LAB FILE

\

NATIONAL INSTITUTE OF TECHNOLOGY DELHI
Department of Computer Science & Engineering

COURSE : DESIGN AND ANALYSIS OF ALGORITHMS
[CSB-252]

Name : **PRASUN VERMA**
Roll No. : **181210036**
Branch : **CSE**
Semester : **IVth**

Submitted to : -

INDEX

S.No	Objective	Date	Signature

1. Performance Analysis of Insertion Sort, Selection Sort and Quick Sort :

Source Code :-

```
1: /*
2: Comparison of Insertion, Selection and Quick Sort
3: with no. of comparisons and time taken to execute */
4: #include<stdio.h>
5: #include<time.h>
6: #include<stdlib.h>
7:
8: long long comp1=0,comp2=0,comp3=0; //comparisons counter
9: // Insertion Sort
10: void insertion(long long int *,long long int);
11: //Selection Sort
12: void selection(long long int *,long long int);
13: //Quick Sort
14: void quicky(long long int *,long long int);
15: long long int partition(long long int *,long long int,long long int);
16: void quickSort(long long int *,long long int,long long int);
17: void swap(long long int *,long long int *);
18:
19: void main() //main func
20: {
21: clock_t t1,t2; // clock vars
22:  double time_taken;
23:  long long arr[100000],arr2[100000],arr3[100000],n,i;    //array declaration
24:  srand(time(0));
25:  printf("Enter the size of array");   //input size
26:  scanf("%lld",&n);
27:  for( i=0;i<n;i++)
28:  {
```

```
29: arr[i]=rand();          //random values assignments
30:  }
31:  for( i=0;i<n;i++)
32:  {
33: arr2[i]=arr[i];          //random values assignments
34: arr3[i]=arr[i];
35:  }
36:
37:
38:  t1=clock();
39:  insertion(arr,n);     //calls insertion method
40:  t2=clock();
41:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
42:       printf("\n Time taken to execute Insertion sort %e ",time_taken);
43:        printf("\n Number of Comparisons %ld",comp1);
44:  t1=clock();
45: selection(arr2,n);    //calls selection method
46: t2=clock();
47:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
48:       printf("\n Time taken to execute Selection sort %e ",time_taken);
49:        printf("\n Number of Comparisons %lld",comp2);
50: t1=clock();
51: quicky(arr3,n);     //calls quick method
52: t2=clock();
53:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
54:       printf("\n Time taken to execute Quick sort %e ",time_taken);
55:        printf("\n Number of Comparisons %lld",comp3);
56:
57:  }
58:  //insertion sort def
59:  void insertion(long long int arr[],long long int up)
```

```
60: {
61: long long i, temp, j;
62:     for (i = 1; i < up; i++)
63:     {
64:         temp = arr[i];
65:         j = i - 1;
66:         while (j >= 0 && arr[j] > temp)
67:         {
68:
69:             arr[j + 1] = arr[j];   //shifts elements
70:             j = j - 1;
71:         comp1++;
72:         }
73:         arr[j + 1] = temp;
74:     }
75: }
76: //selection sort def
77: void selection(long long int arr[],long long int n)
78: {
79:
80: long long int i=0,j,pos,temp;
81: for(i=0;i<n;i++)
82: {
83:  pos=i;
84:  for(j=i+1;j<n;j++)
85:  {
86:    if(arr[j]<arr[pos])
87:   {
88:     pos=j;
89:     comp2++;
90:    }
```

```
91:    }
92:  temp=arr[i];
93:  arr[i]=arr[pos];
94:  arr[pos]=temp;
95:  }
96:   }
97:
98:    //Quick sort def
99: void quicky(long long int arr[],long long int n)
100: {
101:  quickSort(arr,0,n-1);
102:  }
103:
104:
105:  void quickSort(long long int a[],long long int l,long long int h)
106:  {
107:   long long int p;
108:      if(l<h)
109:      {
110:        p=partition(a,l,h);
111:
112:       quickSort(a,l,p-1);
113:       quickSort(a,p+1,h);
114:       }
115:    }
116:
117:    void swap(long long int *x,long long int *y)
118:    {
119:   long long int t=*x;
120:    *x=*y;
121:    *y=t;
```

```
122:    }
123:
124:    long long int partition(long long int a[],long long int i,long long int j)
125:
126:   {
127:        long long int pivot,k,l;
128:        pivot=a[j];
129:        k=(i-1);
130:        for( l=i;l<=j-1;l++)
131:        {
132:             if(a[l]<pivot)
133:             {
134:               k++;
135:                 comp3++;
136:               swap(&a[k],&a[l]);
137:               }
138:    }
139:    swap(&a[k+1],&a[j]);
140:    return (k+1);
141:   }
142:
```

TABLES :-

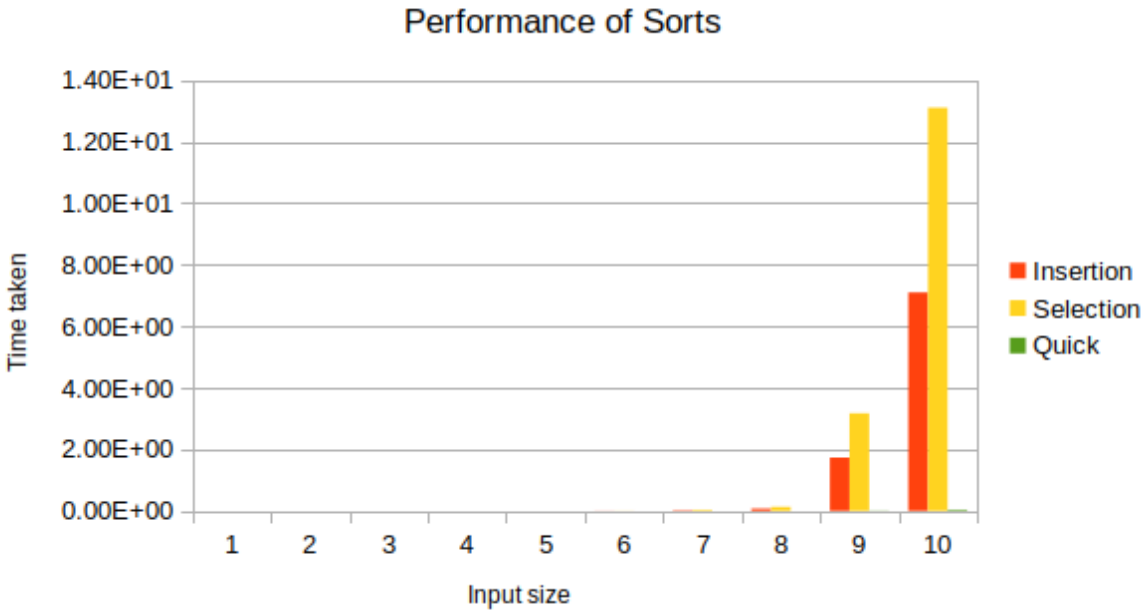
- Input v/s Time :

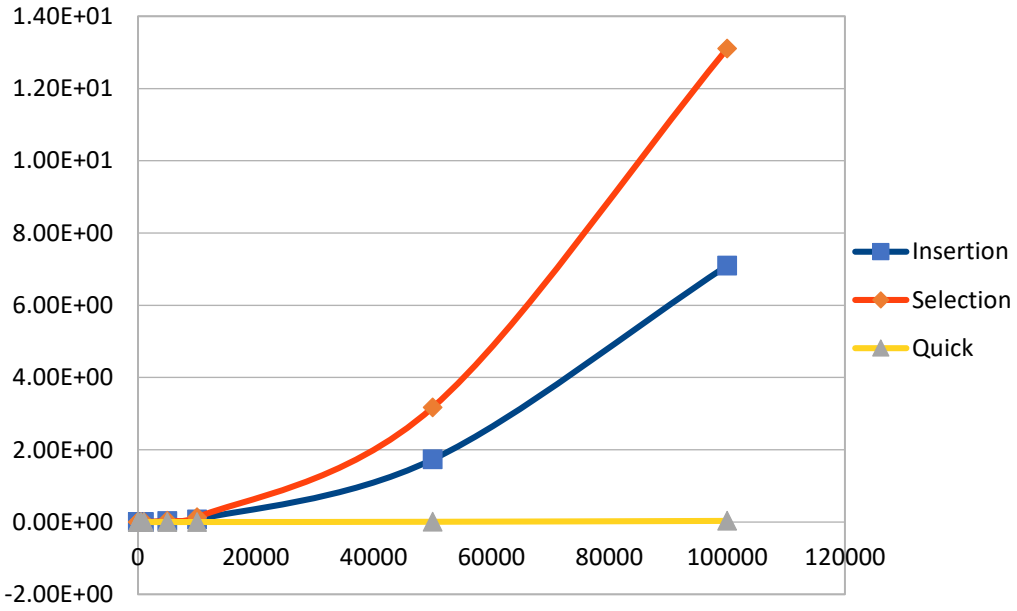
Input Size	Insertion	Selection	Quick
1	3.00E-06	3.00E-06	2.00E-06
5	7.00E-06	4.00E-06	4.00E-06
10	7.00E-06	5.00E-06	4.00E-06
100	3.80E-05	6.90E-05	3.50E-05
500	8.30E-04	1.52E-03	2.18E-04
1000	2.77E-03	3.25E-03	1.09E-04
5000	2.17E-02	3.25E-02	6.36E-04
10000	7.41E-02	1.28E-01	1.36E-03
50000	1.73E+00	3.17E+00	7.83E-03
100000	7.10E+00	1.31E+01	3.28E-02

- Input v/s Comparisons :

Input Size	Insertion	Selection	Quick
1	0	0	0
5	7	5	2
10	25	12	7
100	2713	314	334
500	60568	2403	2974
1000	252508	5313	6822
5000	6239632	35592	34995
10000	25125202	79044	82024
50000	624336643	472112	464654
100000	2502880533	1014901	980678

GRAPH :


Performance of Sorts
1.40E+01
1.20E+01
1.00E+01
8.00E+00
6.00E+00
4.00E+00
2.00E+00
0.00E+00
Time taken
1
2
3
4
5
6
7
8
9
10
Input size
Insertion
Selection
Quick


1.40E+01
1.20E+01
1.00E+01
8.00E+00
6.00E+00
4.00E+00
2.00E+00
0.00E+00
-2.00E+00
0
20000
40000
60000
80000
100000
120000
Insertion
Selection
Quick

2. Performance Analysis of Merge Sort and Quick Sort :

Source Code :-

```
1: /*
2: Comparison of Merge and Quick Sort
3: with no. of comparisons and time taken to execute */
4: #include<stdio.h>
5: #include<time.h>
6: #include<stdlib.h>
7:
8: long long comp1=0,comp2=0; //comparisons counter
9: //merge functions
10: void mergesort(long long int *,long long int ,long long int);
11: void merge(long long int *,long long int,long long int,long long int);
12: //quick functions
13: void quicky(long long int *,long long int);
14: long long int partition(long long int *,long long int,long long int);
15: void quickSort(long long int *,long long int,long long int);
16: void swap(long long int *,long long int *);
17:
18: void main() //main func
19: {
20: clock_t t1,t2; // clock vars
21:  double time_taken;
22:  long long arr[100000],arr2[100000],n,i;    //array declaration
23:  srand(time(0));
24:  printf("Enter the size of array ");   //input size
25:  scanf("%lld",&n);
26:  for( i=0;i<n;i++)
27:  {
28: arr[i]=rand();          //random values assignments
29:  }
```

```
30:  for( i=0;i<n;i++)
31:  {
32: arr2[i]=arr[i];           //random values assignments
33:  }
34:
35:  t1=clock();
36: mergesort(arr,0,n-1);    //calls mergesort method
37: t2=clock();
38:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
39:       printf("\n Time taken to execute Merge sort %e ",time_taken);
40:         printf("\n Number of Comparisons %lld",comp1);
41: t1=clock();
42: quicky(arr2,n);     //calls quick method
43: t2=clock();
44:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
45:       printf("\n Time taken to execute Quick sort %e ",time_taken);
46:         printf("\n Number of Comparisons %lld",comp2);
47:
48:  }
49:   //Merge Sort defs
50:    void mergesort(long long int arr[],long long int l,long long int r)
51: {
52:  long long int i;
53:   if(l<r)
54:    {
55:     long long int m=l+(r-l)/2;
56:      mergesort (arr,l,m);
57:      mergesort (arr,m+1,r);
58:      merge(arr,l,m,r);
59:
60:    }
```

```
61:   }
62: void merge(long long int arr[],long long int l,long long int m,long long int r)
63: {
64:   long long int i, j, k,n1,n2;
65:    n1 = m - l + 1;
66:    n2 =  r - m;
67:
68:
69:    long long int L[n1], R[n2];
70:     for (i = 0; i < n1; i++)
71:         L[i] = arr[l + i];
72:     for (j = 0; j < n2; j++)
73:         R[j] = arr[m + 1+ j];
74:     i = 0;
75:     j = 0;
76:     k = l;
77:     while (i < n1 && j < n2)
78:     {
79:         if (L[i] <= R[j])
80:         {
81:             comp1++;
82:             arr[k] = L[i];
83:             i++;
84:         }
85:         else
86:         {
87:             arr[k] = R[j];
88:             j++;
89:         }
90:         k++;
91:     }
```

```
92:     while (i < n1)
93:     {
94:         arr[k] = L[i];
95:         i++;
96:         k++;
97:
98:     }
99:
100: }
101:
102:    //Quick sort defs
103: void quicky(long long int arr[],long long int n)
104: {
105:  quickSort(arr,0,n-1);
106:  }
107:
108:
109:  void quickSort(long long int a[],long long int l,long long int h)
110:  {
111:   long long int p;
112:       if(l<h)
113:       {
114:        p=partition(a,l,h);
115:        quickSort(a,l,p-1);
116:        quickSort(a,p+1,h);
117:        }
118:    }
119:
120:    void swap(long long int *x,long long int *y)
121:    {
122:   long long int t=*x;
```

```
123:    *x=*y;
124:    *y=t;
125:    }
126:
127:    long long int partition(long long int a[],long long int i,long long int j)
128:
129:   {
130:        long long int pivot,k,l;
131:        pivot=a[j];
132:        k=(i-1);
133:        for( l=i;l<=j-1;l++)
134:        {
135:             if(a[l]<pivot)
136:             {
137:               k++;
138:                 comp2++;
139:               swap(&a[k],&a[l]);
140:               }
141:    }
142:    swap(&a[k+1],&a[j]);
143:    return (k+1);
144:   }
145:
146:
```

TABLES :-

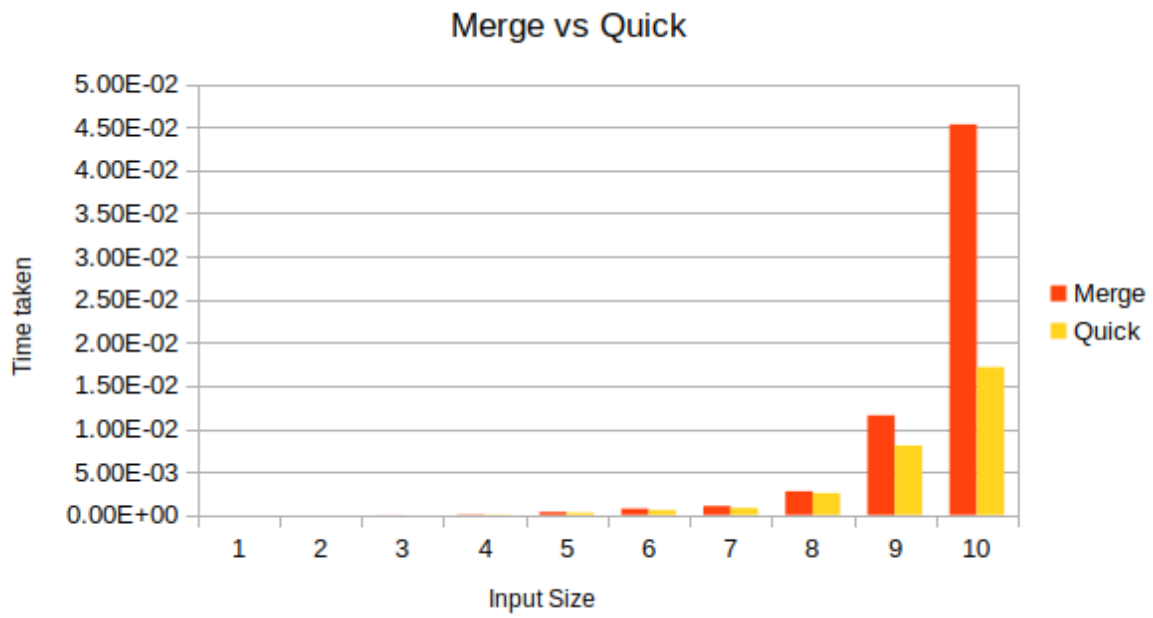
- Input v/s Time :

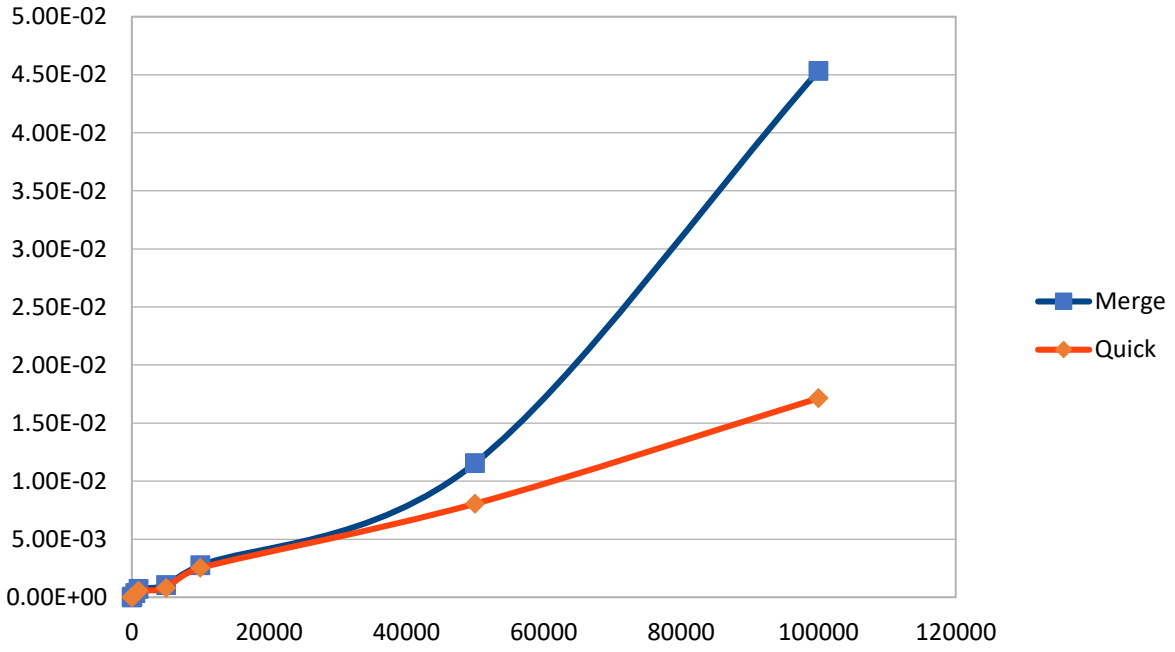
Input Size	Merge	Quick
1	1.00E-06	1.00E-06
5	2.00E-06	2.00E-06
10	1.20E-05	6.00E-06
100	6.10E-05	3.70E-05
500	3.47E-04	2.63E-04
1000	7.27E-04	5.59E-04
5000	1.03E-03	8.03E-04
10000	2.74E-03	2.53E-03
50000	1.16E-02	8.06E-03
100000	4.53E-02	1.71E-02

- Input v/s Comparisons

Input Size	Merge	Quick
1	0	0
5	5	5
10	15	18
100	278	293
500	1939	2991
1000	4349	6715
5000	28272	33092
10000	61350	78856
50000	363298	460487
100000	776630	1099802

GRAPH:-`


Merge vs Quick
Time taken
5.00E-02
4.50E-02
4.00E-02
3.50E-02
3.00E-02
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
1
2
3
4
5
6
7
8
9
10
Input Size
Merge
Quick


5.00E-02
4.50E-02
4.00E-02
3.50E-02
3.00E-02
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
0
20000
40000
60000
80000
100000
120000
Merge
Quick

3. Performance Analysis of Linear Search and Binary Search :

Source Code :-

```
1: #include<stdio.h>
2: #include<time.h>
3: #include<stdlib.h>
4:
5: typedef long long ll;
6: void binary(ll *,ll n,ll num);
7: void linear(ll *,ll n,ll num);
8:
9: void main()   // main func
10: {
11:  clock_t t1,t2;   // clock vars
12: double time_taken;
13: ll arr[100000],n,i;
14: srand(time(0));
15:  printf("\n Enter the size of array : ");
16:  scanf("%lld",&n);
17:  for(i=0;i<n;i++)
18:  {
19:   arr[i]=rand();
20:  }
21:  ll search=rand();
22:   t1=clock();
23:   linear(arr,n,search);// calls linear search
24:   t2=clock();
25:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
26:  printf("\n Time taken to execute Linear Search %e ",time_taken);
27:  t1=clock();
28:  binary(arr,n,search); // calls binary search
```

```
29:   t2=clock();
30:  time_taken = ((double)(t2-t1))/CLOCKS_PER_SEC;
31:      printf("\n Time taken to execute Binary Search %e ",time_taken);
32:
33:  }
34: void linear (ll arr[],ll n,ll num)   // linear search definition
35: {
36: int flag=0;
37: ll i;
38: for(i=0;i<n;i++)
39:  {
40:   if(arr[i]==num)
41:   {
42:   flag=1;
43:   break;
44:   }
45:   }
46:   if(flag)
47:   printf("\n Element is present at position :%lld  ",i+1);
48:   else
49:     printf("\n Element is not present");
50: }
51: void binary(ll arr[],ll n,ll num)   //binary search definition
52: {
53: int flag=0;
54: ll beg,end;
55: beg=0;
56: end=n-1;
57: ll mid=(beg+end)/2;
58:  while(beg<=end)                //Boundary Condition of Binary Search
59:  {
```

```
60:  if(arr[mid]==num)
61:  {
62:  flag=1;
63:  break;
64:  }
65:  else if(arr[mid]>num)
66:  {
67:  end=mid-1;
68:  }
69:  else if(arr[mid]<num)
70:  {
71:  beg=mid+1;
72:  }
73:  mid=(beg+end)/2;
74:  }
75:
76:   if(flag)
77:   printf("\n Element is present at position :%lld  ",mid+1);
78:   else
79:     printf("\n Element is not present");
80:   }
81:
82:
```