U19CS076 DAA ASSIGNMENT 3

KRITHIKHA BALAMURUGAN

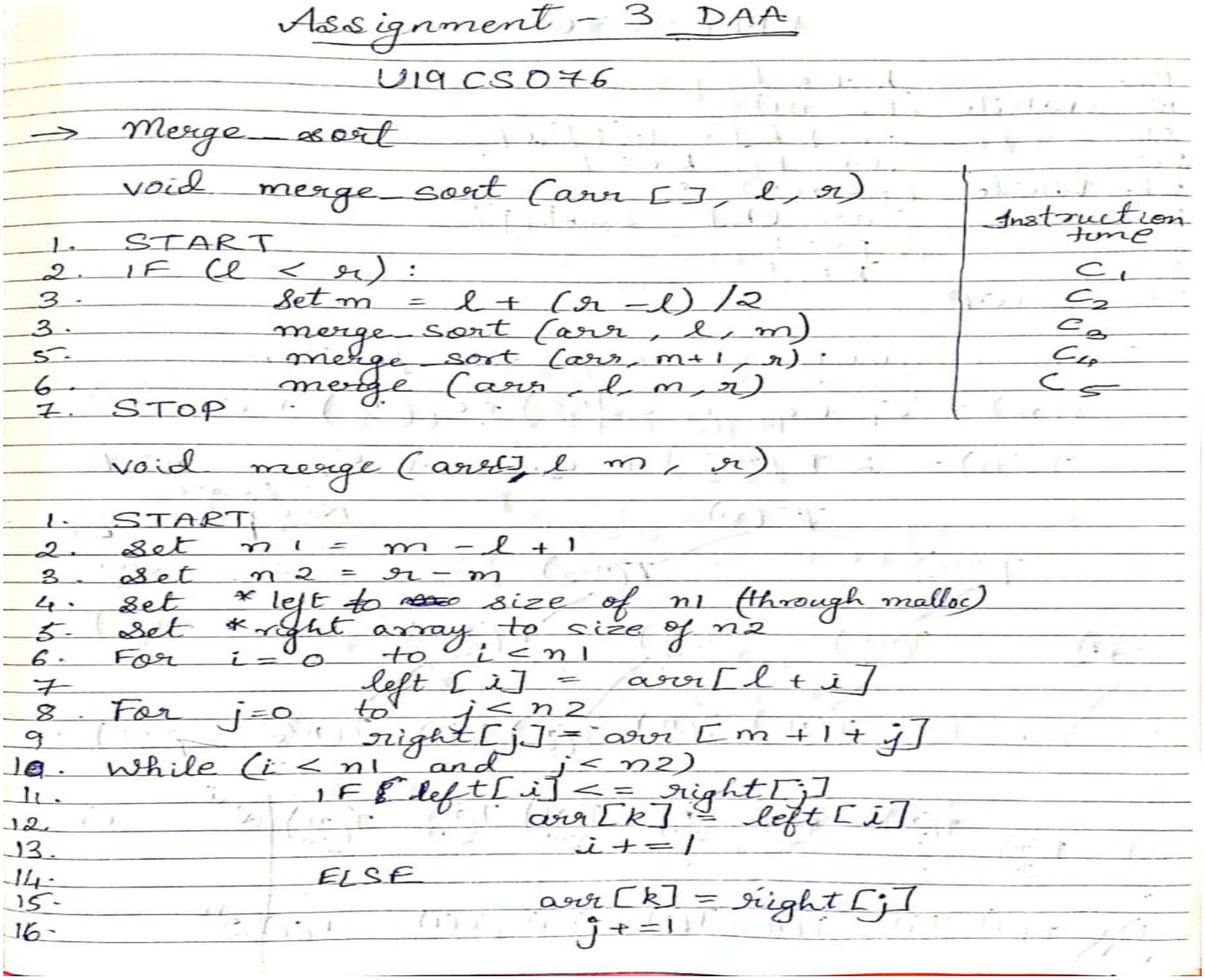
1. Write a program to sort an array arr,consisting n numbers using the divide and conquer approach - Use only merge sort.

(1) The divide step should split the array into two (nearly) equal sub-arrays.

(2) The divide step should split the array into three (nearly) equal sub-arrays. Answer the following questions.

1.1. Write pseudocodes to design the algorithms for above mentioned computa tional problem. Both algorithms should sort the data by dividing them into two and three (nearly) equal sub-arrays respectively.

1.2 Analyze the time complexity of both algorithms (split the array into two and three sub-arrays) using the recursion tree method (Include the handwritten analysis of these algorithms as an image in the latex/word file. Make sure that the images/contents are readable.).



Assignment - 3 DAA

U19CS076

→ Merge_sort

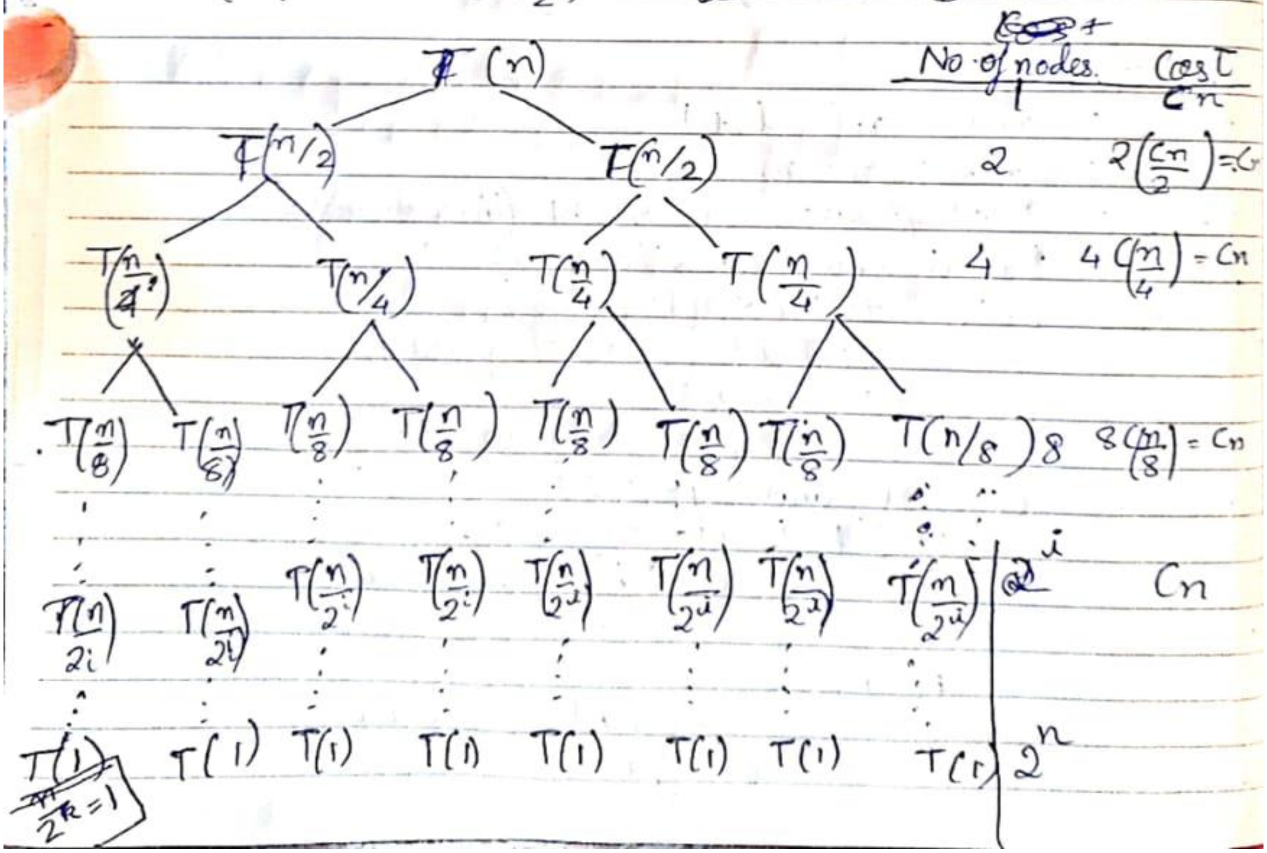
```
void merge_sort (arr [], l, r)                 Instruction
                                                  time
START
IF (l < r) :                                      C1
        Set m = l + (r - l) / 2                   C2
        merge_sort (arr, l, m)                    C3
        merge_sort (arr, m+1, r)                  C4
        merge (arr, l, m, r)                      C5
STOP

void merge (arr[], l m, r)

START
Set n1 = m - l + 1
Set n2 = r - m
Set *left ~~to~~ size of n1 (through malloc)
Set *right array to size of n2
For i = 0 to i < n1
        left [i] = arr[l + i]
For j=0 to j < n2
        right[j] = arr [m + 1 + j]
While (i < n1 and j < n2)
        IF [left[i] <= right[j]
                arr[k] = left [i]
                i += 1
        ELSE
                arr [k] = right [j]
                j += 1
```

```
17.         k += 1
18.  while (i < n1)
19.         arr[k] = left[i]
20.         i += 1 , k += 1
21.  While (j < n2)
22.         arr[k] = right[j]
23.         j += 1
24.         k += 1
25.  STOP
```

Time complexity of merge_sort function

$$T(n) = C_1 + C_2 + C_3(n/2) + C_4(n/2) + C_5(n)$$

$$T(n) = 2T(n/2) + C_a n + C_b \quad , n > 1$$



~~Cost~~
No. of nodes
Cost
$T(n)$
1
Cn
$T(n/2)$
$T(n/2)$
2
$2\left(\frac{Cn}{2}\right) = Cn$
$T\left(\frac{n}{4}\right)$
$T(n/4)$
$T\left(\frac{n}{4}\right)$
$T\left(\frac{n}{4}\right)$
4
$4\left(\frac{n}{4}\right) = Cn$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T\left(\frac{n}{8}\right)$
$T(n/8)$
8
$8\left(\frac{n}{8}\right) = Cn$
$T\left(\frac{n}{2^i}\right)$
2^i
Cn
$T(1)$
$\frac{n}{2^k} = 1$
2^n

U19CS076

$\frac{n}{2^k} = 1 \rightarrow k = \log n$

$\Rightarrow T(n) = n \cdot \log n$
$= O(n \log n)$

Divide and Conquer [merge_sort] $= 2T\left(\frac{n}{2}\right) + c$

merge function $= \Theta(n)$

$\Rightarrow$ Merge Sort [3 Way]

```
void merge_sort (arr[], n) // To copy into duplicate array
1. START
2. IF n = 0 , return
3. Set * dup array with size of n (using malloc)
4. FOR i = 0 to i < n
       dup[i] = arr[i]
5. merge_sort_func(dup, 0, n, arr)
6. For i=0 to i < n
       // Bring back data to array
       arr[i] = dup[i]
7. STOP
```

```
void merge_sort_func (dup[], l, h, arr[])
                                                     Cost
1. START
2. IF (h - l < 2) return                             c1
3. mid1 = l + ((h - l)/3)                            c2
4. mid2 = l + 2* ((h-l)/3 + 1)                       c3
5. merge_sort_func(arr, l, mid1, dup)                T(n/3)
6. merge_sort_func(arr, mid1, mid2, dup)             T(n/3)
7. merge_sort_func(arr, mid2, h, dup)                T(n/3)
8. merge(arr, l, mid1, mid2, h, dup)                 Θ(n)
```

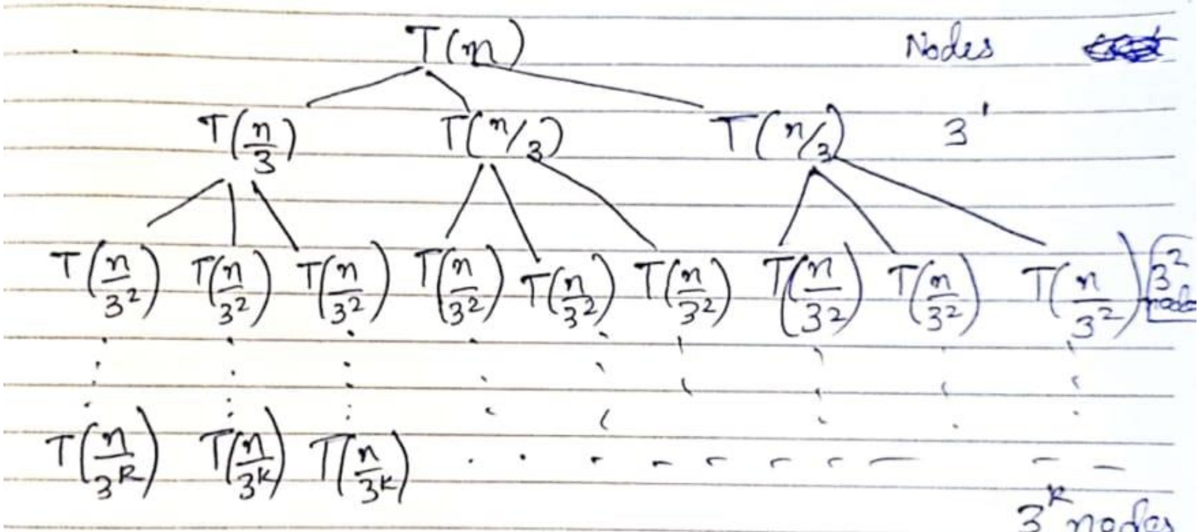
```
void merge (dup[], lw, mid1, mid2, h, arr[])
// lw is lower limit, h is higher limit
1. START
~~2. while i < mid1~~
2. Set i = lw, j = mid1, k = mid2, l = lw
3. while ( i < mid1) & (j < mid2) & (k < h)
4           IF dip[i] < dup[j]
5.               IF dup[i] < dup[k]
6.                   arr[l++] = dup[i++]
7                ELSE
8                    arr[l++] = dup[k++]
9.          ELSE
10.              IF (dup[j] < dup[k]
11.                  arr[l++] = dup[j++]
12               ELSE
13                   arr[l++] = dup[k++]
14. ~~STOP~~ while (j < mid2) & (k < h)
15          IF (dup[j] < dup[k]
16               arr[l++] = dup[j++]
17          ELSE
18               arr[l++] = dup[k++]
19. While (i < mid1) & (k < h)
20          IF dup[i] < dup[k]
21               arr[l++] = dup[i++]
22          ELSE
23               arr[l++] = dup[k++]
24. While (i < mid1)
25          arr[l++] = dup[i++]
26. While (j < mid2)
27          arr[l++] = dup[j++]
28  While (k < h)
29          arr[l++] = dup[k++]
30  STOP
```

U19CS076

For merge_sort_func

$$T(n) = 3\left[T\left(\frac{n}{3}\right) + C_4 n + [C_1 + C_2 + C_3]\right]$$

$$= \begin{cases} 3T\left(\frac{n}{3}\right) + C_a n + C_b & n > 1 \\ 1 & n = 1 \end{cases}$$


$T(n)$
Nodes
$T\left(\frac{n}{3}\right)$
$T(n/3)$
$T(n/3)$
3^1
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
$T\left(\frac{n}{3^2}\right)$
3^2 nodes
$T\left(\frac{n}{3^k}\right)$
$T\left(\frac{n}{3^k}\right)$
$T\left(\frac{n}{3^k}\right)$
3^k nodes

$\Rightarrow \frac{n}{3^k} = 1 \quad \rightarrow n = 3^k, \quad k = \log_3 n$

$T(n) = kn = n \log_3 n$

$T(n) = O(n \log_3 n)$

1.3Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

Compiler	Dev C++ 5.11
OS Name	Microsoft Windows 10 Home (i5 8th Gen)
Version	10.0.19042 Build 19042
System Name	DESKTOP-BLE6CMQ
System Model	HP Pavilion x360 Convertible 14-ba1xx
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Insyde F.54, 04-12-2019
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.88 GB
Available Physical Memory	1.75 GB
Total Virtual Memory	12.4 GB
Available Virtual Memory	4.59 GB
Page File Space	4.50 GB

1.4 Submit the code (complete programs).

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
clock_t begin, end;
double time_;
//u19cs076
void merge(long long arr[], long long l, long long m, long long r)
{
```

```
long long i;
          long long j;
          long long k;
long long n1 = m - l + 1;
long long n2 = r - m;

long long *left;
long long *right;
left=(long long*)malloc(n1*(sizeof(long long)));
          right=(long long*)malloc(n2*(sizeof(long long)));


for (i = 0; i < n1; i++)
    left[i] = arr[l + i];
for (j = 0; j < n2; j++)
    right[j] = arr[m + 1 + j];

i = 0;
j = 0;
k = l;
while (i < n1 && j < n2) {
    if (left[i] <= right[j]) {
        arr[k] = left[i];
        i++;
    }
```

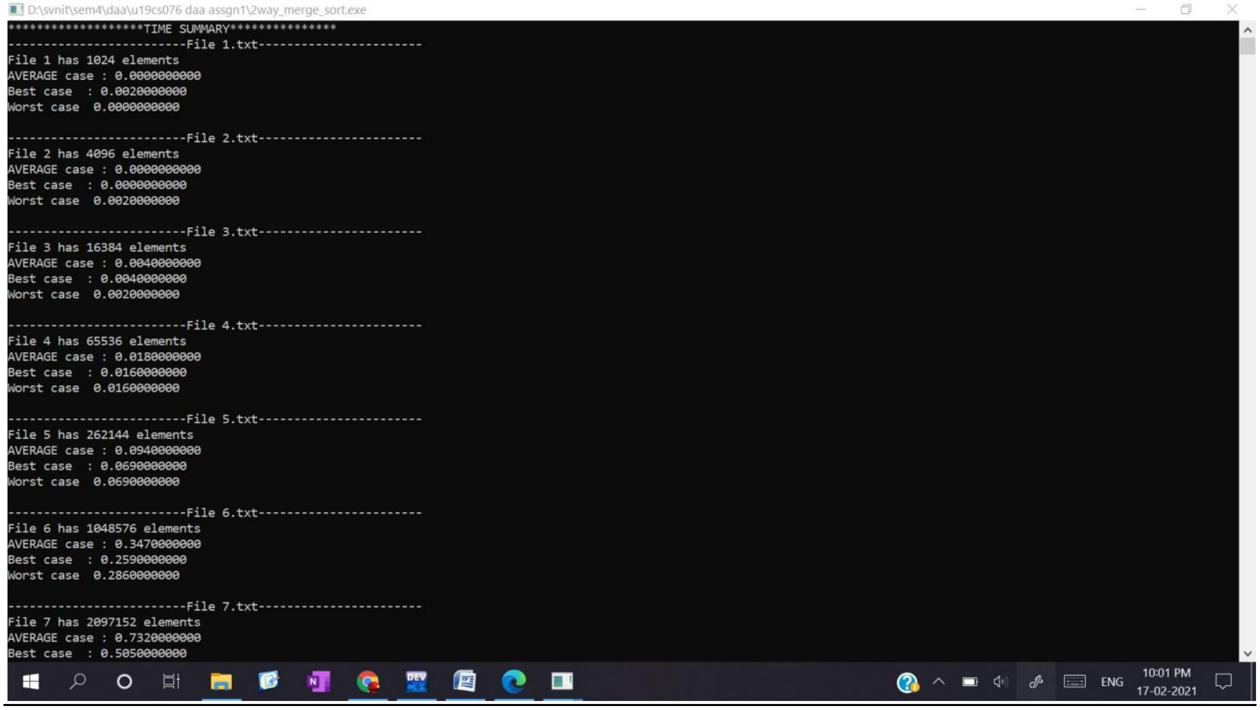
```
        else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = left[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = right[j];
        j++;
        k++;
    }
    free(left);
    free(right);
}
```

```
void merge_sort(long long arr[], long long l, long long r)
{
    if (l < r) {

        long long m = l + (r - l) / 2;
        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
void merge_dec(long long arr[], long long l, long long m, long long r)
{
   long long i;
   long long j;
    long long k;
    long long n1 = m - l + 1;
    long long n2 = r - m;
    long long *left;
    long long *right;
    left=(long long*)malloc(n1*(sizeof(long long)));
              right=(long long*)malloc(n2*(sizeof(long long)));


    for (i = 0; i < n1; i++)
        left[i] = arr[l + i];
```

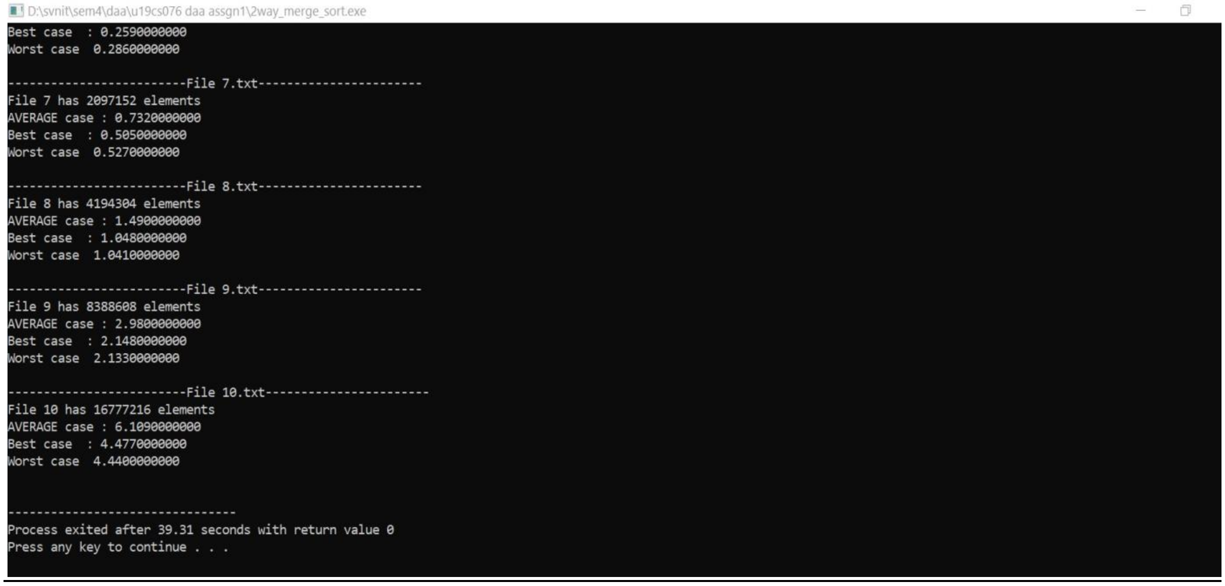
```
for (j = 0; j < n2; j++)
    right[j] = arr[m + 1 + j];

i = 0;
j = 0;
k = l;
while (i < n1 && j < n2) {
    if (left[i] >= right[j]) {
        arr[k] = left[i];
        i++;
    }
    else {
        arr[k] = right[j];
        j++;
    }
    k++;
}
while (i < n1) {
    arr[k] = left[i];
    i++;
    k++;
}
while (j < n2) {
    arr[k] = right[j];
    j++;
```

```
        k++;
    }
}
void merge_sort_dec(long long arr[], long long l, long long r)
{
    if (l < r) {

        long long m = l + (r - l) / 2;

        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);

        merge_dec(arr, l, m, r);
    }
}

long long count(char file[])
{
              FILE *fp = fopen(file, "r");
              long long count = 0;
              char b[100];
                            while(fscanf(fp, "%s\n", &b) == 1)
                                                        count++;
              fclose(fp);
              return count;
```

```
}

int main()

{

        long long n;

        long long j;

        long long *arr;             //array to hold data

        int i;

        char filename[15];

        FILE *fp;

        printf("*******************TIME SUMMARY***************\n");

        for(i=0;i<10;i++)

        {

                sprintf(filename, "File %d.txt", i+1);

                n = count(filename);

                printf("------------------------File %d.txt----------------------\n",i+1);

                printf("File %d has %lld elements\n",i+1,n);

                fp = fopen(filename, "r");

                arr=(long long*)malloc(n*((long long)sizeof(long long)));

                for(j=0; j<n; j++)

                {

                        fscanf(fp, "%lld", &arr[j]);

                }

                begin= clock();

                merge_sort(arr,0,n-1);

                end = clock();
                fclose(fp);
                time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
                printf("AVERAGE case : %0.10lf\n", time_);

                begin = clock();
                merge_sort(arr,0,n-1);
                end = clock();
                fclose(fp);
                time_= ((double)(end-begin)) / CLOCKS_PER_SEC;
                printf("Best case  : %0.10lf\n",time_);

                begin = clock();
                merge_sort_dec(arr,0,n-1);
                end = clock();
                fclose(fp);
                time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
                printf("Worst case  %0.10lf\n\n", time_);
                free(arr);
        }
}
```



```
******************TIME SUMMARY***************
------------------------File 1.txt----------------------
File 1 has 1024 elements
AVERAGE case : 0.0000000000
Best case  : 0.0020000000
Worst case  0.0000000000

------------------------File 2.txt----------------------
File 2 has 4096 elements
AVERAGE case : 0.0000000000
Best case  : 0.0000000000
Worst case  0.0020000000

------------------------File 3.txt----------------------
File 3 has 16384 elements
AVERAGE case : 0.0040000000
Best case  : 0.0040000000
Worst case  0.0020000000

------------------------File 4.txt----------------------
File 4 has 65536 elements
AVERAGE case : 0.0180000000
Best case  : 0.0160000000
Worst case  0.0160000000

------------------------File 5.txt----------------------
File 5 has 262144 elements
AVERAGE case : 0.0940000000
Best case  : 0.0690000000
Worst case  0.0690000000

------------------------File 6.txt----------------------
File 6 has 1048576 elements
AVERAGE case : 0.3470000000
Best case  : 0.2590000000
Worst case  0.2860000000

------------------------File 7.txt----------------------
File 7 has 2097152 elements
AVERAGE case : 0.7320000000
Best case  : 0.5050000000
```



```
Best case  : 0.2590000000
Worst case  0.2860000000

------------------------File 7.txt----------------------
File 7 has 2097152 elements
AVERAGE case : 0.7320000000
Best case  : 0.5050000000
Worst case  0.5270000000

------------------------File 8.txt----------------------
File 8 has 4194304 elements
AVERAGE case : 1.4900000000
Best case  : 1.0480000000
Worst case  1.0410000000

------------------------File 9.txt----------------------
File 9 has 8388608 elements
AVERAGE case : 2.9800000000
Best case  : 2.1480000000
Worst case  2.1330000000

------------------------File 10.txt----------------------
File 10 has 16777216 elements
AVERAGE case : 6.1090000000
Best case  : 4.4770000000
Worst case  4.4400000000


--------------------------------
Process exited after 39.31 seconds with return value 0
Press any key to continue . . .
```

3 WAY MERGE SORT

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<time.h>
```

```
clock_t begin, end;
double time_;



void merge(long long dup[], long long lw, long long mid1,
          long long mid2, long long h, long long arr[])
{
    long long i = lw, j = mid1, k = mid2, l = lw;

    while ((i < mid1) && (j < mid2) && (k < h))
    {
        if(dup[i] < dup[j])
        {
            if(dup[i] < dup[k])
            {
                arr[l++] = dup[i++];
            }
            else
            {
                arr[l++] = dup[k++];
            }
        }
        else
        {
            if(dup[j] < dup[k])
            {

                arr[l++] = dup[j++];

            }

            else

            {

                arr[l++] = dup[k++];

            }

        }

    }



    while ((i < mid1) && (j < mid2))

    {

        if(dup[i] < dup[j])

        {

            arr[l++] =dup[i++];

        }

        else

        {

            arr[l++] = dup[j++];

        }

    }



    while ((j < mid2) && (k < h))
```

```
{
    if(dup[j] < dup[k])
    {
        arr[l++] =dup[j++];
    }
    else
    {
        arr[l++] = dup[k++];
    }
}


while ((i < mid1) && (k < h))
{
    if(dup[i] < dup[k])
    {
        arr[l++] = dup[i++];
    }
    else
    {
        arr[l++] = dup[k++];
    }
}

while (i < mid1)
```

```
        arr[l++] = dup[i++];

    while (j < mid2)
        arr[l++] = dup[j++];

    while (k < h)
        arr[l++] = dup[k++];
}



void mergeD(long long dup[], long long lw, long long mid1,
           long long mid2, long long h, long long arr[])
{
    long long i = lw, j = mid1, k = mid2, l = lw;

    while ((i < mid1) && (j < mid2) && (k < h))
    {
        if(dup[i] > dup[j])
        {
            if(dup[i] > dup[k])
            {
                arr[l++] = dup[i++];
            }
            else
            {
```

```
            arr[l++] = dup[k++];
        }
    }
    else
    {
        if(dup[j] > dup[k])
        {
            arr[l++] = dup[j++];
        }
        else
        {
            arr[l++] = dup[k++];
        }
    }
}


while ((i < mid1) && (j < mid2))
{
    if(dup[i] > dup[j])
    {
        arr[l++] =dup[i++];
    }
    else
    {
        arr[l++] = dup[j++];
    }
}


while ((j < mid2) && (k < h))
{
    if(dup[j] > dup[k])
    {
        arr[l++] =dup[j++];
    }
    else
    {
        arr[l++] = dup[k++];
    }
}


while ((i < mid1) && (k < h))
{
    if(dup[i] > dup[k])
    {
        arr[l++] = dup[i++];
    }
    else
```

```
        {
            arr[l++] = dup[k++];
        }
    }

    while (i < mid1)
        arr[l++] = dup[i++];

    while (j < mid2)
        arr[l++] = dup[j++];

    while (k < h)
        arr[l++] = dup[k++];
}

void merge_sort_func(long long dup[], long long l,
                     long long h, long long arr[])
{
    if (h - l < 2)
        return;

    long long mid1 = l + ((h - l) / 3);             //1/3 part
    long long mid2 = l + 2 * ((h - l) / 3) + 1; //2/3part

    merge_sort_func(arr,l,mid1,dup)  ;
    merge_sort_func(arr,mid1,mid2,dup) ;
    merge_sort_func(arr,mid2,h,dup)  ;

    merge(arr, l, mid1, mid2, h, dup);
}

void merge_sort(long long arr[], long long n)
{
    if (n == 0)
        return;

    long long *dup=(long long*)malloc(n*((long long)sizeof(long long)));



              long long i;
    for (i = 0; i < n; i++)
        dup[i] = arr[i];

    merge_sort_func(dup, 0, n, arr);


    for (i = 0; i < n; i++)
        arr[i] = dup[i];
}
```

```
void merge_sort_dfunc(long long dup[], long long l,
                    long long h, long long arr[])
{
    if (h - l < 2)
        return;

    long long mid1 = l + ((h - l) / 3);            //1/3 part
    long long mid2 = l + 2 * ((h - l) / 3) + 1; //2/3part

    merge_sort_func(arr,l,mid1,dup)  ;
    merge_sort_func(arr,mid1,mid2,dup) ;
    merge_sort_func(arr,mid2,h,dup)  ;

    mergeD(arr, l, mid1, mid2, h, dup);
}

void merge_sort_dec(long long arr[], long long n)
{
    if (n == 0)
        return;

    long long *dup=(long long*)malloc(n*((long long)sizeof(long long)));
```

```
	long long i;
    for (i = 0; i < n; i++)
        dup[i] = arr[i];

    merge_sort_dfunc(dup, 0, n, arr);


    for (i = 0; i < n; i++)
        arr[i] = dup[i];
    free(dup);
}
long long count(char file[])
{
	FILE *fp = fopen(file, "r");
	long long count = 0;
	char b[100];
		while(fscanf(fp, "%s\n", &b) == 1)
				count++;
	fclose(fp);
	return count;
}

int main()
{
```

```
long long n;

long long j;

long long *arr;                //array to hold data

int i;

char filename[15];

FILE *fp;

printf("*******************TIME SUMMARY***************\n");

for(i=0;i<10;i++)

{

        sprintf(filename, "File %d.txt", i+1);

        n = count(filename);

        printf("-------------------------File %d.txt-----------------------\n",i+1);

        printf("File %d has %lld elements\n",i+1,n);


        fp = fopen(filename, "r");

        arr=(long long*)malloc(n*((long long)sizeof(long long)));

        for(j=0; j<n; j++)

        {

                fscanf(fp, "%lld", &arr[j]);

        }

        begin= clock();

        merge_sort(arr,n-1);

        end = clock();

        fclose(fp);

        time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
}


                    printf("AVERAGE case : %0.10lf\n", time_);


                    sprintf(filename, "File %d_asc.txt", i+1);
                    fp = fopen(filename, "r");
                    for(j=0; j<n; j++)
                    {
                              fscanf(fp, "%lld", &arr[j]);
                    }
                    begin = clock();
                    merge_sort(arr,n-1);
                    end = clock();
                    fclose(fp);
                    time_= ((double)(end-begin)) / CLOCKS_PER_SEC;
                    printf("Best case  : %0.10lf\n",time_);


                    begin = clock();
                    merge_sort_dec(arr,n-1);
                    end = clock();
                    fclose(fp);
                    time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
                    printf("Worst case  %0.10lf\n\n", time_);


                    free(arr);
          }
}
```

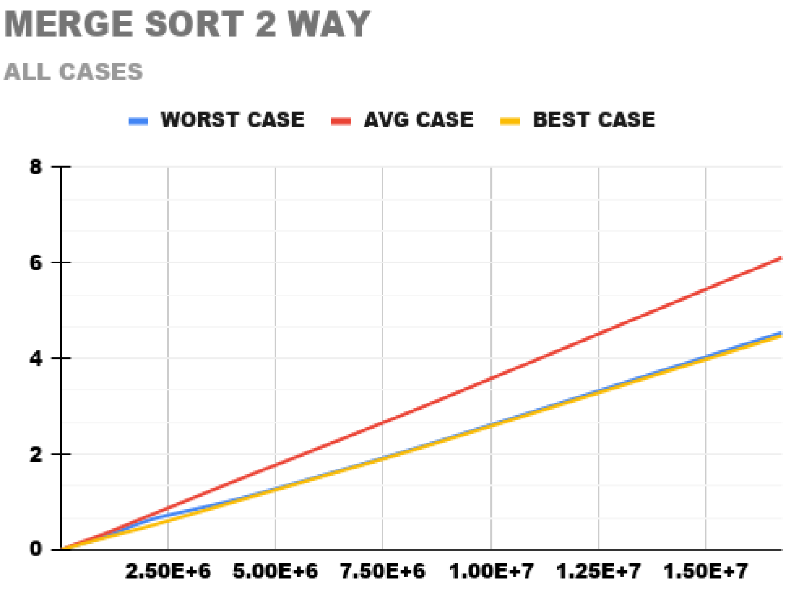
1.5Measure the best-case time, average-case time and worst-case time of the above two algorithms for all ten files (Assignment 1). Plot a graph.

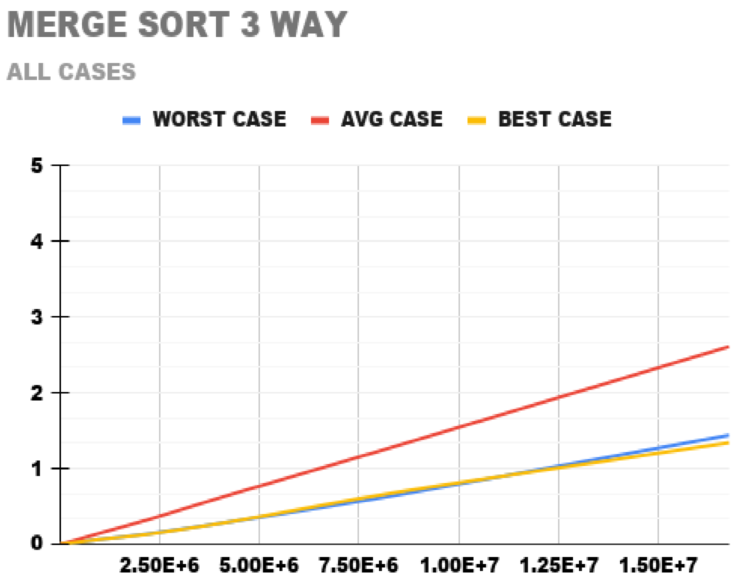
2 way merge sort

N	AVG CASE	BEST CASE	WORST CASE
1024	0	0	0.002
4096	0	0	0.002
16384	0.004	0.004	0.004
65536	0.018	0.016	0.017
262144	0.094	0.069	0.074
1048576	0.347	0.259	0.286
2097152	0.732	0.505	0.634
4194304	1.49	1.048	1.08
8388608	2.98	2.14	2.16
16777216	6.109	4.34	4.54

3 way merge sort

File	N	AVG CASE	BEST CASE	WORST CASE
1	1024	0	0	0
2	4096	0	0	0
3	16384	0.002	0.002	0
4	65536	0.006	0.002	0.002
5	262144	0.034	0.016	0.016
6	1048576	0.153	0.066	0.066
7	2097152	0.307	0.125	0.129
8	4194304	0.642	0.289	0.289
9	8388608	1.291	0.681	0.644
10	16777216	2.611	1.33	1.44





1.6 Compare the best-case performance of bubble sort, selection sort, insertion sort, and merge sort for all ten files. Plot a graph.

Best Case Analysis

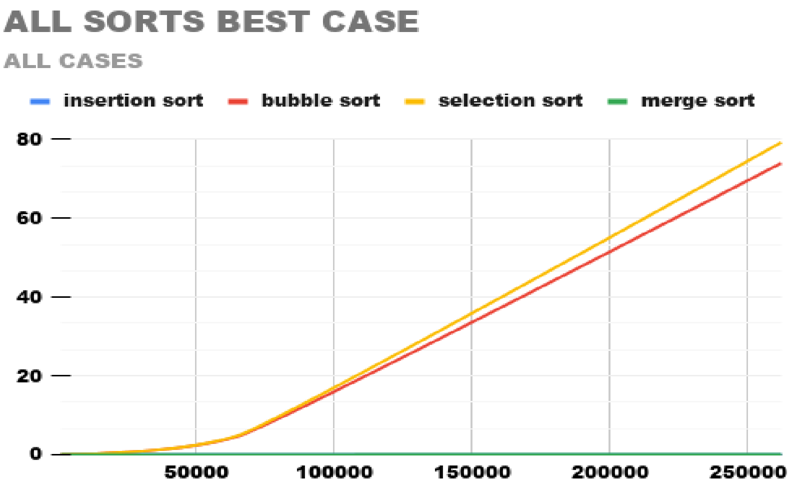
Bubble Sort: Ω(n)

Selection Sort: Ω (n^2)

Insertion Sort: Ω (n)

Merge Sort: O(n log(n))

FILE	N	bubble sort	selection sort	insertion sort	merge sort
1	1024	0.001	0	0	0
2	4096	0.02	0.02	0	0
3	16384	0.3	0.309	0	0.004
4	65536	4.753	4.949	0.001	0.016
5	262144	74	79.263	0.002	0.069



- Merge Sort takes least Time in Best Case as its Time Complexity is least.
- Next is Insertion Sort as there is no swapping so number of instructions are less.
- Next is Bubble Sort and Selection Sort takes the most time in Best Case.

1.7. (L) Compare the average-case performance of bubble sort, selection sort, insertion sort, and merge sort for all ten files. Plot a graph.

Average Case Analysis

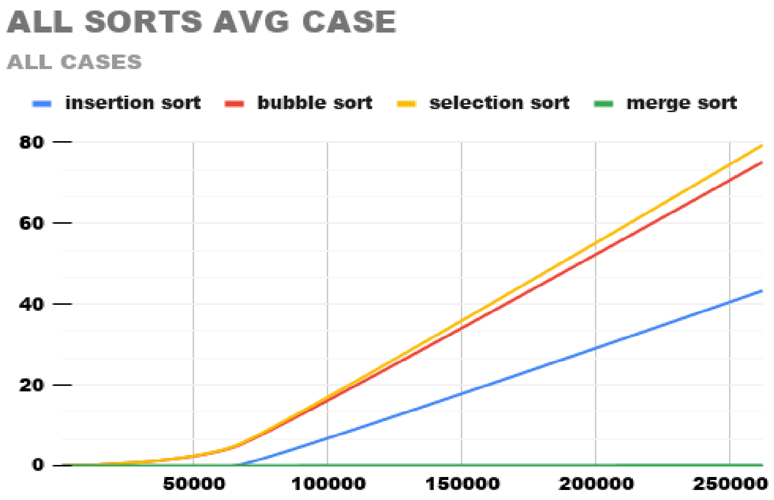
Bubble Sort: Θ(n^2)

Selection Sort: Θ(n^2)

Insertion Sort: Θ(n^2)

Merge Sort: Θ(n log(n))

FILE	N	bubble sort	selection sort	insertion sort	merge sort
1	1024	0.002	0	0	0
2	4096	0.023	0.022	0	0
3	16384	0.304	0.309	0	0.004
4	65536	4.766	4.955	0.04	0.018
5	262144	75.2	79.365	43.34	0.094



- Merge Sort(GREEN) takes least Time in Average Case as its Time Complexity is least.
- All the rest Sorting Algorithm have Average Time complexity O(n^2)

1.8. Compare the worst-case performance of bubble sort, selection sort, insertion sort, and merge sort for all ten files. Plot a graph.

Worst Case Analysis

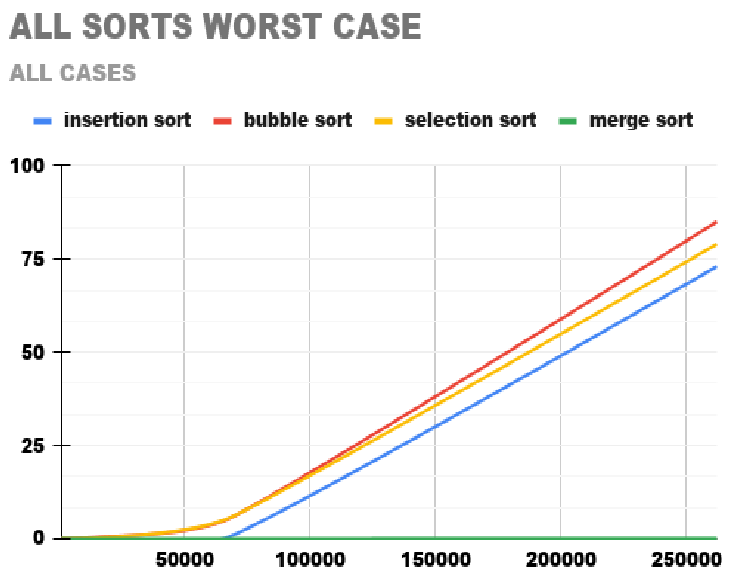
Bubble Sort: O(n^2)

Selection Sort: O(n^2)

Insertion Sort: O(n^2)

Merge Sort: O(n log(n))

FILE	N	bubble sort	selection sort	insertion sort	merge sort
1	1024	0.003	0.002	0.002	0.002
2	4096	0.022	0.02	0.002	0.002
3	16384	0.497	0.32	0.004	0.004
4	65536	4.784	4.944	0.008	0.017
5	262144	85.34	79	73.22	0.074



- Merge Sort takes least Time in Worst Case as its Time Complexity is least.
- All the rest Sorting Algorithm have Worst Time complexity O(n^2)