DAA ASSINGMENT UNIT 1&2(UE19CS251)

SUBJECT:DESIGN AND ANALYSIS OF ALGORITHMS
NAME:SATYAM RUDRADUTTA DWIVEDI
SEM:4
BRANCH CSE
SECTION F
SRN:PES2UG19CS370

I HAVE 1 HEADER FILE
1 IMPLEMENATAION FILE AND
4 DRIVER FILES(ALL SORTING ALGORITHMS HAVE ONE DRIVER CODE FOR EACH)

I used input files namely
100k.txt
150k.txt
200k.txt
250k.txt
300k.txt
350k.txt
400k.txt
450k.txt
500k.txt all are generated using **generate.c** and directing output to a file(code given below)

HEADER FILE

all.h

```
typedef struct
{
    long int *first, *second;
    long int firstLen, secondLen;
} ptrPair;
extern long long int count;
extern long int count1;
// Quick Sort
ptrPair divide(long int *, int);
int getPartitionIndex(long int *, int);
void conquer(long int *, int);
long long int quickSort(long int *, int);
int issorted(long int *, int);

//Bubble Sort
long long int BubbleSort(long int *A,int n);
//Selection Sort
long long int SelectionSort(long int *A, int n);
long int mergeSort(long int arr[], long int l, long int r);
// Implement a function which applies merge sort only if
// size of the array is larger than that of parameter
// if not apply insertion sort.
// Use mergeSortedHalfs() and insertionSort() declared above to do the same.
void merge(long int arr[], long int l, long int m, long int r);
unsigned long int merge_Sort(long int arr[], long int l, long int r);
```

DRIVER CODES

1)Selectionsort_main.c

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

void main()//this function is responsible for populating the structure by reading the file
{
   FILE *fp;//definig a file pointer
   long int arr[500000];
   long int counter=0;
   char  line[64];
   fp=fopen("500k.txt","r");//opening the file in read mode as we are only accessing informati
on
  while(fgets(line,64,fp) !=NULL) //accessing the lines one by one and trying to recognize the
 end of a line
   {
       if(line != NULL)
       {
            arr[counter]=atol(line);
            counter=counter+1;
       }
   }
   fclose(fp);
 clock_t t;
    t = clock();
long long int iterations = SelectionSort(arr,500000);
if (1 == isSorted(arr,500000))
  {
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
    for(long int k=0;k<500000;k++)
    {
        printf("%ld\n",arr[k]);
    }

    printf("took %f seconds to execute \n", time_taken);
    printf("The number of comparisons=%llu\n",iterations);
  }
  else
        printf("FAILED test of SelectionSort()\n");
}
```

2)Bubblesortmain.c

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

void main()//this function is responsible for populating the structure by reading the file
{
   FILE *fp;//definig a file pointer
   long int arr[450000];
   int counter=0;
   char  line[64];
```

```
   fp=fopen("450k.txt","r");//opening the file in read mode as we are only accessing informati
on
   while(fgets(line,64,fp) !=NULL) //accessing the lines one by one and trying to recognize th
e end of a line
   {
       if(line != NULL)
       {
            arr[counter]=atol(line);
            counter=counter+1;
       }
   }
   fclose(fp);
 clock_t t;
    t = clock();
long long int iterations = BubbleSort(arr,450000);
if (1 == isSorted(arr, 450000))
  {
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
    for(long int k=0;k<450000;k++)
    {
        printf("%ld\n",arr[k]);
    }

    printf("took %f seconds to execute \n", time_taken);
    printf("The number of comparisons=%llu\n",iterations);
  }
  else
    printf("FAILED test of bubbleSort()\n");
}
```

3)Quicksortmain.c

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

void main()//this function is responsible for populating the structure by reading the file
{
   FILE *fp;//definig a file pointer
   long int arr[300000];
   int counter=0;
   char  line[64];
   fp=fopen("300k.txt","r");//opening the file in read mode as we are only accessing informati
on
   while(fgets(line,64,fp) !=NULL) //accessing the lines one by one and trying to recognize th
e end of a line
   {
       if(line != NULL)
       {
            arr[counter]=atol(line);
            counter=counter+1;
       }
   }
  fclose(fp);

   //QuickSort
     clock_t t;
    t = clock();
    long long int iterations =quickSort(arr, 300000);
  if (1 == isSorted(arr, 300000))
```

```
  {
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

    printf("took %f seconds to execute \n", time_taken);

    printf("The number of comparisons=%llu\n",iterations);
  }
      else
        printf("FAILED test of quickSort()\n");
     int j;
    for(j=0;j<300000;j++)
    {
         printf("%ld \n",arr[j]);
    }

}
```

4)Mergesortmain.c

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

void main()//this function is responsible for populating the structure by reading the file
{
   FILE *fp;//definig a file pointer
   long int arr[300000];
   int counter=0;

   char  line[64];
   fp=fopen("300k.txt","r");//opening the file in read mode as we are only accessing informati
on

   while(fgets(line,64,fp) !=NULL) //accessing the lines one by one and trying to recognize th
e end of a line
   {
       if(line != NULL)
       {
            arr[counter]=atol(line);
            counter=counter+1;
       }
   }
   fclose(fp);
 clock_t t;
    t = clock();
  long int iterations = mergeSort(arr,0,300000-1);
if (1 == isSorted(arr, 300000))
  {
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
    for(long int k=0;k<300000;k++)
    {
        printf("%ld\n",arr[k]);
    }

    printf("took %f seconds to execute \n", time_taken);
    printf("The number of comparisons=%lu\n",iterations);
  }
  else
```

```
        printf("FAILED test of bubbleSort()\n");

}
```

I used multiple driver codes instead of one to rewrite array after sorting in every iteration

Implementation files

File has -selection&bubble,quick,merge

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

long long int BubbleSort(long int *A,int n)
{
    long int temp;
    long int i;
    long long int count = 0;
    for(i = 0; i < n - 1; i++)
    {
        int noSwaps = 0;
        for (int j = 0; j < n - 1 - i ; j++)
        {
            count++;
            if (A[j + 1] < A[j] )
            {
                temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
                noSwaps = 1;
            }
        }
        if(noSwaps == 0)
            return count;
    }
}
long long int SelectionSort(long int *A, int n)
{
    long int temp;
    long int i;
    long long int count = 0;
    for(i = 0; i < n - 1; i++)
    {
        long int min = i;
        for(int j = i + 1; j < n; j++)
        {
            count++;
            if(A[j] < A[min])
            {
                min = j;
            }
        }
        if(min != i)
        {
            temp = A[i];
            A[i] = A[min];
            A[min] = temp;
        }
```

```
    }
    return count;
}
int isSorted(long int *a, int n)
{
    for(int i=0; i<n; ++i) {
        for(int j=i+1; j<n; ++j) {
            if(a[i] > a[j]) return 0;
        }
    }
    return 1;
}


long int count1=0;
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(long int arr[], long int l, long int m, long int r)
{
    long int i, j, k;
    long int n1 = m - l + 1;
    long int n2 = r - m;

    /* create temp arrays */
    long int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back long into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        count1=count1+1;
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
```

```
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
long int mergeSort(long int arr[], long int l, long int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        long int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
    //unsigned long int count=100;
    return count1;
}




int issorted(long int *a, int n)
{
    for(int i=0; i<n; ++i) {
        for(int j=i+1; j<n; ++j) {
            if(a[i] > a[j]) return 0;
        }
    }
    return 1;
}


//quick
long long int count;
ptrPair divide(long int *a, int n)
{
   ptrPair pair;
   int p = getPartitionIndex(a, n) + 1;

   pair.first = a;
   pair.firstLen = p;
   pair.second = a + p;
   pair.secondLen = n-p;
   return pair;
}
int getPartitionIndex(long int *a, int n)
{
    int pivot = a[0];
    int i = -1;
    int j = n;
    while(1) {
        do {
            i++;
        } while(a[i] < pivot);
```

```
        do {
            j--;
        } while(a[j] > pivot);
        if(j > i) {
            int temp = a[j];
            a[j] = a[i];
            a[i] = temp;
        } else {
            return j;
        }
    }
}
void conquer(long int *a, int n)
{
    ptrPair pair;
    if(!isSorted(a, n)) {
        count=count+1;
        pair = divide(a, n);
        conquer(pair.first, pair.firstLen);
        conquer(pair.second, pair.secondLen);
    }
}
long long int  quickSort(long int *a, int n)
{
    conquer(a, n);
    return count;
}
int isSorted(long int *a, int n)
{
    for(int i=0; i<n; ++i) {
        for(int j=i+1; j<n; ++j) {
            if(a[i] > a[j]) return 0;
        }
    }
    return 1;
}
```

Input files are made using the below given code

Generate.c

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

int main()
{
    long int num;
    for(long int i=0;i<450000;i++)
    {
        num=rand();
        printf("%ld\n",num);
    }
    return 0;
}
```

We write the output to the file using
./a.out > filename.txt
And thus we obtain randomly generated value set

STATS

1)BUBBLE SORT STATS

100K
took 55.618000 seconds to execute
The number of comparisons=704885684

150K
took 94.268000 seconds to execute
The number of comparisons=2659930723

200K
took 132.512000 seconds to execute
The number of comparisons=2819884205

250K
took 244.703125 seconds to execute
The number of comparisons=31249676865

300K
took 304.781250 seconds to execute
The number of comparisons=44999846997

350K
took 418.250000 seconds to execute
The number of comparisons=61249545622

400K
took 561.687500 seconds to execute
The number of comparisons=79999670205

450K
took 710.750000 seconds to execute
The number of comparisons=101249732222

500K
took 830.631000 seconds to execute
The number of comparisons=125879450050

2)SELECTION SORT STATS

100K
took 25.789000 seconds to execute
The number of comparisons=704982704

150K
took 56.378000 seconds to execute
The number of comparisons=2659990408

200K
took 110.851000 seconds to execute
The number of comparisons=2820030816

250K
took 156.046875 seconds to execute
The number of comparisons=31249875000

300K
took 186.343750 seconds to execute
The number of comparisons=44999850000

350K
took 245.390625 seconds to execute
The number of comparisons=61249825000

400K
took 312.890625 seconds to execute

The number of comparisons=79999800000

450K
took 385.812500 seconds to execute
The number of comparisons=101249775000

500K
took 477.687000 seconds to execute
The number of comparisons=124999750000

3)MERGE SORT STATS

100K
took 6.724000 seconds to execute
The number of comparisons=1536356

150K
took 17.357000 seconds to execute
The number of comparisons=2392187
200K
took 31.899000 seconds to execute
The number of comparisons=3272743

250K
took 53.899000 seconds to execute
The number of comparisons=4168781

300K
took 89.453125 seconds to execute
The number of comparisons=5085011

350K
took 121.609375 seconds to execute
The number of comparisons=6011998

400K

took 159.765625 seconds to execute
The number of comparisons=6945971

450K
took 202.500000 seconds to execute
The number of comparisons=7889401

500K
took 249.984375 seconds to execute
The number of comparisons=8838321

4)QUICK SORT STATS

100k
took 9.937500 seconds to execute
The number of comparisons=83112

150k
took 22.421875 seconds to execute
The number of comparisons=124653

200k
took 39.968750 seconds to execute
The number of comparisons=166250

250k
took 62.000000 seconds to execute
The number of comparisons=208069

300k
took 89.640625 seconds to execute
The number of comparisons=249481

350k
took 122.546875 seconds to execute
The number of comparisons=290562

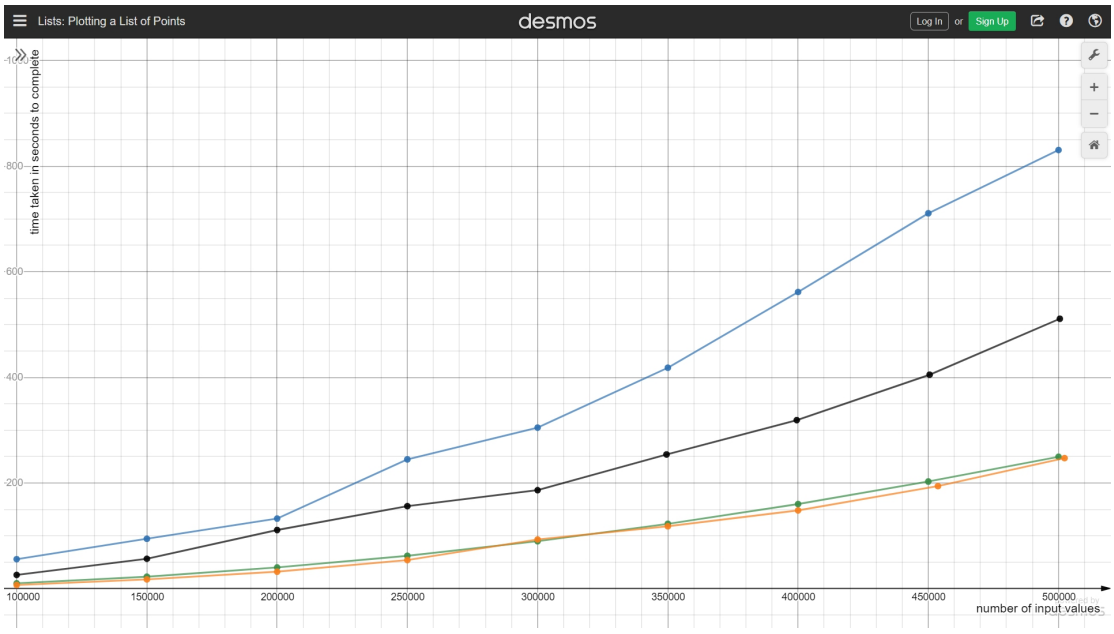
400k
took 160.031250 seconds to execute
The number of comparisons=332032

450k
took 202.703125 seconds to execute
The number of comparisons=373619

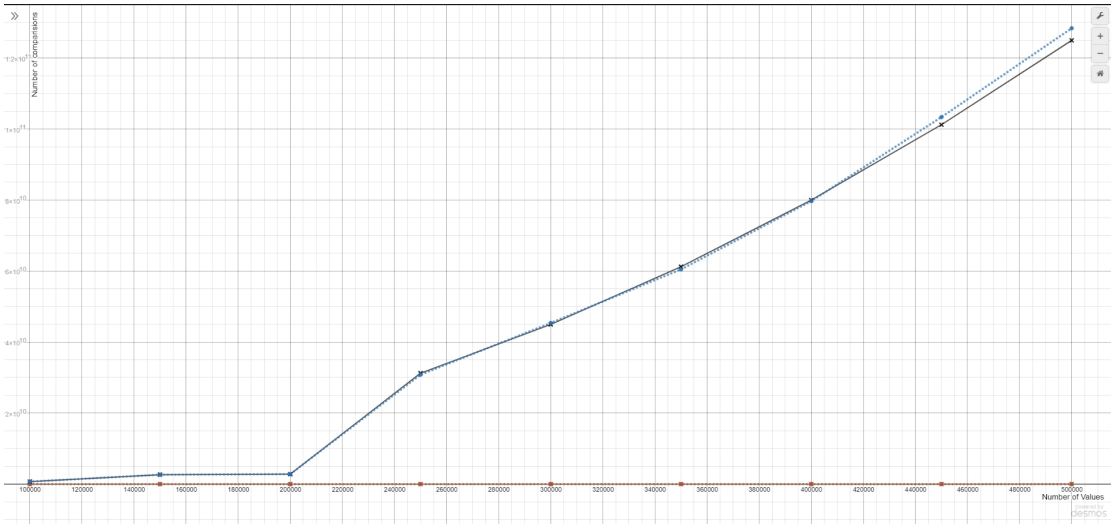
500k
took 249.578125 seconds to execute
The number of comparisons=414845

GRAPHS

1)SIZE AND TIME OF EXECUTION
Colurs and their meaning
Green-quick sort
Orage- merge srt
Black-selection sort
Blue-bubble sort



2)SIZE AND NUMBER OF ITERATIONS


Number of comparisons
1.2×10^11
1×10^11
8×10^10
6×10^10
4×10^10
2×10^10
100000
120000
140000
160000
180000
200000
220000
240000
260000
280000
300000
320000
340000
360000
380000
400000
420000
440000
460000
480000
500000
Number of Values
desmos