Algorithm: Assignment 1:
Name: Abhishek Sharma
Professor: Mohan Kumar

Analysis For MY_CHOICE_QSORT and Quicksort :

The original Quicksort was developed by Hoare in 1961 . In his original algorithm the pivot is been chosen randomly . This algorithm has the time complexity of O(nlogn) but it yields to worst case when the data are already shorted or sorted in reverse order where it takes nearly $O(n^2)$. There is variation of this algorithm as well in which the pivot is taken from the beginning or from the end side. This algorithm work quite fast but still has the same worse scenario of $O(n^2)$.

There is a improvisation for this in which the pivot is chosen from the middle of the list which gives O(nlogn) and doesn't have the problem of the above mentioned worst case as the list will always be divided into equal parts. In my program I used these method and checked against the original quicksort method.

Below are the table containing the generated dataset which can provide the running time comparison between the 4 sorting algorithm as required explanation.

Note: The memory usages are the total memory usages including the generation of dataset
but we can use it for relative comparison between the algorithms as the memory usages for datasets are approximately remains the same for all the program for a given data range.
In below table the calculation for memory usages is respective to each different data set
But at run time each sorting algorithm will give his total memory usages cumulative of all the data set.
For 1 million records the available CPU resource was slow so it's been not included in the dataset as desired.

Reference: http://www.liacs.nl/~graaf/STUDENTENSEMINARIUM/quicksorthistorical.pdf

For Random Values

	Memory Usages	CPU Usages
For 1000		
MY_CHOICE_QSORT	658144	3
Quicksort	987240	4
Mergesort	658168	5
Heapsort	658200	4
For 10,000		
MY_CHOICE_QSORT	658168	8
Quicksort	1316232	14
Mergesort	1700240	31
Heapsort	658200	13
For 50,000		
MY_CHOICE_QSORT	1994960	20
Quicksort	1994560	29
Mergesort	6877688	46
Heapsort	1994704	33
For 1,00000		
MY_CHOICE_QSORT	3391528	30
Quicksort	3412224	50
Mergesort	13762888	69
Heapsort	3361608	61

For Sorted Values

1000		
MY_CHOICE_QSORT	658168	2
Quicksort	658200	4
Mergesort	658168	3
Heapsort	658200	4
10,000		
MY_CHOICE_QSORT	658168	10
Quicksort	988408	16
Mergesort	1700240	27
Heapsort	658200	13
50,000		
MY_CHOICE_QSORT	2036800	17
Quicksort	1999752	30
Mergesort	6877352	39
Heapsort	1881536	24
1,00000		
MY_CHOICE_QSORT	3395720	22
Quicksort	3411384	42
Mergesort	13700728	45
Heapsort	3433200	43
1000		

For

One-forth Sorted Data

MY_CHOICE_QSORT	618144	3
Quicksort	987280	4
Mergesort	1019416	4
Heapsort	987208	5
10,000		
MY_CHOICE_QSORT	658144	10
Quicksort	3878240	15
Mergesort	4865208	25
Heapsort	3878104	12
50,000		
MY_CHOICE_QSORT	2031680	22
Quicksort	6660120	31
Mergesort	12485528	49
Heapsort	6655744	39
1,00000		
MY_CHOICE_QSORT	3361216	37
Quicksort	16057272	58
Mergesort	24441416	75
Heapsort	14042264	55

For

Possion Distribution

Number

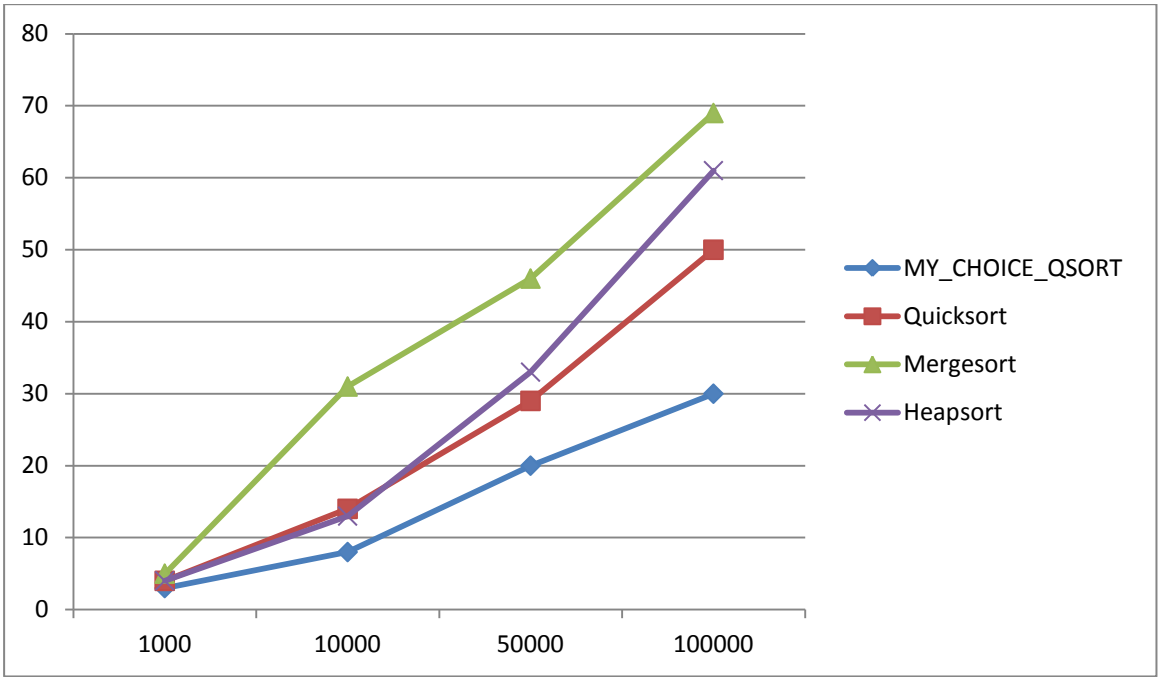
1000		
MY_CHOICE_QSORT	10609064	3
Quicksort	10588528	3
Mergesort	10921672	4
Heapsort	11530560	5
10,000		
MY_CHOICE_QSORT	482312448	43
Quicksort	16536864	49
Mergesort	16487704	50
Heapsort	16487744	43
50,000		
MY_CHOICE_QSORT	206127016	81
Quicksort	206167984	80
Mergesort	2061639361	42
Heapsort	206177576	32
1,00000		
MY_CHOICE_QSORT	31339088	280
Quicksort	33703648	283
Mergesort	41923424	56
Heapsort	33713024	48

Comparison from Graph:

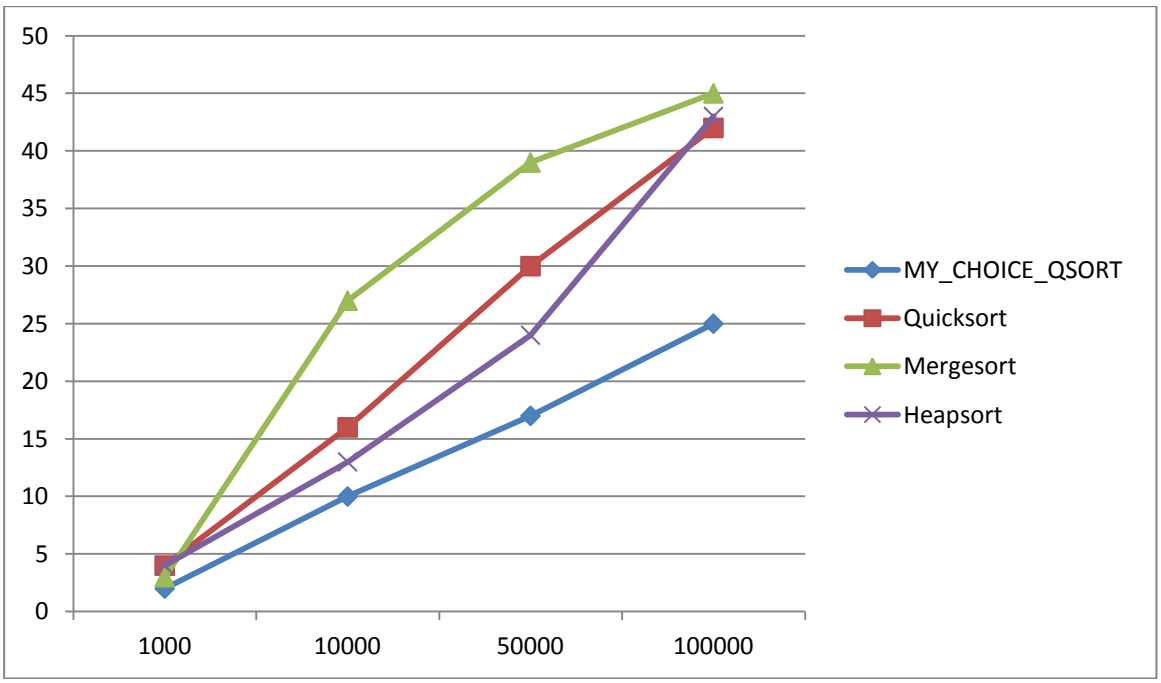
As we can see from the below graphs MY_CHOICE_QSORT is efficient in three out of four kind

Of data sets but it's not efficient for the data set where all the elements have very less difference among each other and the frequency of a particular number is high as compare to other elements in the set (in our example: poisson distribution data set).

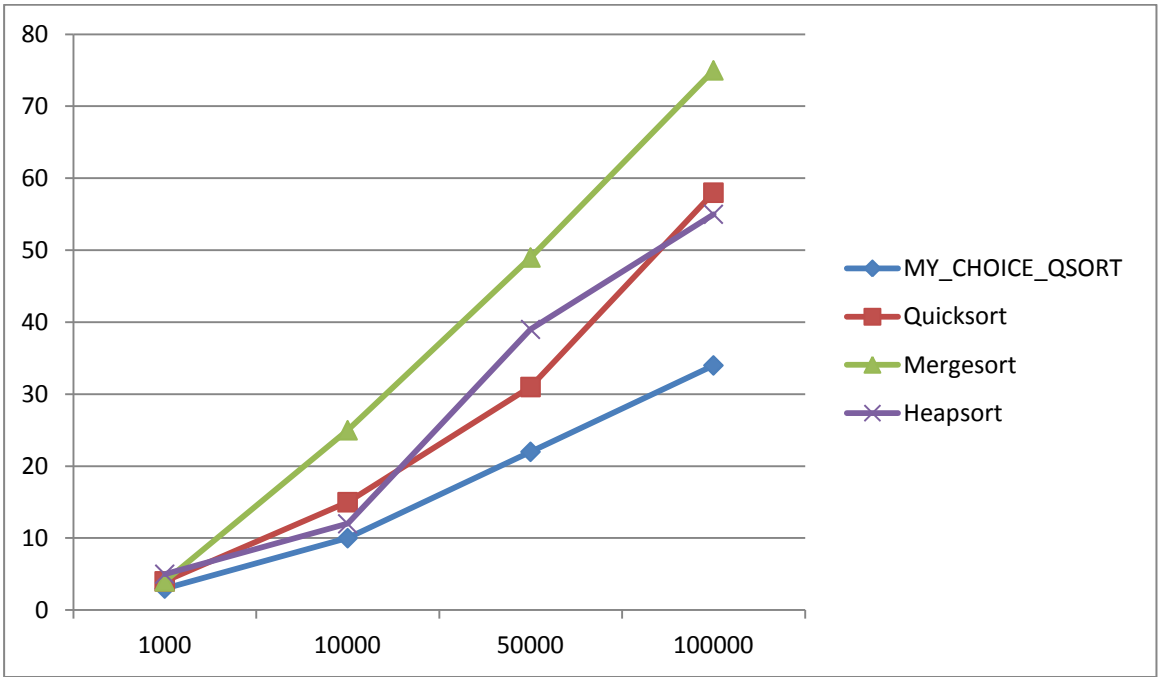
For Random Values:



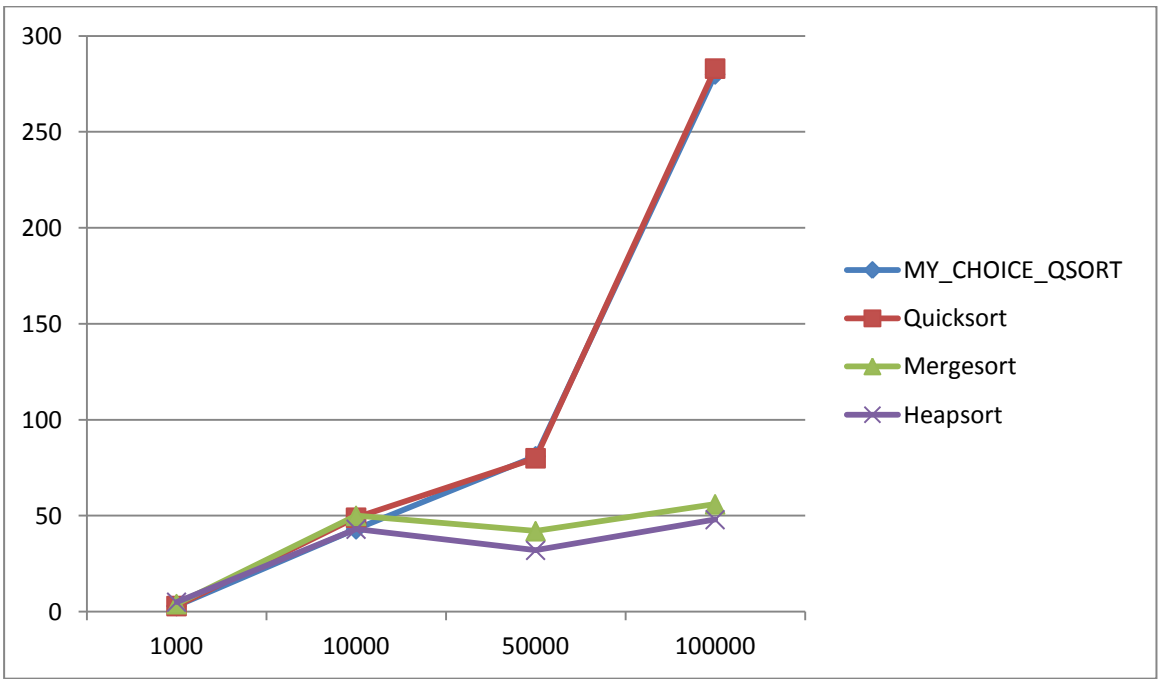
For Sorted Data:



For One-Forth sorted Data:



For Poisson Distributed Data:



Partition comparision between MY_CHOICE_QSORT and Quicksort:

Below table provides the comparison:

Data Sets	**Quicksort**	**MY_CHOICE_QSORT**
For random values		
1000	667	654
10000	6666	6696
50000	33303	33441
100000	66671	66588
For sorted values		
1000	633	512
10000	6324	5905
50000	31642	32768
100000	63169	65536
For $1/4^{th}$ Sorted Values		
1000	672	654
10000	6711	6658
50000	33373	33243
100000	66712	66623
For Poisson Distributed		
1000	900	899
10000	9860	9865
50000	49838	49844
100000	99833	99830