Gokul Manohar
CHN18CS045
Roll no. CS18D26

Assignment No. 1

Implement **quicksort** as we discussed in class for large inputs. Feed it with large sets of random numbers and worst case scenario numbers. Tabulate the running times and make relevant inference. Could you see any pattern resembling the theoretical complexity values?

Code

```
// Written by Gokul Manohar (CHN18CS045)
// Roll No. CS18D26
// Quick Sort

// Two files: a1.cs18d26.average_case.txt &
a1.cs18d26.worst_case.txt
// Supports both command line arguments and choice based inputs

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// The size of array has to be changed in accordance with the
number of elements in the file
// Otherwise the sorting algorithm takes too long to iterate
through unnecessary elements
int numberArray[250000];

int partition(int a[], int p, int r)
{
 int t = a[p];
 int q = p;
 for (int i = p + 1; i <= r; i++)
   if (a[i] < t)
   {
     a[q++] = a[i];
     if (q < i)
```

```
       a[i] = a[q];
   }
 a[q] = t;
 return q;
}

void quickSort(int a[], int p, int r)
{
 if (p < r)
 {
   int q = partition(a, p, r);
   quickSort(a, p, q - 1);
   quickSort(a, q + 1, r);
 }
}

void display(int a[], int start, int end, char *msg)
{
 printf("\n%s", msg);
 for (int i = start; i < end; i++)
 {
   if (a[i] != -1)
   {
     printf("%d, ", a[i]);
   }
 }
 printf("....\n\n");
}

// Main
int main(int argc, char *argv[])
{
 // Calculating the length of the array
 int length = sizeof(numberArray) / sizeof(numberArray[0]);
```

```c
// To avoid printing leading/trailing zeros if
// array size and elements in files doesn't match
for (int x = 0; x < length; x++)
{
  numberArray[x] = -1;
}
int choice;
char *filename = malloc(30);
if (argc >= 2) // Filename is given as command line argument
{
  strcpy(filename, argv[1]);
}
else // Filename not given as command line argument
{
  printf("1: Average Case Numbers (Random Numbers)\n2: Worst Case Numbers (Sorted Numbers)\nChoice: ");
  scanf("%d", &choice);
  if (choice == 1)
  {
    strcpy(filename, "a1.cs18d26.average_case.txt");
  }
  else
  {
    strcpy(filename, "a1.cs18d26.worst_case.txt");
  }
}
// Reading from file
printf("\nReading from %s\n", filename);
FILE *file = fopen(filename, "r");
free(filename);
int i;
for (i = 0; i < length; i++)
{
  fscanf(file, "%d", &numberArray[i]);
}
fclose(file);
```

```
 printf("Read complete\n");
 printf("\nLength of the array: %d\n", length);

 // Displaying elements of array
 display(numberArray, 0, 15, "Array before sorting: ");

 // Quick Sort
 printf("Sorting started\n");
 quickSort(numberArray, 0, length - 1);
 printf("Sorting complete\n");

 // Displaying elements of array
 display(numberArray, 0, 15, "Array after sorting: ");
 return 0;
}
```

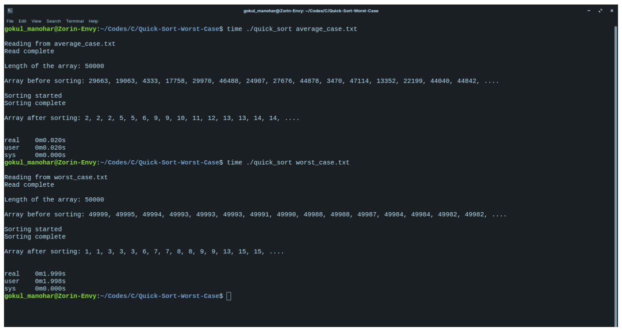
Run

On Linux terminal, run this command for determining the practical time complexity.

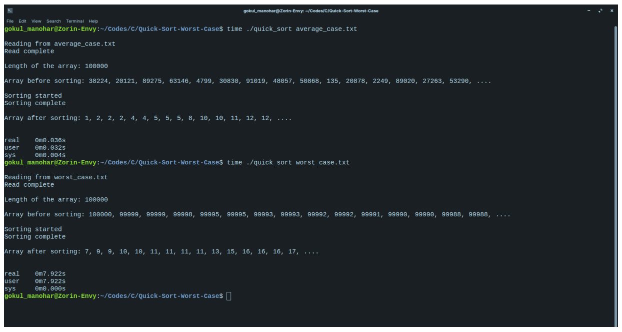
```
time ./a1.cs18d26 a1.cs18d26.average_case.txt
```

```
time ./a1.cs18d26 a1.cs18d26.worst_case.txt
```

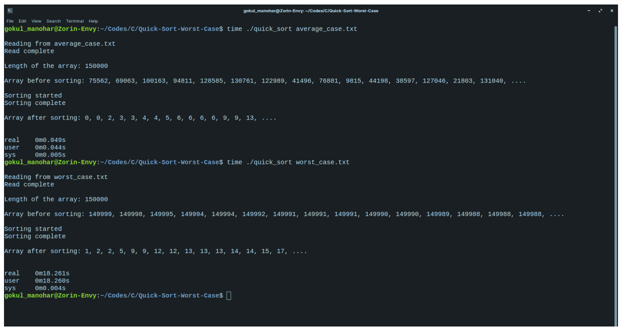
Output (Run #1)



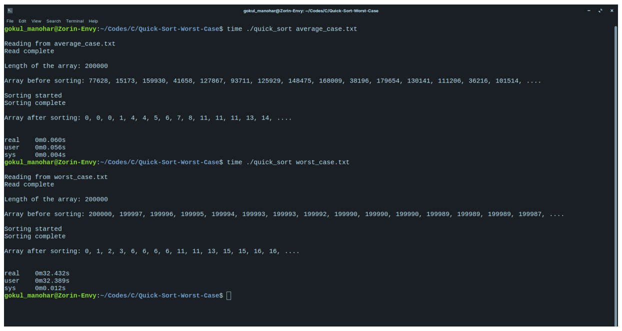
```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 50000

Array before sorting: 29663, 19063, 4333, 17758, 29970, 46488, 24907, 27676, 44878, 3470, 47114, 13352, 22199, 44040, 44842, ....

Sorting started
Sorting complete

Array after sorting: 2, 2, 2, 5, 5, 6, 9, 9, 10, 11, 12, 13, 13, 14, 14, ....


real    0m0.020s
user    0m0.020s
sys     0m0.000s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 50000

Array before sorting: 49999, 49995, 49994, 49993, 49993, 49993, 49991, 49990, 49988, 49988, 49987, 49984, 49984, 49982, 49982, ....

Sorting started
Sorting complete

Array after sorting: 1, 1, 3, 3, 3, 6, 7, 7, 8, 8, 9, 9, 13, 15, 15, ....


real    0m1.999s
user    0m1.998s
sys     0m0.000s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```



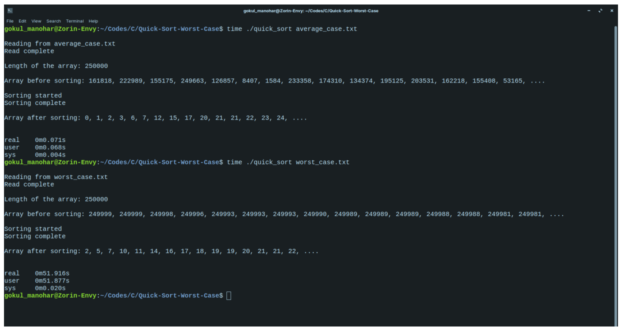
```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 100000

Array before sorting: 38224, 20121, 89275, 63146, 4799, 30830, 91019, 48057, 50868, 135, 20878, 2249, 89020, 27263, 53290, ....

Sorting started
Sorting complete

Array after sorting: 1, 2, 2, 2, 4, 4, 5, 5, 5, 8, 10, 10, 11, 12, 12, ....


real    0m0.036s
user    0m0.032s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 100000

Array before sorting: 100000, 99999, 99999, 99998, 99995, 99995, 99993, 99993, 99992, 99992, 99991, 99990, 99990, 99988, 99988, ....

Sorting started
Sorting complete

Array after sorting: 7, 9, 9, 10, 10, 11, 11, 11, 11, 13, 15, 16, 16, 16, 17, ....


real    0m7.922s
user    0m7.922s
sys     0m0.000s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```



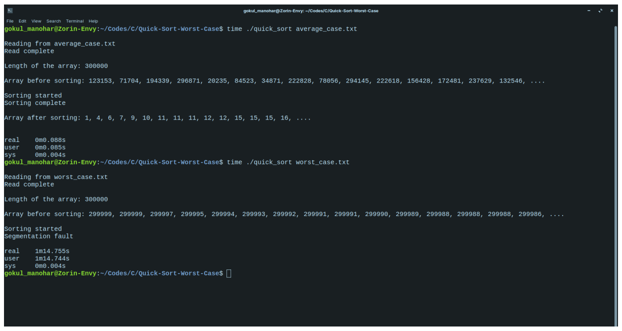
```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 150000

Array before sorting: 75562, 69063, 100163, 94811, 128585, 130761, 122989, 41496, 76881, 9815, 44198, 38597, 127046, 21803, 131040, ....

Sorting started
Sorting complete

Array after sorting: 0, 0, 2, 3, 3, 4, 4, 5, 6, 6, 6, 6, 9, 9, 13, ....


real    0m0.049s
user    0m0.044s
sys     0m0.005s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 150000

Array before sorting: 149999, 149998, 149995, 149994, 149994, 149992, 149991, 149991, 149991, 149990, 149990, 149989, 149988, 149988, 149988, ....

Sorting started
Sorting complete

Array after sorting: 1, 2, 2, 5, 9, 9, 12, 12, 13, 13, 13, 14, 14, 15, 17, ....


real    0m18.261s
user    0m18.260s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```



```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 200000

Array before sorting: 77628, 15173, 159930, 41658, 127867, 93711, 125929, 148475, 168009, 38196, 179654, 130141, 111206, 36216, 101514, ....

Sorting started
Sorting complete

Array after sorting: 0, 0, 0, 1, 4, 4, 5, 6, 7, 8, 11, 11, 11, 13, 14, ....


real    0m0.060s
user    0m0.056s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 200000

Array before sorting: 200000, 199997, 199996, 199995, 199994, 199993, 199993, 199992, 199990, 199990, 199990, 199989, 199989, 199989, 199987, ....

Sorting started
Sorting complete

Array after sorting: 0, 1, 2, 3, 6, 6, 6, 6, 11, 11, 13, 15, 15, 16, 16, ....


real    0m32.432s
user    0m32.389s
sys     0m0.012s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```



```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 250000

Array before sorting: 161818, 222989, 155175, 249663, 126857, 8407, 1584, 233358, 174310, 134374, 195125, 203531, 162218, 155408, 53165, ....

Sorting started
Sorting complete

Array after sorting: 0, 1, 2, 3, 6, 7, 12, 15, 17, 20, 21, 21, 22, 23, 24, ....


real    0m0.071s
user    0m0.068s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 250000

Array before sorting: 249999, 249999, 249998, 249996, 249993, 249993, 249993, 249990, 249989, 249989, 249989, 249988, 249988, 249981, 249981, ....

Sorting started
Sorting complete

Array after sorting: 2, 5, 7, 10, 11, 14, 16, 17, 18, 19, 19, 20, 21, 21, 22, ....


real    0m51.916s
user    0m51.877s
sys     0m0.020s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```



```
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort average_case.txt

Reading from average_case.txt
Read complete

Length of the array: 300000

Array before sorting: 123153, 71704, 194339, 296871, 20235, 84523, 34871, 222828, 78056, 294145, 222618, 156428, 172481, 237629, 132546, ....

Sorting started
Sorting complete

Array after sorting: 1, 4, 6, 7, 9, 10, 11, 11, 11, 12, 12, 15, 15, 15, 16, ....


real    0m0.088s
user    0m0.085s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ time ./quick_sort worst_case.txt

Reading from worst_case.txt
Read complete

Length of the array: 300000

Array before sorting: 299999, 299999, 299997, 299995, 299994, 299993, 299992, 299991, 299991, 299990, 299989, 299988, 299988, 299988, 299986, ....

Sorting started
Segmentation fault

real    1m14.755s
user    1m14.744s
sys     0m0.004s
gokul_manohar@Zorin-Envy:~/Codes/C/Quick-Sort-Worst-Case$ 
```

txt files

Here I used the included python file ‘a1.cs18d26.number_generator.py’ to generate two txt files. One file ‘a1.cs18d26.average_case.txt’ filled with random numbers and another file ‘a1.cs18d26.worst_case.txt’ filled with sorted elements.

Observations

n	Time Complexity (seconds)			
	Average Case (random numbers)		Worst Case (sorted numbers)	
	Run #1	Run #2	Run #1	Run #2
50,000	0.020	0.021	1.999	2.016
1,00,000	0.034	0.041	7.922	8.928
1,50,000	0.049	0.051	18.261	18.813
2,00,000	0.060	0.062	32.432	34.139
2,50,000	0.071	0.073	51.916	52.103
3,00,000	0.088	0.089	segmentation fault	segmentation fault

I also tested the algorithm with n = 50,00,00 (5 million) random elements in the average case scenario.

n	Time Complexity (seconds)	
	Average Case (random numbers)	
	Run #1	Run #2
50,00,000	1.298	1.303

Experimental Inference

- Having random elements in the arrays makes the sorting process much faster than having completely sorted arrays.
- Higher numbers of sorted elements in the array makes the sorting process more complex for the algorithm.
- Even for very large values of n (random numbers), the algorithm can very easily sort through the elements. This is evident in the n = 5 million (random numbers) case.
- It is found that doubling the input almost quadruples the time in worst case scenarios.

Theoretical Values

For Quick sort according to Big O notation

- Average case has a time complexity of $\boldsymbol{O(nlogn)}$**.**
- Worst case has a time complexity of $\boldsymbol{O(n^2)}$**.**

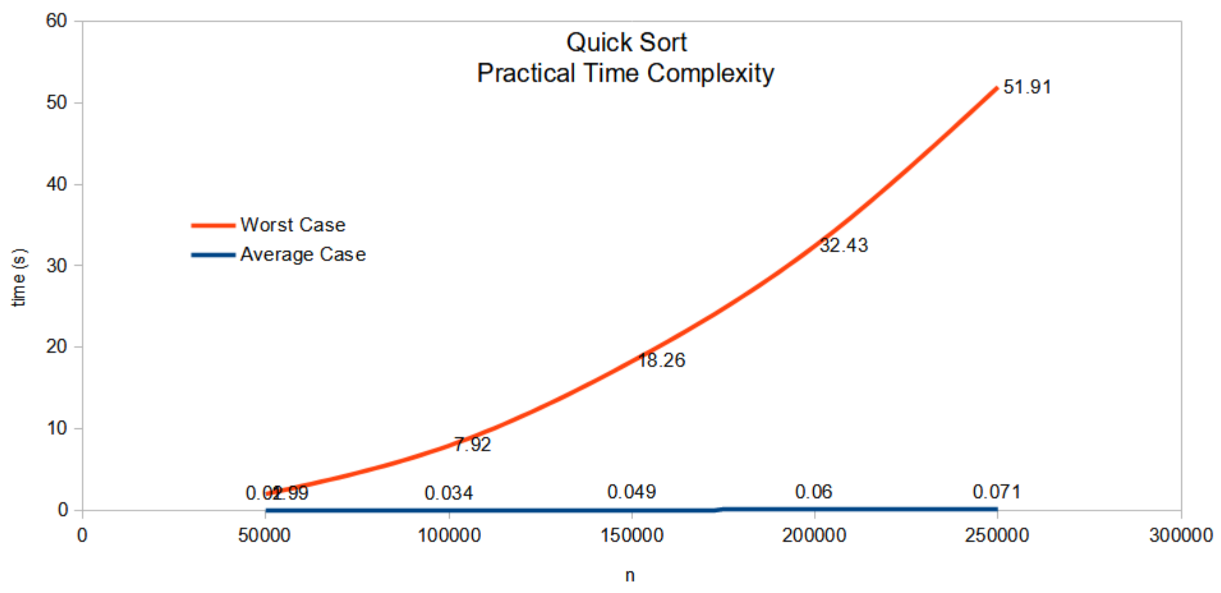
So in worst case for

n = 0.8 million, the system performs $6.4*10^{11}$ operations.

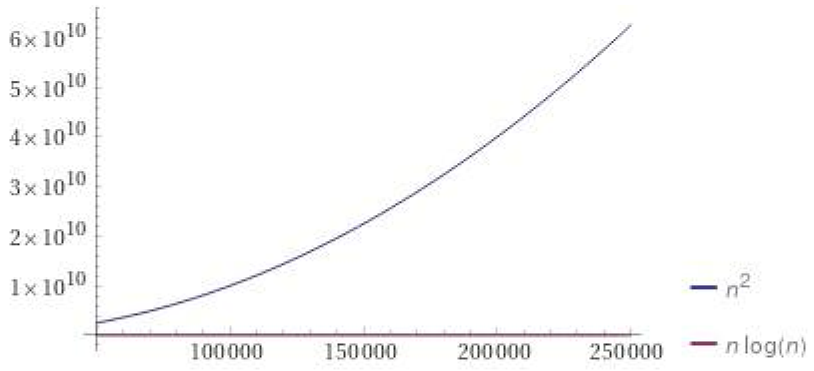
n = 1.6 million, the system performs $25.6*10^{11}$ operations.

This means doubling the input actually quadruples the number of operations. Thus the time it takes for doing 4 times the work makes the experimental time complexity to roughly quadruple.

If a graph is plotted from the experimental time complexity values (Run #1) given in the table above



And compare it to the mathematical plotting of n^2 and ***nlogn***



Source: wolframalpha.com
https://www.wolframalpha.com/input/?i=plot+n%5E2+and+n*log+n+where+n+from+50000+to+250000

It is evidently clear that the experimental values are very close to the theoretical and mathematical values.

Conclusion

The main advantage of using Quick sort is that in normal cases most of the inputs fall into the average case scenario and for that the Quick sort algorithm is extremely fast ***O(nlogn)***.

But the main disadvantage of quicksort is that a bad choice of pivot element and pre-sorting can decrease the time complexity of the algorithm down to $O(n^2)$.