

NAME:	NEHA NILESH GODE
EXPERIMENT:	1b
BATCH:	A3
DATE OF EXPERIMENT:	13/02/2023
DATE OF SUBMISSION:	20/02/2023

Aim: Experiment on finding the running time of the algorithms merge sort and quick sort.

Theory:

Merge Sort:

It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the **merge()** function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

Quick Sort:

This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer: Recursively, sort two subarrays with Quicksort.

Code:

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

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

```
#include<math.h>
```

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

```
void dataInput() {
```

```
    //generate 100000 random numbers
```

```
    srand(time(NULL));
```

```
    for (int i=0;i<100000; i++)
```

```
    {
```

```
        int temp = rand()%1000000000;
```

```
        FILE *fptr;
```

```
        fptr = fopen("dataInput.txt", "a");
```

```
        fprintf(fptr, "%d\n", temp);
```

```
        fclose(fptr);
```

```
    }
```

```
}
```

```
//swap function
```

```
void swap(long *xp, long *yp) {
```

```
    long temp = *xp;
```

```
    *xp = *yp;
```

```
    *yp = temp;
```

```
}
```

```
//merge sort algorithm
```

```

void merge(long arr[],long temp[],int mid,int left,int right) {
    int i,left_end,size,temp_pos;
    left_end = mid-1;
    temp_pos = left;
    size = right-left+1;
    while((left<=left_end)&&(mid<=right)) {
        if(arr[left]<=arr[mid]) {
            temp[temp_pos] = arr[left];
            temp_pos = temp_pos+1;
            left = left+1;
        }
        else {
            temp[temp_pos] = arr[mid];
            temp_pos = temp_pos+1;
            mid = mid+1;
        }
    }
    while(left<=left_end) {
        temp[temp_pos] = arr[left];
        left = left+1;
        temp_pos = temp_pos+1;
    }
    while(mid<=right) {
        temp[temp_pos] = arr[mid];
        mid = mid+1;
        temp_pos = temp_pos+1;
    }
}

```

```

    }
    for(i=0;i<=size;i++) {
        arr[right] = temp[right];
        right = right-1;
    }
}

void mergeSort(long arr[],long temp[],int left,int right) {
    int mid;
    if(right>left) {
        mid = (right+left)/2;
        mergeSort(arr,temp,left,mid);
        mergeSort(arr,temp,mid+1,right);
        merge(arr,temp,mid+1,left,right);
    }
}

```

```

//quick sort algorithm
int partition(long arr[], int low, int high) {
    int left, right, pivot_item = arr[low];
    left = low;
    right = high;
    while(left<right) {
        while(arr[left]<=pivot_item) {
            left++;
        }
        while(arr[right]>pivot_item) {

```

```

        right--;
    }
    if(left<right) {
        swap(&arr[left], &arr[right]);
    }
}
arr[low] = arr[right];
arr[right] = pivot_item;
return right;
}

void quickSort(long arr[], int low, int high) {
    int pivot;
    if (low<high) {
        pivot = partition(arr, low, high);
        quickSort(arr, low, pivot-1);
        quickSort(arr, pivot+1, high);
    }
}

int main(int argc, char const *argv[])
{
    //gen data
    dataInput();
    //read data from file
    FILE *fptr;
    fptr = fopen("dataInput.txt", "r");
    long arr[100000], arr1[100000], arr2[100000];

```

```

for (int i = 0; i < 100000; i++)
{
    fscanf(fp, "%8ld", &arr[i]);
}
fclose(fp);
int s = 100;
printf("Size\tMerge Sort\tQuick Sort\n");
for(int i=0;i<1000;i++)
{
    for(int j=0;j<s;j++)
    {
        arr1[j] = arr[j];
        arr2[j] = arr[j];
    }
    double diff1, diff2;
    struct timespec start, end;

    //merge sort
    clock_gettime(CLOCK_MONOTONIC, &start);
    long temp[s];
    mergeSort(arr1, temp, 0, s-1);
    clock_gettime(CLOCK_MONOTONIC, &end);
    diff1 = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) /
1000000000.0;

    //quick sort
    clock_gettime(CLOCK_MONOTONIC, &start);

```

```

    quickSort(arr2, 0, s-1);

    clock_gettime(CLOCK_MONOTONIC, &end);

    diff2 = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) /
1000000000.0;

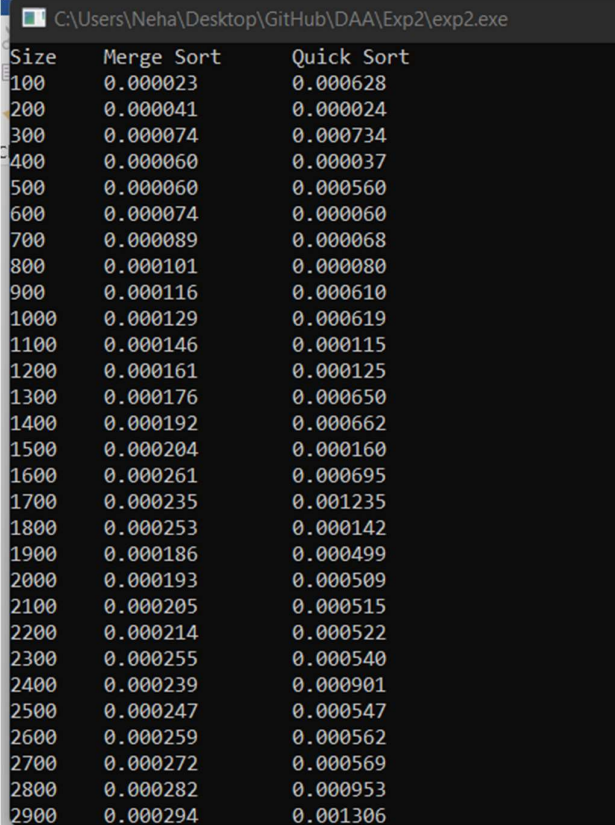
    printf("%d\t%f\t%f\n", s, diff1, diff2);

    s += 100;
}

return 0;
}

```

Output:

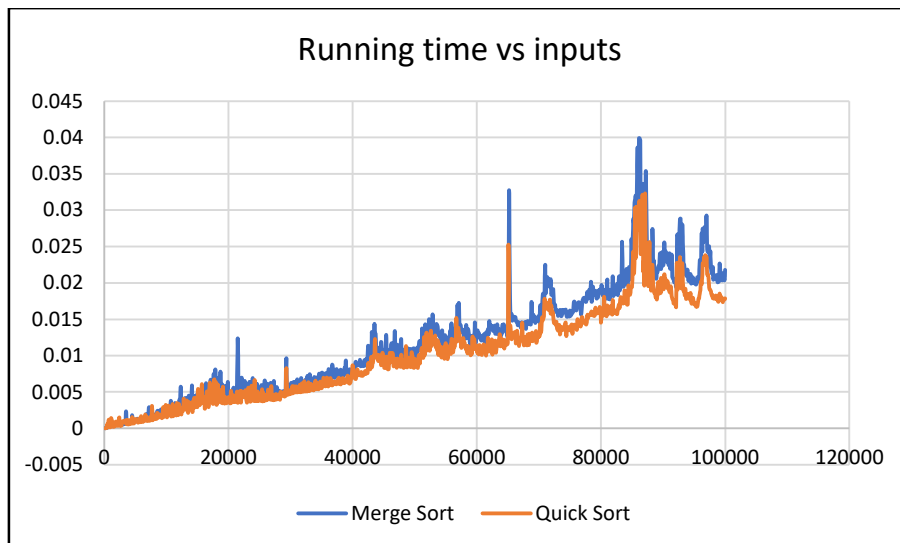


Size	Merge Sort	Quick Sort
100	0.000023	0.000628
200	0.000041	0.000024
300	0.000074	0.000734
400	0.000060	0.000037
500	0.000060	0.000560
600	0.000074	0.000060
700	0.000089	0.000068
800	0.000101	0.000080
900	0.000116	0.000610
1000	0.000129	0.000619
1100	0.000146	0.000115
1200	0.000161	0.000125
1300	0.000176	0.000650
1400	0.000192	0.000662
1500	0.000204	0.000160
1600	0.000261	0.000695
1700	0.000235	0.001235
1800	0.000253	0.000142
1900	0.000186	0.000499
2000	0.000193	0.000509
2100	0.000205	0.000515
2200	0.000214	0.000522
2300	0.000255	0.000540
2400	0.000239	0.000901
2500	0.000247	0.000547
2600	0.000259	0.000562
2700	0.000272	0.000569
2800	0.000282	0.000953
2900	0.000294	0.001306

```
Home Insert Draw Design Layout References Mailings
C:\Users\Neha\Desktop\GitHub\DAA\Exp2\exp2.exe
97800 0.039046 0.036715
97900 0.040895 0.035649
98000 0.037832 0.035130
98100 0.037139 0.035679
98200 0.040442 0.037805
98300 0.039403 0.035226
98400 0.037607 0.035209
98500 0.039969 0.035979
98600 0.038810 0.036411
98700 0.046706 0.037955
98800 0.038509 0.035749
98900 0.039376 0.037482
99000 0.038157 0.036031
99100 0.041566 0.036014
99200 0.039195 0.035804
99300 0.039428 0.036802
99400 0.039458 0.036379
99500 0.039252 0.034604
99600 0.036186 0.031956
99700 0.039727 0.035493
99800 0.039597 0.034861
99900 0.037950 0.036012
100000 0.042187 0.036533

Process returned 0 (0x0)   execution time : 48.799 s
Press any key to continue.
```

Graphs:



Observation:

For the initial lower input numbers, both merge and quick sort provide results in almost similar runtimes. Quick sort is a tad bit faster than merge sort for higher number of inputs.

Conclusion:

Quick sort takes less runtime and hence is a little more feasible than Merge sort for higher number of inputs.