Complexity Theory and Algorithms(3CS1109)

Practical:-1

Aim:Implement following sorting algorithms.

a) Bubble sort

b) Selection Sort

c) Insertion Sort

d) Quick Sort

Evaluate the time complexity of each algorithm on already sorted (ascending and descending) and non-sorted input values with varying size of input values. Visualize the same using graphical representation.

❖ Implementation and analysis of Bubble Sort algorithm

```
// bubble sort

#include<stdio.h>

#include<time.h>

double totaltime=0;

void bubble(long int *a,long int n)

{

   //printf("\n%d",stime);

   long temp,i,j;

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

   {

      for(j=0; j<n-1; j++)

      {
```

```
            if(a[j]>a[j+1])
            {
                swap(&a[j], &a[j+1]);
            }
        }
    }
}

void swap(int *x, int *y)
{
    long temp = *x;
    *x = *y;
    *y = temp;
}

void print(int *a,int n)
{
    long i,p;
    printf("\nSorted array is:\n ");
    for (i=0; i<10; i++)
    {
        printf(" %d",a[i]);
    }
```

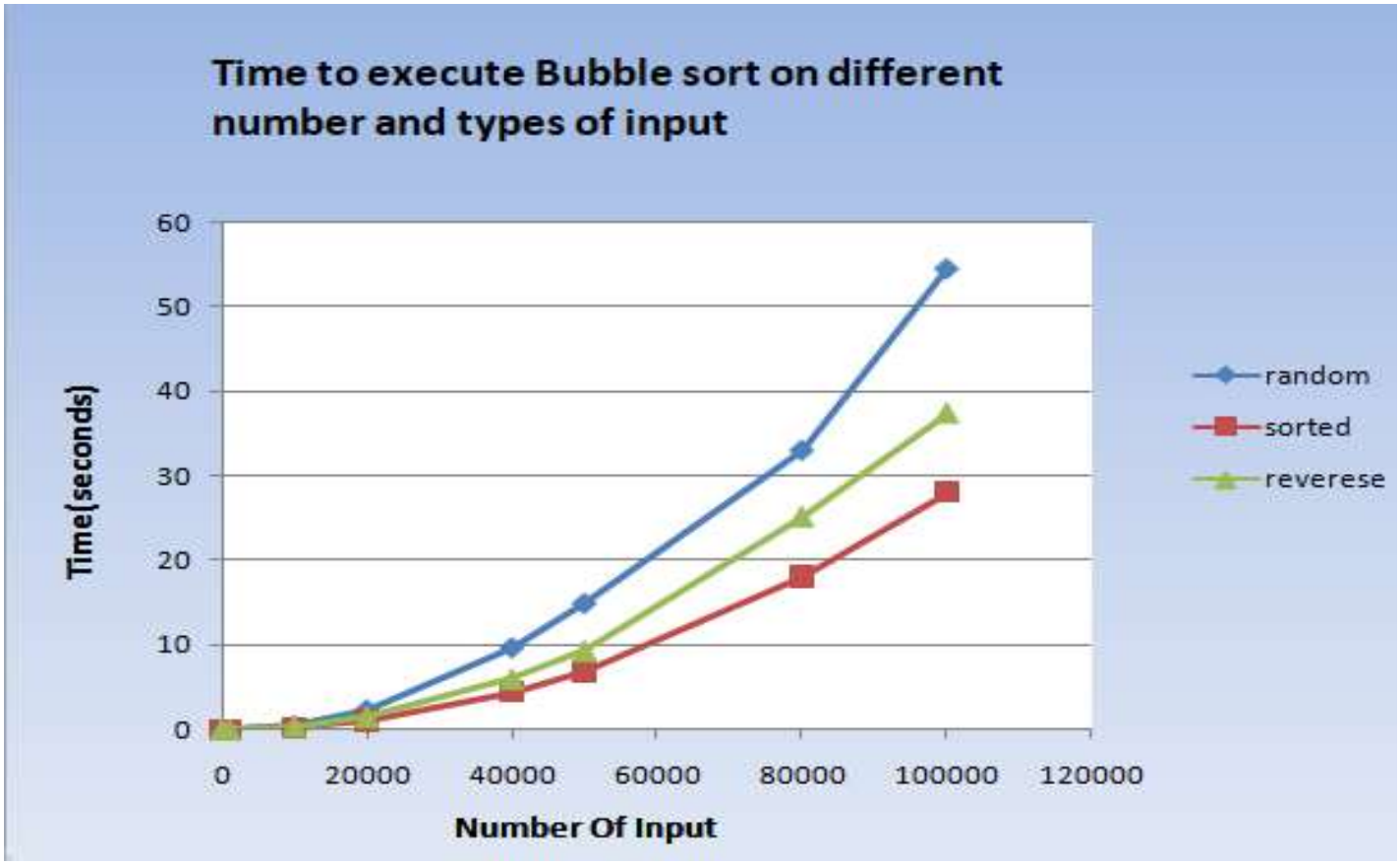
```
    if(n>10)
    {
        printf(".............");
        p=n-10;
        for (i=p; i<n; i++)
        {
            printf(" %d",a[i]);
        }
    }
}

void main()
{
   clock_t start, end;
   long  int *a,*b,*c,*d;
   long int n,i,j;
   char ch;
    printf("-----------------------------------------------Bubble Sort-----------------------------------
----------------\n");
    do{
    printf("\nEnter the range of  elements in array :");
    scanf("%d",&n);
    a= (long int*)malloc(n * sizeof(long int));
    b= (long int*)malloc(n * sizeof(long int));
```

```
    c= (long int*)malloc(n * sizeof(long int));


    printf("\n-------------------------------Sorting of Random numbers-------------------------\n");
    for (i = 0; i < n; i++)
    {
        a[i] = rand();
    }
    start = clock();
    bubble(a,n);
    end = clock();
    //printf("\n%d",etime);
    totaltime=((double) (end - start)) / CLOCKS_PER_SEC;
    print(a,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");


    printf("\n------------------Sorting of numbers which are sorted in ascending order------------
\n");
    for (i = 0; i < n; i++)
    {
        b[i] = a[i];
    }
     start = clock();
    bubble(b,n);
```

```
    end = clock();
    //printf("\n%d",etime);
    totaltime=((double) (end - start)) / CLOCKS_PER_SEC;
    print(b,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");

    printf("\n------------------Sorting of numbers which are sorted in descending order----------
\n");
       for (i=0,j=n-1; i < n; i++,j--)
    {
        c[i] = b[j];
    }
    start = clock();
    bubble(c,n);
    end = clock();
    //printf("\n%d",etime);
    totaltime=((double) (end - start)) / CLOCKS_PER_SEC;
    print(c,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");

    printf("\n Do you want to continue ? Press 'y' to continue:");
    fflush(stdin);
```

```
    scanf("%c",&ch);
     free(a);
    }
while((ch=='y') || (ch=='Y'));
}
```

Outcome:



Observation:

- For random input, bubble sort algorithm takes largest amount of time for sorting.
- For reversed sorted input, bubble sort algorithm takes moderate amount of time for sorting.
- For sorted input, bubble sort algorithm takes lowest amount of time for sorting.

Time Complexity:

- Best case:$O(n^2)$
- Average case: $\Theta (n^2)$
- Worst case: $\Omega(n^2)$

❖ Implementation and analysis of Selection sort algorithm

```
// selection sort
#include<stdio.h>
#include<time.h>
double totaltime=0;
void selection(int *a,int n)
{
   clock_t start, end;
   start = clock();
   int min;
   int temp,i,j;
   for(i=0; i<n-1; i++)
   {
      min=i;
      for(j=i+1; j<n; j++)
      {
         if(a[min]>a[j])
            min=j;
      }
      if(min!=i)
      {
         temp=a[i];
         a[i]=a[min];
```

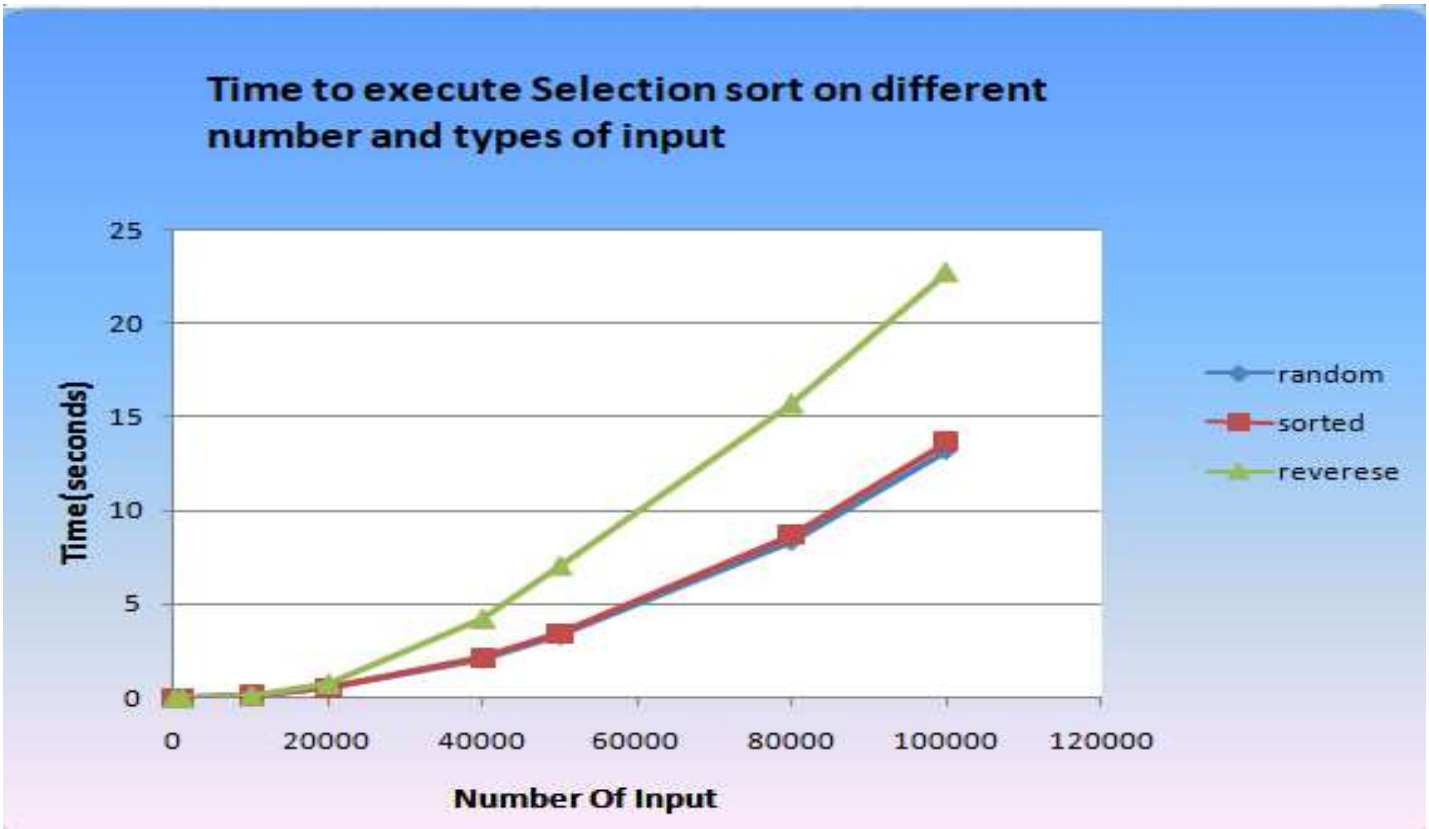
```
            a[min]=temp;
        }
    }
    end = clock();
    totaltime=((double) (end - start)) / CLOCKS_PER_SEC;
}


void print(int *a,int n)
{
    int i,p;
    printf("\nSorted array is:\n ");
    for (i=0; i<10; i++)
    {
        printf(" %d",a[i]);
    }
    if(n>10)
    {
        printf(".....");
        p=n-10;
        for (i=p; i<n; i++)
        {
            printf(" %d",a[i]);
        }
```

```
    }
}


void main()
{
    int *a,*b,*c;
    int n,i,j;
    char ch;
    printf("-----------------------------------------------Selection Sort----------------------------------
----------------------\n");
    do
    {
        printf("\nEnter the range of  elements in array :");
        scanf("%d",&n);
        a= (int*)malloc(n * sizeof(int));
        b= (int*)malloc(n * sizeof(int));
        c= (int*)malloc(n * sizeof(int));
        printf("\n------------------------------Sorting of Random numbers-------------------------
\n");
        for (i = 0; i < n; i++)
        {
            a[i] = rand();
        }
        selection(a,n);
```

```
        print(a,n);
        printf("\n total time in sorting: %f",totaltime);
        printf(" sec\n");

        printf("\n------------------Sorting of numbers which are sorted in ascending order-----------
-\n");
        for (i = 0; i < n; i++)
        {
            b[i] = a[i];
        }
        /*printf("\n sorted array input\n");

        for (i = 0; i < n; i++)
        {
            printf(" %d",b[i]);
        }*/
        selection(b,n);
        print(b,n);
        printf("\n total time in sorting: %f",totaltime);
        printf(" sec\n");

        printf("\n------------------Sorting of numbers which are sorted in descending order----------
-\n");
        for (i=0,j=n-1; i < n; i++,j--)
```

```
        {
            c[i] = b[j] ;
        }
        /*printf("\nReversed array input\n");
        for (i = 0; i < n; i++)
        {
            printf(" %d",c[i]);
        }*/
        selection(c,n);
        print(c,n);
        printf("\n total time in sorting: %f",totaltime);
        printf(" sec\n");

        printf("\n Do you want to continue ? Press 'y' to continue or any other key to exit:");
        fflush(stdin);
        scanf("%c",&ch);
        free(a);
    }
    while((ch=='y') || (ch=='Y'));
}
```

Outcome:



Observation:

- For reversed sorted input, selection sort algorithm takes largest amount of time for sorting.
- For random input, selection sort algorithm takes moderate amount of time for sorting.
- For sorted input, selection sort algorithm takes lowest amount of time for sorting.

Time Complexity:

- Best case:$O(n^2)$
- Average case: $\Theta (n^2)$
- Worst case: $\Omega(n^2)$

❖ Implementation and analysis of Insertion sort algorithm

```
// insertion sort
#include<stdio.h>
#include<time.h>
double totaltime=0;
void insertion(int *a,int n)
{
   clock_t start, end;
   start = clock();
   int temp,i,j;
   for(i=1; i<n; i++)
   {
      temp=a[i];
      j=i-1;
      while(j>=0 && a[j]>temp)
      {
         a[j+1]=a[j];
         j=j-1;
      }
      a[j+1]=temp;
   }
   end = clock();
   totaltime=((double) (end - start))/ CLOCKS_PER_SEC;
```

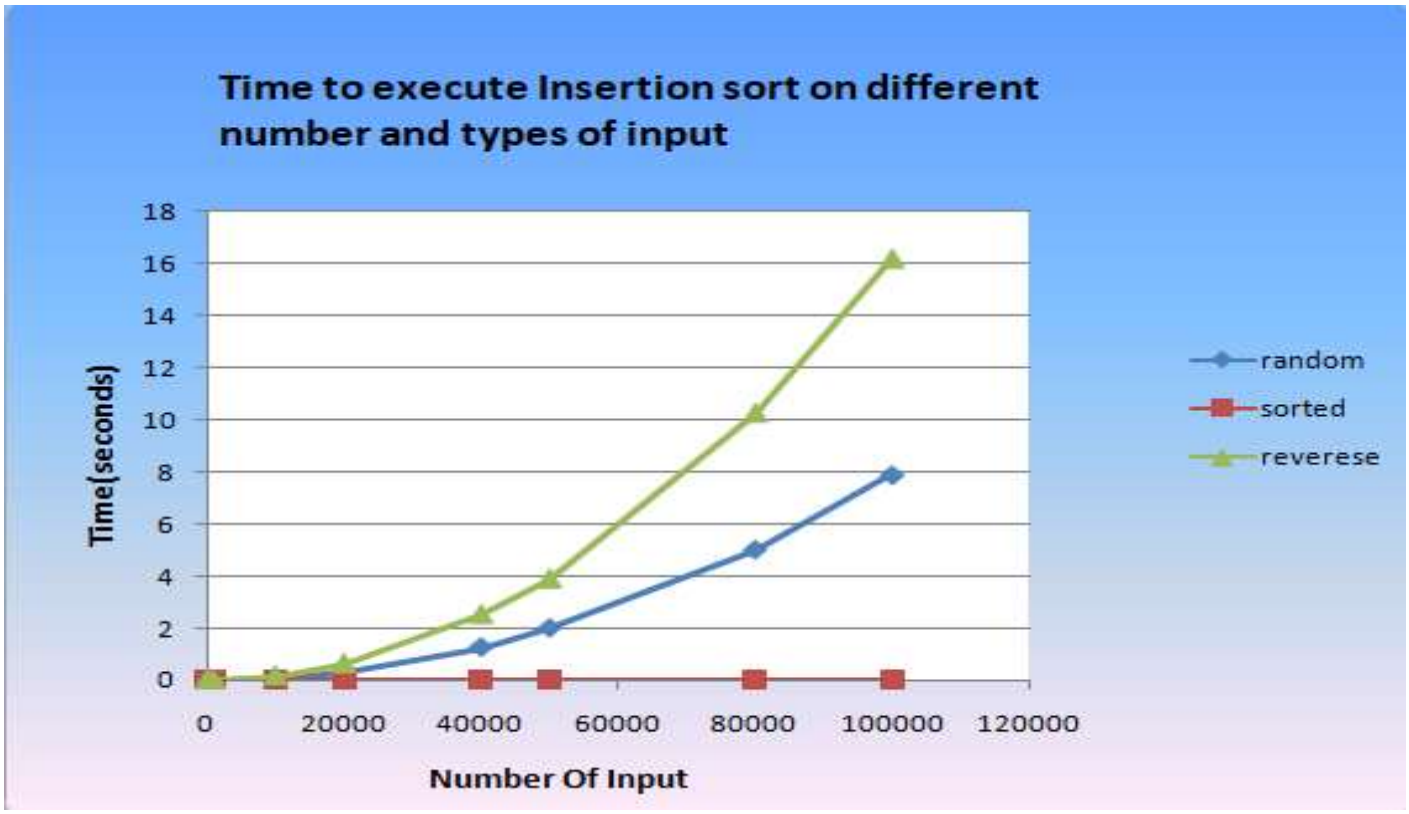
```
}


void print(int *a,int n)
{
   int i,p;
   printf("\nSorted array is:\n ");
   for (i=0; i<10; i++)
   {
      printf(" %d",a[i]);
   }
   if(n>10)
   {
      printf(".....");
      p=n-10;
      for (i=p; i<n; i++)
      {
         printf(" %d",a[i]);
      }
   }
}


void main()
{
```

```
    int *a,*b,*c;
    int n,i,j;
    char ch;
    printf("-----------------------------------------------insertion Sort-----------------------------------
----------\n");
    do
    {
        printf("\nEnter the range of  elements in array :");
        scanf("%d",&n);
        a= (int*)malloc(n * sizeof(int));
        b= (int*)malloc(n * sizeof(int));
        c= (int*)malloc(n * sizeof(int));

        printf("\n-------------------------------Sorting of Random numbers-------------------------
\n");
        for (i = 0; i < n; i++)
        {
            a[i] = rand();
        }
        insertion(a,n);
        print(a,n);
        printf("\n total time in sorting: %f",totaltime);
        printf(" sec\n");

        printf("\n------------------Sorting of numbers which are sorted in ascending order-----------
```

```
-\n");
        for (i = 0; i < n; i++)
        {
            b[i] = a[i];
        }
        /*printf("\n sorted array input\n");
        for (i = 0; i < n; i++)
        {
            printf(" %d",b[i]);
        }*/
        insertion(b,n);
        print(b,n);
        printf("\n total time in sorting: %f",totaltime);
        printf(" sec\n");

        printf("\n------------------Sorting of numbers which are sorted in descending order----------
-\n");
        for (i=0,j=n-1; i < n; i++,j--)
        {
            c[i] = b[j] ;
        }
        /*printf("\nReversed array input\n");
        for (i = 0; i < n; i++)
        {
```

```
        printf(" %d",c[i]);
    }*/
    insertion(c,n);
    print(c,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");

    printf("\n Do you want to continue ? Press 'y' to continue:");
    fflush(stdin);
    scanf("%c",&ch);
    free(a);
  }
  while((ch=='y') || (ch=='Y'));
}
```

Outcome:



Observation:

- For reversed sorted input, insertion sort algorithm takes largest amount of time for sorting.
- For random input, insertion sort algorithm takes moderate amount of time for sorting.
- For sorted input, insertion sort algorithm takes lowest amount of time for sorting.

Time Complexity:

- Best case:O(n)
- Average case: $\Theta (n^2)$
- Worst case: $\Omega(n^2)$

❖ Implementation and analysis of Quicksort algorithm

```
// Quick sort
#include<stdio.h>
#include<time.h>
double totaltime=0;
void quick(int a[],int low,int high);
int partition(int a[], int low,int high);
void swap(int a[],int *,int *);
void quick(int a[],int low,int high)
{
   clock_t start, end;
   start = clock();
   //printf("\n%d",stime);
   int mid;
   if(low<high)
   {
      mid=partition(a,low,high);
      quick(a,low,mid-1);
      quick(a,mid+1,high);
   }
    end = clock();
   //printf("\n%d",etime);
   totaltime=((double) (end - start)) / CLOCKS_PER_SEC;
```

```
}

int partition(int a[],int low,int high)
{
   int pivot,i,j;
   pivot=a[low];
   i=low;
   j=high;
   while(i<=j)
   {
      while(a[i]<=pivot)
      i++;
      while(a[j]>pivot)
         j--;
      if(i<j)
      swap(a,&i,&j);
   }
   swap(a,&low,&j);
   return j;
}

void swap(int a[],int *i,int *j)
{
```

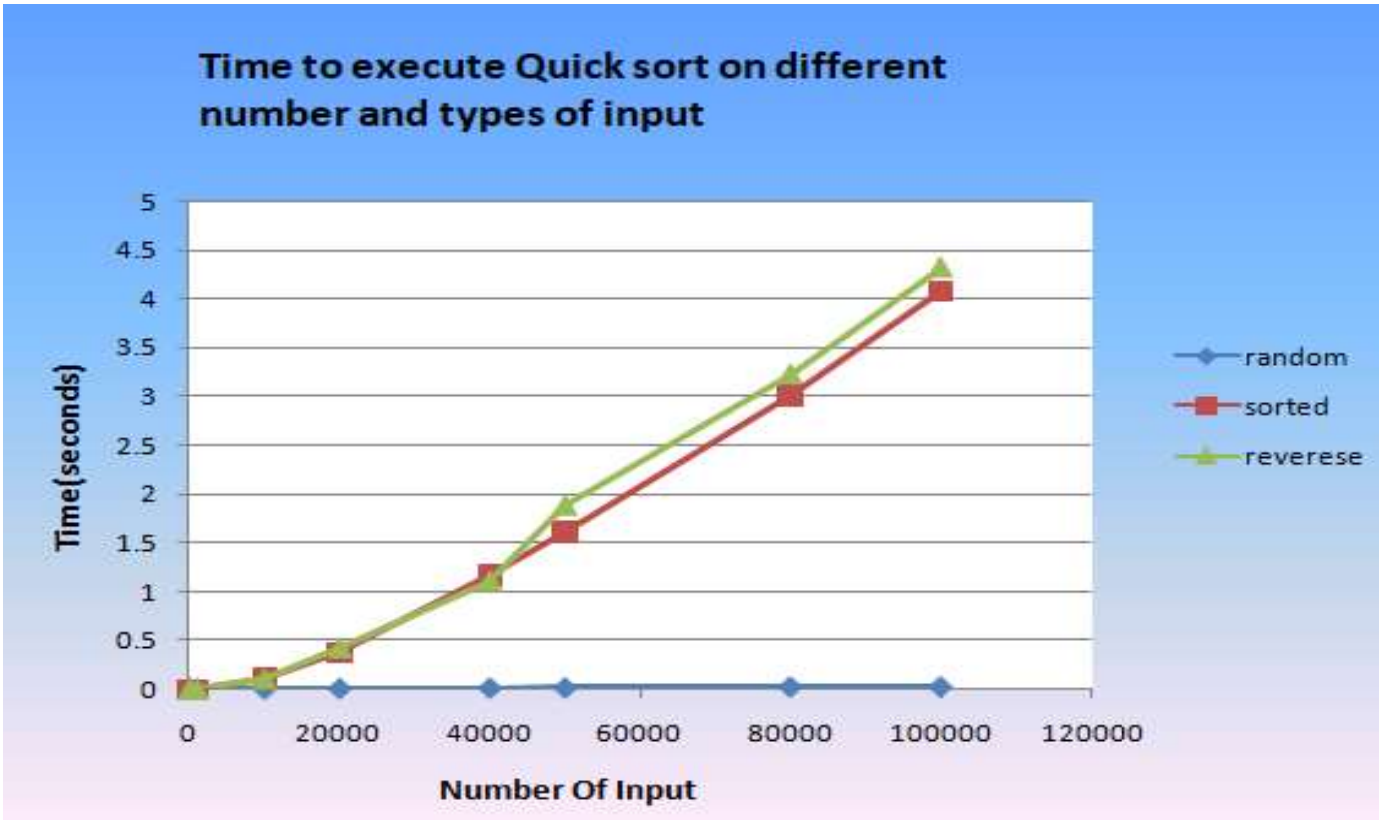
```
    int temp;
    temp=a[*i];
    a[*i]=a[*j];
    a[*j]=temp;
}

void print(int *a,int n)
{
    int i,p;
    printf("\nSorted array is:\n ");
    for (i=0; i<10; i++)
    {
        printf(" %d",a[i]);
    }
    if(n>10)
    {
        printf(".....");
        p=n-10;
        for (i=p; i<n; i++)
        {
            printf(" %d",a[i]);
        }
    }
```

```
}


void main()
{
   int *a,*b,*c;
   int n,i,j;
   char ch;
   printf("-----------------------------------------------Quick Sort-----------------------------------
---------------\n");


   do{
   printf("\nEnter the range of  elements in array :");
   scanf("%d",&n);
   a= (int*)malloc(n * sizeof(int));
   b= (int*)malloc(n * sizeof(int));
   c= (int*)malloc(n * sizeof(int));


   printf("\n------------------------------Sorting of Random numbers-------------------------\n");
   for (i = 0; i < n; i++)
   {
      a[i] = rand();
   }
   quick(a,0,n-1);
   print(a,n);
```

```
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");


    printf("\n------------------Sorting of numbers which are sorted in ascending order------------
\n");
    for (i = 0; i < n; i++)
    {
        b[i] = a[i];
    }
    /*printf("\n sorted array input\n");
    for (i = 0; i < n; i++)
    {
        printf(" %d",b[i]);
    }*/
    quick(b,0,n-1);
    print(b,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");


    printf("\n------------------Sorting of numbers which are sorted in descending order-----------
\n");
    for (i=0,j=n-1; i < n; i++,j--)
    {
        c[i] = b[j] ;
```

```
    }
    /*printf("\nReversed array input\n");
    for (i = 0; i < n; i++)
    {
       printf(" %d",c[i]);
    }*/
    quick(c,0,n-1);
    print(c,n);
    printf("\n total time in sorting: %f",totaltime);
    printf(" sec\n");

    printf("\n Do you want to continue ? Press 'y' to continue:");
    fflush(stdin);
    scanf("%c",&ch);
    free(a);
    }
while((ch=='y') || (ch=='Y'));
}
```

Outcome:



Observation:

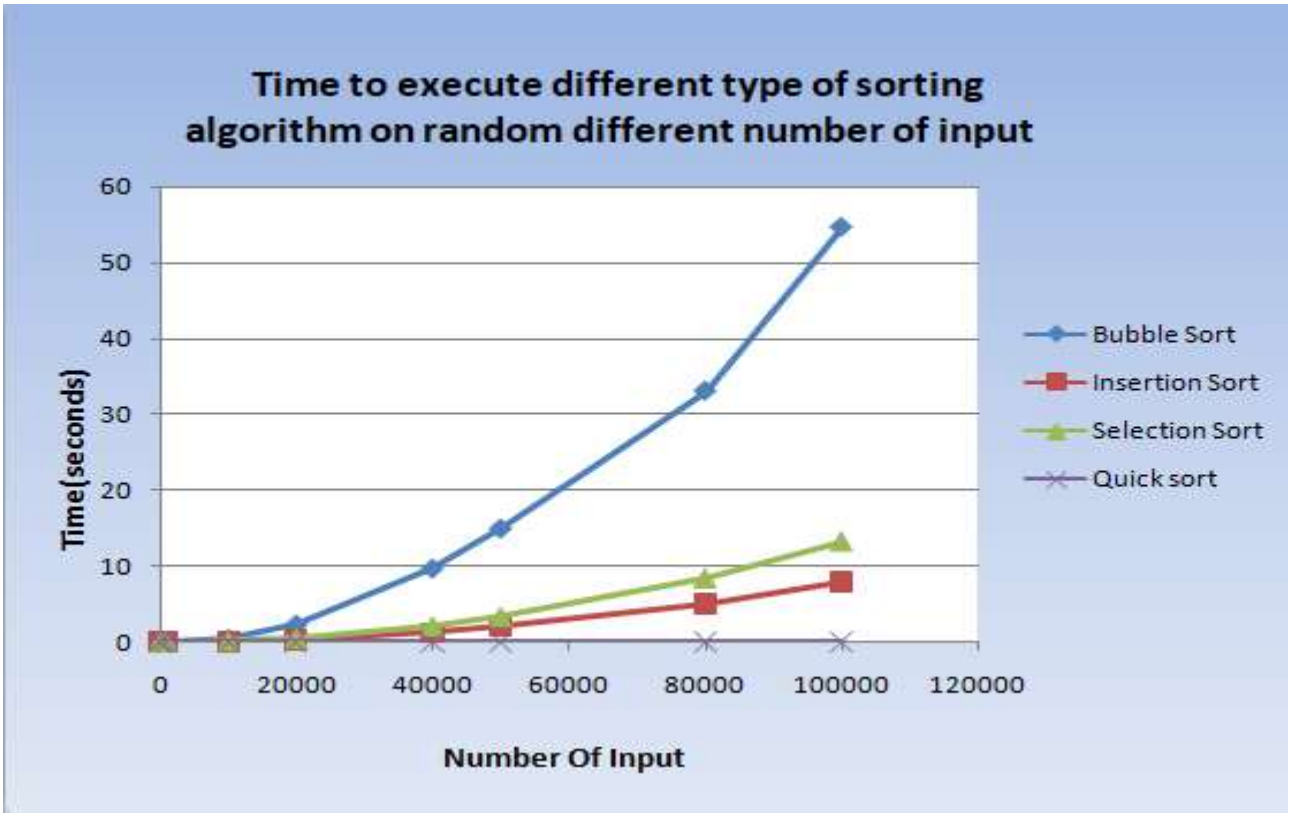
- For reversed sorted input, quick sort algorithm takes largest amount of time for sorting.
- For sorted input, quick sort algorithm takes moderate amount of time for sorting.
- For random input, quick sort algorithm takes lowest amount of time for sorting.

Time Complexity:

- Best case:O(nlogn)
- Average case: Θ (nlogn)
- Worst case: $\Omega(n^2)$

❖ **Comparison of different sorting algorithms on different kind and different numbers of inputs**

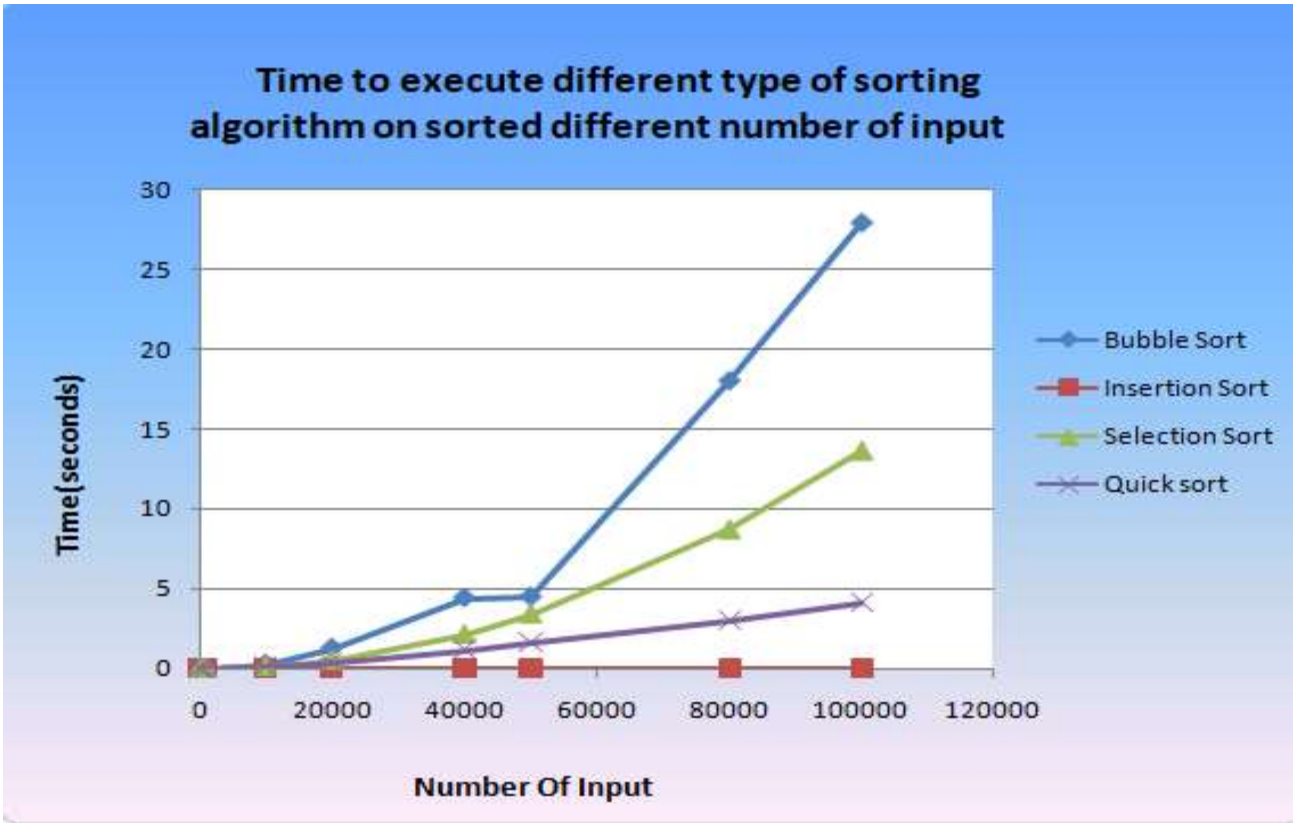
➢ **Random input**



Observations:

- Bubble sort takes largest amount of time for sorting random input
- Quick sort takes smallest amount of time for sorting random input
- Time to execute different type of sorting:
  Quick sort < Insertion sort < Selection sort < Bubble sort
- Hence, Quick sort algorithm is best for random inputs.

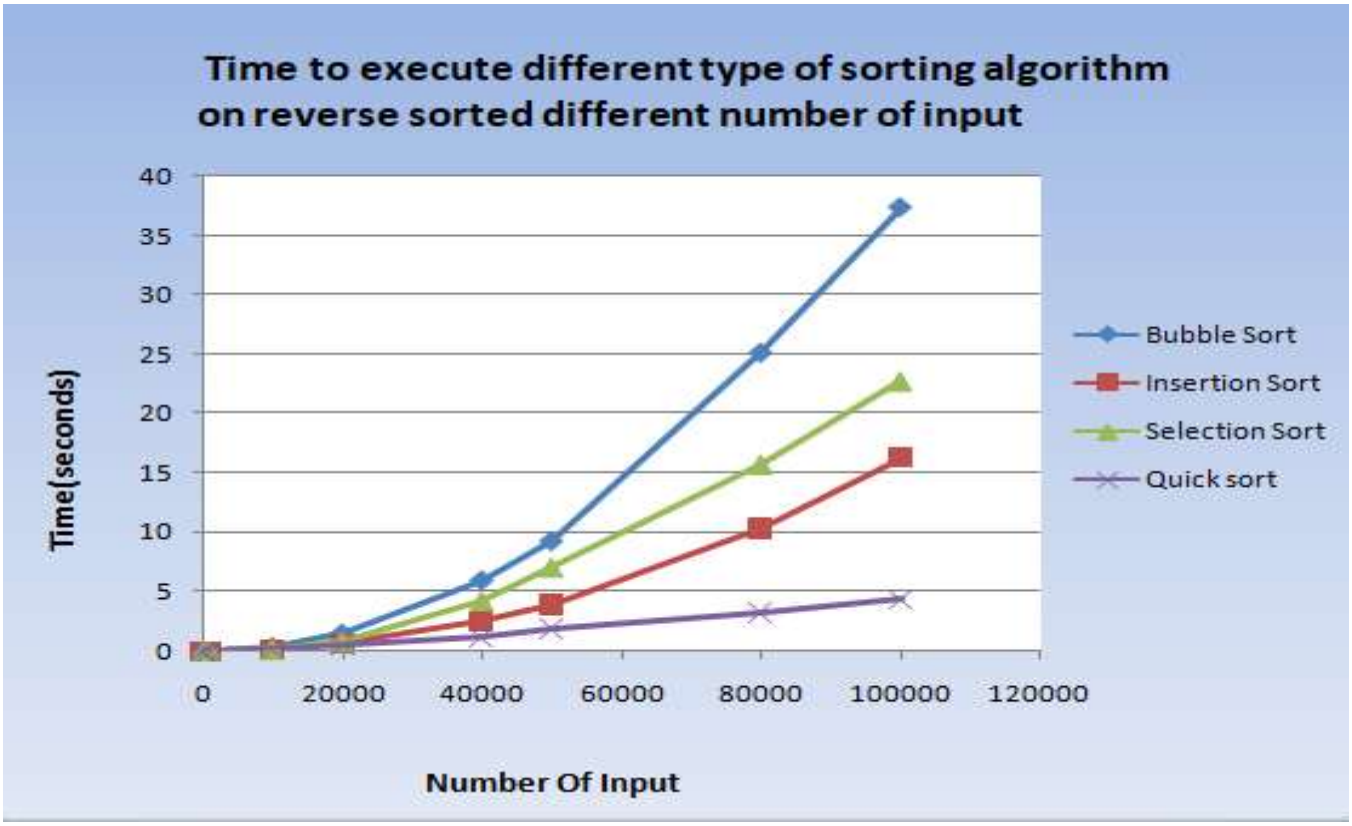
➢ **Sorted input**



Observations:

- Bubble sort takes largest amount of time for sorting sorted input
- Insertion sort takes smallest amount of time for sorting sorted input
- Time to execute different type of sorting:
  Insertion sort <Quick sort < Selection sort < Bubble sort
- Hence, Insertion sort algorithm is best for sorted inputs.

➢ **Reversed input**



Observations:

- Bubble sort takes largest amount of time for sorting reversed input
- Quick sort takes smallest amount of time for sorting reversed input
- Time to execute different type of sorting:
  Quick sort < Insertion sort <Selection sort < Bubble sort
- Hence, Quick sort algorithm is best for reversed input.