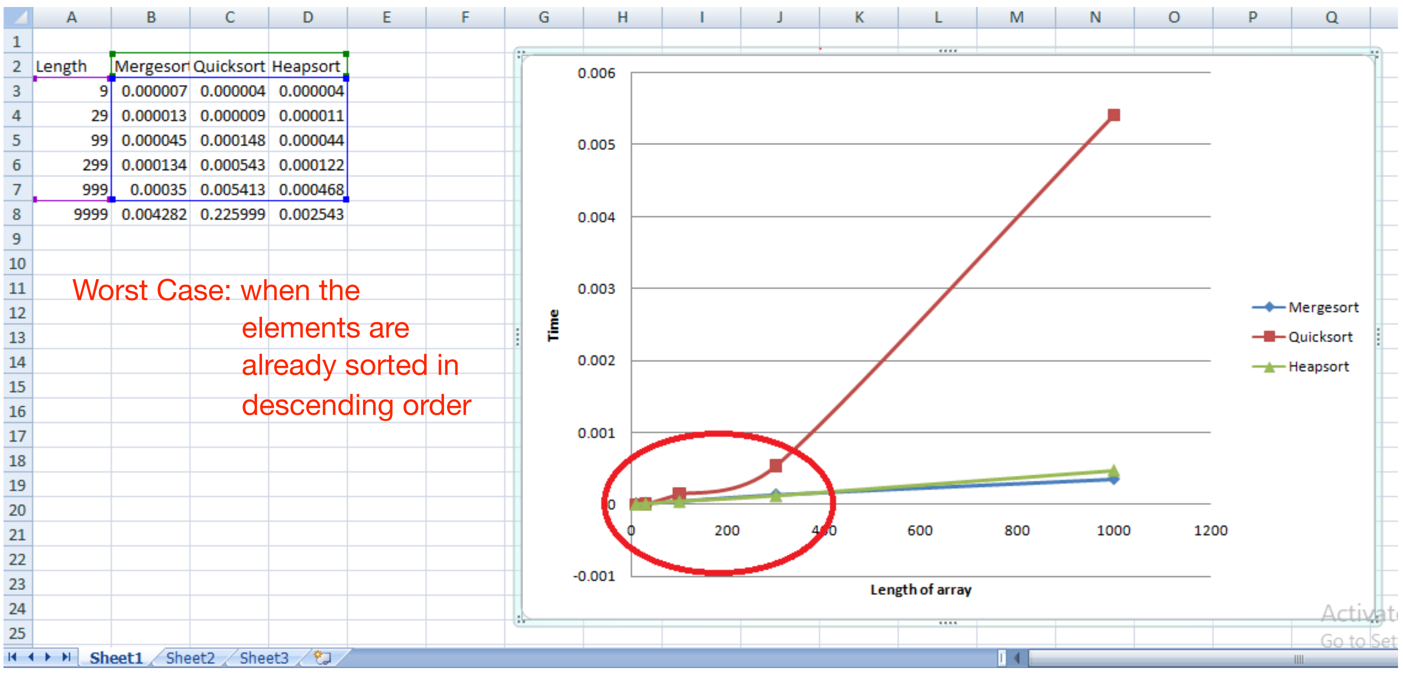
Result

Although the time complexity matters a lot for calculating the efficiency of an algorithm but it does not have a very strong relation with the runtime. Runtime for all the above mentioned algorithms are mentioned below with their graphs.
The first image used is for calculating the runtime of the different sizes of arrays with elements in sorted order from 0 to the required number n.
The second image used is for calculating the time taken to sort an array if the elements are entered in the descending to ascending order from n to 0.



Length	Mergesort	Quicksort	Heapsort
9	0.000008	0.000004	0.000004
29	0.000016	0.000011	0.000012
99	0.000033	0.000075	0.000038
299	0.000108	0.000574	0.000128
999	0.000343	0.005508	0.000467
9999	0.004885	0.222582	0.002546

Best Case: when the elements are already sorted in ascending order.



Length	Mergesort	Quicksort	Heapsort
9	0.000007	0.000004	0.000004
29	0.000013	0.000009	0.000011
99	0.000045	0.000148	0.000044
299	0.000134	0.000543	0.000122
999	0.00035	0.005413	0.000468
9999	0.004282	0.225999	0.002543

Worst Case: when the elements are already sorted in descending order

Conclusion

From the above project we can conclude that the time complexity might matter a lot in worst case as well as in best case but the runtime of all the three basic sorting algorithms almost behaves similar in its worst as well as best case. This is because of when we calculate the time complexity we neglect the constant factors which are associated with the order of time time complexity but when we calculate the total runtime, all factors come in play including those which are neglected while calculating time complexity.