Design and Analysis of Algorithms

Experiment 1

Palaash Jain, 2021300050, Comps A(Batch – C)

Date of performance: 13th February, 2022

Date of submission: 19th February, 2022

AIM: To calculate running time for merge and quick sort for different number of elements (100 – 100000) and compare the efficiency of both with respect to the running time.

CODE:

1) **To generate random numbers:**

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

int main()
{
        int count = 100000;
        FILE* fp;
        fp = fopen("Random Numbers.txt", "w");
        while (count > 0)
        {
                fprintf(fp, "%d\n", rand());
                --count;
        }
        fclose(fp);
        return 0;
}
```

2) **Sorting, running time calculation code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void swap(int* a, int* b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void conquer(int arr[], int l_start, int r_end)
{
        int mid = l_start + (r_end - l_start) / 2;
        int left = l_start;
```

```
	int right = mid + 1;
	int sorted[r_end - l_start + 1];
	int index = 0;
	while (left <= mid && right <= r_end)
	{
		if (arr[right] < arr[left])
			sorted[index++] = arr[right++];
		else
			sorted[index++] = arr[left++];
	}
	while (left <= mid)
		sorted[index++] = arr[left++];
	while (right <= r_end)
		sorted[index++] = arr[right++];
	index = 0;
	while (l_start <= r_end)
		arr[l_start++] = sorted[index++];
}

void mergesort(int arr[], int left, int right)
{
	if (left == right)
		return;
	int mid = left + (right - left) / 2;
	mergesort(arr, left, mid);
	mergesort(arr, mid + 1, right);
	conquer(arr, left, right);
}

int partition(int arr[], int low, int high)
{
	int pivot = arr[high];
	int i = (low - 1);
    int j;
	for (j = low; j <= high - 1; j++) {
		if (arr[j] <= pivot) {
			i++;
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[high]);
	return (i + 1);
}
```

```
void quicksort(int arr[], int low, int high)
{
        if (low < high) {
                int pi = partition(arr, low, high);
                quicksort(arr, low, pi - 1);
                quicksort(arr, pi + 1, high);
        }
}

int main()
{
   FILE* fp_rt;
   fp_rt = fopen("Running Times2.txt", "w");
   clock_t c_start, c_end;
   c_start = clock();
   int blocks = 1;
   int num_ele = 100;
   fprintf(fp_rt, "Block No.\t\tNo.Elements\t\tQuick Sort\t\tMerge Sort\n");
   while (blocks <= 1000)
   {
      clock_t i_start, i_end;
      clock_t s_start, s_end;
      int arr[num_ele];
      int arr2[num_ele];
      FILE* fp;
      fp = fopen("Random Numbers.txt", "r");
      for (int i = 0; i < num_ele; ++i)
      {
         fscanf(fp, "%d", &arr[i]);
         arr2[i] = arr[i];
      }
      s_start = clock();
      mergesort(arr, 0, num_ele - 1);
      s_end = clock();
      i_start = clock();
      quicksort(arr2, 0, num_ele - 1);
      i_end = clock();
      fprintf(fp_rt, "%d\t\t\t%d\t\t\t\t%lf\t\t\t\t%lf\n", blocks, num_ele, ((double)i_end -
(double)i_start) / CLOCKS_PER_SEC, ((double)s_end - (double)s_start) /
CLOCKS_PER_SEC);
      fclose(fp);
      num_ele += 100;
      ++blocks;
   }
```

```
    c_end = clock();
    printf("\nCode Running Time: %lf\n", ((double)c_end - (double)c_start) /
CLOCKS_PER_SEC);
    fclose(fp_rt);
        return 0;
}
```

<u>OUTPUT/CONCLUSION:</u>

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1265]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pcm20\Documents\DAA\Experiment 1>gcc sort2.c

C:\Users\pcm20\Documents\DAA\Experiment 1>a

Code Running Time: 16.449000

C:\Users\pcm20\Documents\DAA\Experiment 1>
```

Running Times2 - Notepad

File Edit View

Block No.	No.Elements	Quick Sort	Merge Sort
1	100	0.000000	0.000000
2	200	0.000000	0.000000
3	300	0.000000	0.000000
4	400	0.000000	0.000000
5	500	0.000000	0.000000
6	600	0.000000	0.000000
7	700	0.000000	0.000000
8	800	0.000000	0.000000
9	900	0.000000	0.000000
10	1000	0.000000	0.000000
11	1100	0.000000	0.001000
12	1200	0.000000	0.000000
13	1300	0.000000	0.000000
14	1400	0.000000	0.001000
15	1500	0.000000	0.000000
16	1600	0.000000	0.000000
17	1700	0.000000	0.000000
18	1800	0.000000	0.000000
19	1900	0.000000	0.000000
20	2000	0.001000	0.000000
21	2100	0.001000	0.000000
22	2200	0.000000	0.000000
23	2300	0.000000	0.000000
24	2400	0.001000	0.000000
25	2500	0.000000	0.000000
26	2600	0.000000	0.001000
27	2700	0.001000	0.000000

Random Numbers - Notepad

File Edit View

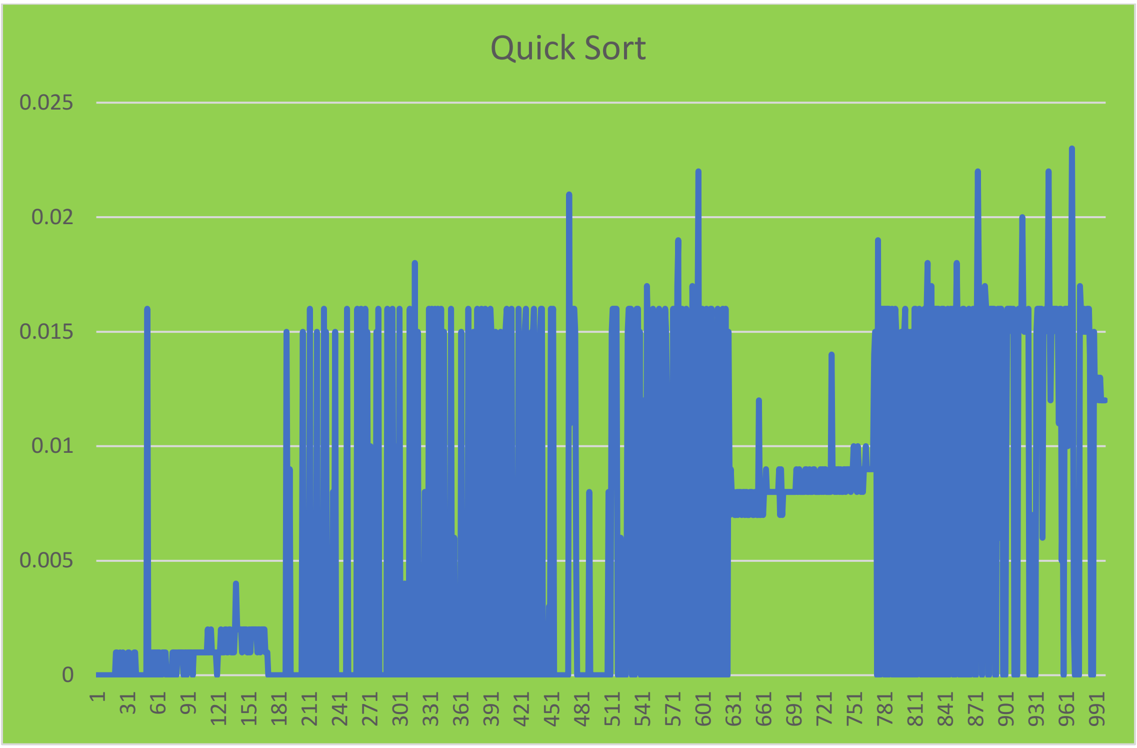
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
28145
23281
16827
9961
491
2995
11942
4827
5436
32391
14604
3902
153
292
12382
17421
18716

Random Numbers2 - Notepad

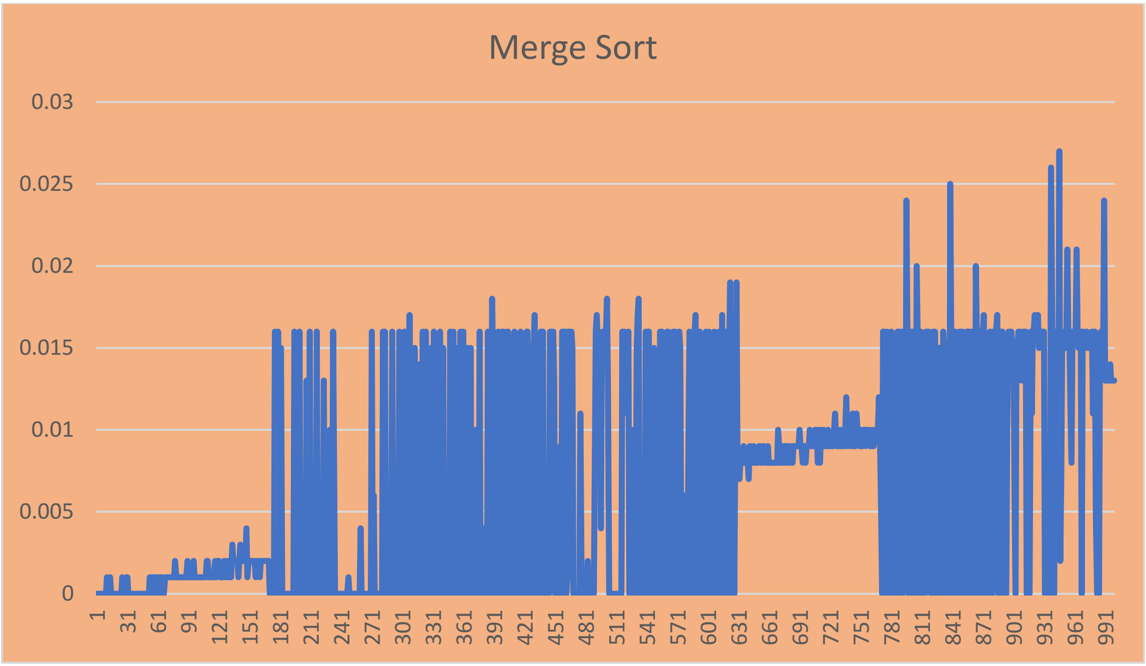
File Edit View

1804289383
846930886
1681692777
1714636915
1957747793
424238335
719885386
1649760492
596516649
1189641421
1025202362
1350490027
783368690
1102520059
2044897763
1967513926
1365180540
1540383426
304089172
1303455736
35005211
521595368
294702567
1726956429
336465782
861021530
278722862

1) Quick Sort



2) Merge Sort:



<u>CONCLUSION:</u> The worst-case running time for merge sort is always O(nlogn) whereas worst-case running time for quick sort can be O(n^2). Merge sort is faster than quick sort at the cost of space complexity. The merge sort uses an additional array for storing thus having more space complexity than quick sort. The average run time for all the blocks for merge sort is lesser than quick sort though both behave in almost the same manner in ideal conditions. I have successfully implemented the quick sort and merge sort in C and used file handling in order to generate random numbers and store the run time in a proper format in a text file.