CHAPTER 5

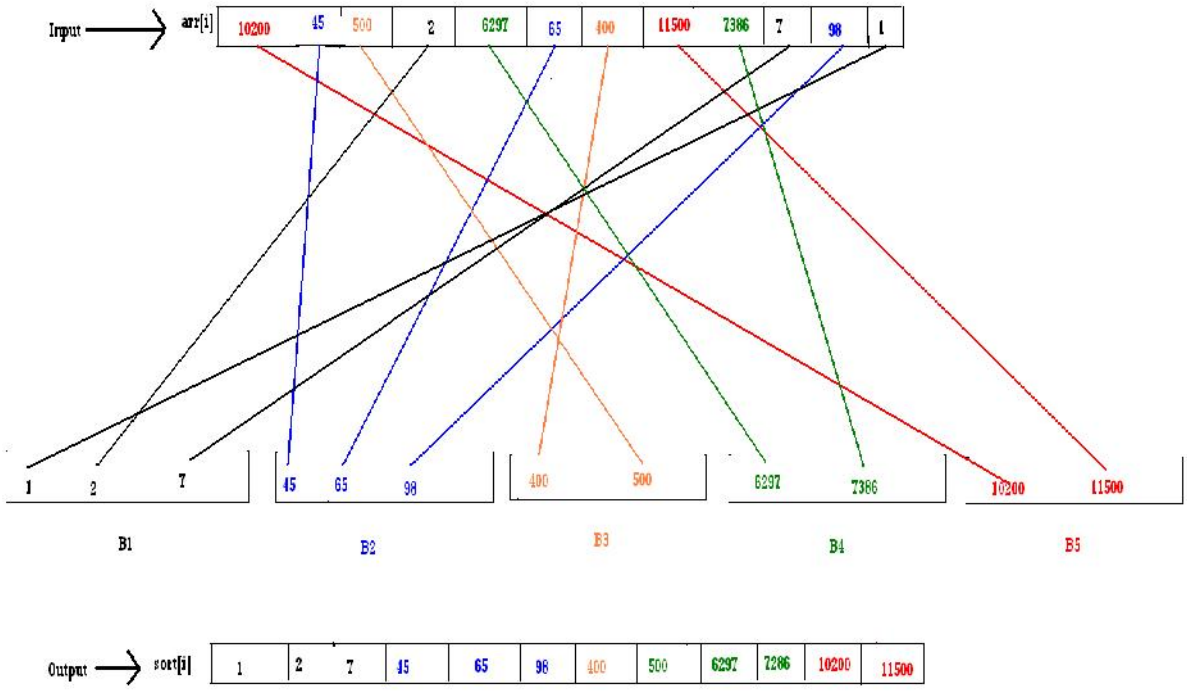
PROPOSED SORT

IN THIS CHAPTER

5. 1. OUR PROPOSED ALGORITHM

5. 1.1. DIGIT SORT

IDEA



In the above figure the idea behind digit sort is illustrated i.e. if the input elements are of different digit elements then according to their digit it will be inserted. At the time of inserting an element into the corresponding digit bucket it will see that the bucket is empty or not if the bucket is empty then the element is inserted into the bucket, Otherwise it compare the element with other elements one by one and break the comparison when the element is inserted into its proper location.

After inserting the elements into its proper position each digit buckets are printed in an ascending order.

At last we get the sorted list.

ALGORITHM

Step 1: Start

Step 2: Initialize pointer array of unsigned integer type variables to dynamically allocate memory i.e; **arrin, arrout, b1, b2, b3, b4, b5** also initialize unsigned integer type variables i.e; **count1, count2, count3, count4, count5, n, no, count, i, j, k=0.**

Step 3: TAKE 'n' number of elements from user.

Step 4: TAKE 'n' number of input elements from user between 0 to 32767 range.

IF arrin[i] is greater than MAXD **THEN**,

! INPUT ELEMENT IS OUT OF RANGE.

EXIT

ELSE

TAKE input from user into arrin[i]

BEFORE SORTING ELEMENTS ARE:

for i=0, i **less than** 'n', **increment** i

PRINT arrin[i]

Step 5: TO COUNT NUMBER OF DIGITS

FOR(i=0, i **less than** 'n', **increment** i)

count=0.

no=arrin[i].

DO(no=no **divided by** 10

Count **increment**)

Check (no **not equal to** 0)

Step 6: GO TO CORRESPONDING DIGIT BUCKET

IF (COUNT=1)

PUT THE ELEMENT INTO b1

b1[count1]=arrin[i]

FOR(j=count1, j **is greater than** 0, j **decrement**)

IF(b1[j-1] is **less than equal to** b1[j])

BREAK.

ELSE

Temp=b1[j-1]

b1[j-1]=b1[j]

b1[j]=temp

Count1 increment

ELSE IF (COUNT=2)

PUT THE ELEMENT INTO b2

b2[count2]=arrin[i]

FOR(j=count2, j **is greater than** 0, j **decrement**)

IF(b2[j-1] **is less than equal to** b2[j])

BREAK.

ELSE

Temp=b2[j-1]

b2[j-1]=b2[j]

b2[j]=temp

Count2 increment

ELSE IF (COUNT=3)

PUT THE ELEMENT INTO b3

b3[count3]=arrin[i]

FOR(j=count3, j **is greater than** 0, j **decrement**)

IF(b3[j-1] **is less than equal to** b3[j])

BREAK.

ELSE

Temp=b3[j-1]

b3[j-1]=b3[j]

b3[j]=temp

Count3 increment

ELSE IF (COUNT=4)

PUT THE ELEMENT INTO b4

b4[count4]=arrin[i]

FOR(j=count4, j **is greater than** 0, j **decrement**)

IF(b4[j-1] **is less than equal to** b4[j])

BREAK.

ELSE

Temp=b4[j-1]

b4[j-1]=b4[j]

b4[j]=temp

Count4 increment

ELSE IF (COUNT=5)

PUT THE ELEMENT INTO b5

b5[count5]=arrin[i]

FOR(j=count5, j **is greater than** 0, j **decrement**)

IF(b5[j-1] **is less than equal to** b5[j])

BREAK.

ELSE

Temp=b5[j-1]

b5[j-1]=b5[j]

b5[j]=temp

Count5 increment

Step 7: TAKE ALL THE ELEMENTS OF EVERY BUCKET INTO ONE ARRAY.

FOR(j=0, j **is less than** count1, j **increment)**

arrout[k **increment**]=b1[j].

FOR(j=0, j **is less than** count2, j **increment**)

arrout[k **increment**]=b2[j].

FOR(j=0, j **is less than** count3, j **increment**)

arrout[k **increment**]=b3[j].

FOR(j=0, j **is less than** count4, j **increment**)

arrout[k **increment**]=b4[j].

FOR(j=0, j **is less than** count5, j **increment**)

arrout[k **increment**]=b5[j].

Step 8: AFTER SORTING ELEMENTS ARE:

FOR(i=0, i **less than** n, i **increment**)

PRINT arrout[i]

Step 9: STOP

COMPLEXITY ANALYSIS

Best Case : $O(n^2)$
Worst Case : $O(n^2)$
Average Case : $O(n^2)$

ADVANTAGE

- If the input elements are of different digits then the sorting algorithm is useful.
- If the all input elements are of different digits then the comparison will decrease.

DISADVANTAGE

- If all the input elements are of same digit then the complexity of the program will increase and it is not applicable.

'C' CODING FOR DIGIT SORT

//Digit sort(Using Dynamic allocation Sort Positive Elements Only)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
#define MAX 32767
void main()
{
unsigned int * arrin,* arrout;
unsigned int * b1,* b2,* b3,* b4,* b5;
unsigned int count1=0,count2=0,count3=0,count4=0,count5=0;
unsigned int n,no,count,i,j,k=0;
        clrscr();
        printf("\n\t\t\t\tDIGIT SORT");
        printf("\n\t\t\t**********\n\n\n");
        printf("\n\nENTER THE NUMBER OF ELEMENTS:");
        scanf("%d",&n);

        arrin=(unsigned int *) malloc(n *sizeof(unsigned int));
```

```
b1=(unsigned int *) malloc(n *sizeof(unsigned int));
b2=(unsigned int *) malloc(n *sizeof(unsigned int));
b3=(unsigned int *) malloc(n *sizeof(unsigned int));
b4=(unsigned int *) malloc(n *sizeof(unsigned int));
b5=(unsigned int *) malloc(n *sizeof(unsigned int));
arrout=(unsigned int *) malloc(n *sizeof(unsigned int));

printf("\nENTER THE INPUT ELEMENTS BETWEEN 0 TO 32767:\n");
for(i=0;i<n;i++)
{
        scanf("%d",&arrin[i]);
        if((arrin[i])>MAX)
        {
        printf("!OUT OF RANGE");
        getch();
        exit(1);
        }
}
printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i=0;i<n;i++)
        printf("%d ",arrin[i]);
for(i=0;i<n;i++)
{
count=0;
no=arrin[i];
do{
/*x=no%10;*/
no=no/10;
count++;
}while(no!=0);

        if(count==1)
        {
                b1[count1]=arrin[i];
                for(j=count1;j>0;j--)
                {
                        if(b1[j-1]<=b1[j])
                        {
                                break;
                        }
                        else
```

```
                {
                        int temp=b1[j-1];
                        b1[j-1]=b1[j];
                        b1[j]=temp;
                 }
        }
        count1++;
}
else if(count==2)
{
        b2[count2]=arrin[i];
        for(j=count2;j>0;j--)
        {
                if(b2[j-1]<=b2[j])
                {
                        break;
                }
                else
                {
                        int temp=b2[j-1];
                        b2[j-1]=b2[j];
                        b2[j]=temp;
                 }
        }
        count2++;
}
else if(count==3)
{
        b3[count3]=arrin[i];
        for(j=count3;j>0;j--)
        {
                if(b3[j-1]<=b3[j])
                {
                        break;
                }
                else
                {
                        int temp=b3[j-1];
                        b3[j-1]=b3[j];
                        b3[j]=temp;
                 }
```

```
                }
                count3++;
        }
        else if(count==4)
        {
                b4[count4]=arrin[i];
                for(j=count4;j>0;j--)
                {
                        if(b4[j-1]<=b4[j])
                        {
                                break;
                        }
                        else
                        {
                                int temp=b4[j-1];
                                b4[j-1]=b4[j];
                                b4[j]=temp;
                         }
                }
                count4++;
        }
        else if(count==5)
        {
                b5[count5]=arrin[i];
                for(j=count5;j>0;j--)
                {
                        if(b5[j-1]<=b5[j])
                        {
                                break;
                        }
                        else
                        {
                                int temp=b5[j-1];
                                b5[j-1]=b5[j];
                                b5[j]=temp;
                         }
                }
                count5++;
        }
}
```

```
        for(j=0;j<count1;j++)
        arrout[k++]=b1[j];
        for(j=0;j<count2;j++)
        arrout[k++]=b2[j];
        for(j=0;j<count3;j++)
        arrout[k++]=b3[j];
        for(j=0;j<count4;j++)
        arrout[k++]=b4[j];
        for(j=0;j<count5;j++)
        arrout[k++]=b5[j];
printf("\n\nAFTER SORTING ELEMENTS ARE: ");
        for(i=0;i<n;i++)
                {
                 printf("%d ",arrout[i]);
                }
getch();
}
```

CHAPTER 6

CODING IN 'C'

IN THIS CHAPTER

6. 1. CODES FOR VARIOUS SORTS

6. 1.1. BUBBLE SORT

```
#include<stdio.h>
void main()
{
int A[200],N,Temp,i,j;
clrscr();
printf("\n\t\t\t\tBUBBLE SORT");
printf("\n\t\t\t\t**********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &N);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(i=0;i<N;i++)
{
    scanf("%d",&A[i]);
}
printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i=0; i<N; i++)
    printf("%d ",A[i]);
for(i=0;i<N-1;i++)
    for(j=0;j<N-i;j++)
    if(A[j]>A[j+1])
        {
        Temp = A[j];
        A[j] = A[j+1];
        A[j+1] = Temp;
        }
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for(i=0; i<N; i++)
    printf("%d ",A[i]);
getch();
}
```

6. 1.2. BUCKET SORT

```
#include <conio.h>
#include <stdio.h>
void main() {
int unsorted[50] , bucket[10][50]={{0}} , sorted[50] ;
int   j , k , m , p , flag = 0, num, N;
clrscr();
printf("\n\t\t\t\tBUCKET SORT");
printf("\n\t\t\t\t**********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &N);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(k=0 ;  k < N ; k++){
scanf("\n%d",&num);
sorted[k] = unsorted[k] = num;
}
for(p=1; flag != N ; p*=10 {
flag = 0;
for(k=0;k<N;k++) {
bucket[(sorted[k]/p)%10][k] = sorted[k];
if ( (sorted[k]/p)%10  == 0 ) {
      flag++;
      }
}
if (flag == N) {
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for(j=0 ;  j < N ; j++) {
printf("%d\t", sorted[j]);
}
printf("\n");
getch();
}
for(j=0,m=0;j<10;j++){
for(k=0;k<N;k++){
if( bucket[j][k] > 0 ){
sorted[m] = bucket[j][k];
bucket[j][k] = 0 ;
m++;
} } } } }
```

6. 1.3. COCKTAIL SORT

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void main() {
int a[MAX],b[MAX];
int n,i,j,pass,sw=1,temp;
clrscr();
printf("\n\t\t\t\tCOCKTAIL SORT");
printf("\n\t\t\t\t**************\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(i=0;i<n;i++) {
    scanf("%d",&a[i]);
    b[i]=a[i];
}
printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n\nAFTER SORTING ELEMENTS ARE:");
sw=1;
for(i=0;i<n-1 && sw==1;i++) {
sw=0;
for(j=0;j<n-1-i;j++) {
if(b[j]>b[j+1]) {
    temp=b[j+1];
    b[j+1]=b[j];
    b[j]=temp;
    sw=1;
}
if(b[n-1-j]<b[n-2-j]) {
    temp=b[n-2-j];
    b[n-2-j]=b[n-1-j];
    b[n-1-j]=temp;
    sw=1;
} } }
for(j=0;j<n;j++)
printf("%d ",b[j]);
printf("\n\nNUMBER OF PASSES REQUIRED:%d",i);
getch();
}
```

6. 1.4. COMB SORT

```
#include<stdio.h>
#include<conio.h>
void Combsort11(int a[], int nElements) {
int i=0,j=0,k=0,gap, swapped = 1;
int temp=0;
gap = nElements;
while (gap > 1 || swapped == 1) {
gap = gap/1.3;
//printf("gap=%d\n",gap);
if (gap < 1)gap=1;
swapped = 0;
for (i = 0,j=gap;j<nElements;i++,j++) {
        //printf("%d%d--",i,j);
        if(a[i]>a[j]) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
        swapped = 1;
        } } } }
void main() {
int a[100];
int i,n;
clrscr();
printf("\n\t\t\t\tCOMB SORT");
printf("\n\t\t\t\t*********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(i=0;i<n;i++) {
scanf("%d",&a[i]);
}
printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i=0; i<n; i++) {
printf("%d ",a[i]);
}
Combsort11(a,n);
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for(i=0;i<n;i++) {
printf("%d ",a[i]);
}
getch();
}
```

6. 1.5. COUNTING SORT

```
#include<stdio.h>
#include<conio.h>
int Counting_sort(int A[], int k, int n){
    int i, j;
    int B[15], C[100];
    for(i = 0; i <= k; i++)
        C[i] = 0;
    for(j =1; j <= n; j++)
        C[A[j]] = C[A[j]] + 1;
    for(i = 1; i <= k; i++)
        C[i] = C[i] + C[i-1];
    for(j = n; j >= 1; j--) {
        B[C[A[j]]] = A[j];
        C[A[j]] = C[A[j]] - 1;
        }
    printf("\nAFTER SORTING ELEMENTS ARE:");
    for(i = 1; i <= n; i++)
        printf("%d ",B[i]);
return 0;
}
void main() {
    int n,i,k = 0, A[15];
    clrscr();
    printf("\n\t\t\t\tCOUNTING SORT");
    printf("\n\t\t\t\t**********\n\n\n");
    printf("\n\nENTER THE NUMBER OF ELEMENTS:");
    scanf("%d", &n);
    printf("\nAFTER SORTING ELEMENTS ARE:");
    for ( i = 1; i <= n; i++) {
        scanf("%d ",&A[i]);
    if(A[i] > k) {
        k = A[i];
    }
}
    Counting_sort(A, k, n);
getch();
}
```

6. 1.6. HEAP SORT

```
#include<stdio.h>
#include<conio.h>
int hsort[25],n,i;
void adjust(int,int);
void heapify();
void main(){
int temp;
clrscr();
printf("\n\t\t\t\tHEAP SORT");
printf("\n\t\t\t\t*********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(i=1;i<=n;i++)
scanf("%d",&hsort[i]);
printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i=1;i<=n;i++)
printf("%d ",hsort[i]);
heapify();
for(i=n;i>=2;i--) {
    temp=hsort[1];
    hsort[1]=hsort[i];
    hsort[i]=temp;
    adjust(1,i-1);
}       printf("\n\nAFTER SORTING ELEMENTS ARE:");
        for(i=1;i<=n;i++)
        printf("%d ",hsort[i]);
        getch();
        }       void heapify() {
                int i;
                for(i=n/2;i>=1;i--)
                adjust(i,n);
                }       void adjust(int i,int n){
                        int j,element;
                        j=2*i;
                        element=hsort[i];
                        while(j<=n) {
                                if((j<n)&&(hsort[j]<hsort[j+1]))
                                j=j++;
                                if(element>=hsort[j])
                        break;
                        hsort[j/2]=hsort[j];
                        j=2*j;
                        }       hsort[j/2]=element;    }
```

6. 1.7. INSERTION SORT

```
#include<stdio.h>
void main()
    {
        int A[20], N, Temp, i, j;
        clrscr();
        printf("\n\t\t\t\tINSERTION SORT");
        printf("\n\t\t\t\t**************\n\n\n");
        printf("\n\nENTER THE NUMBER OF ELEMENTS:");
        scanf("%d", &N);
        printf("\nENTER THE INPUT ELEMENTS:\n\n");
        for(i=0; i<N; i++)
        {
        scanf("\n%d", &A[i]);
        }
        printf("\nBEFOR SORTING ELEMENTS ARE:");
        for(i=0; i<N; i++)
        {
         printf("%d ",A[i]);
        }

        for(i=1; i<N; i++)
        {
                Temp = A[i];
                j = i-1;
                while(Temp<A[j] && j>=0)
                {
                        A[j+1] = A[j];
                        j = j-1;
                }
                A[j+1] = Temp;
        }
        printf("\n\nAFTER SORTING ELEMENTS ARE:");
        for(i=0; i<N; i++)
                printf("%d ", A[i]);
        getch();
    }
```

6. 1.8. MERGE SORT

```
#include <stdio.h>
#include <stdlib.h>
#define MAXARRAY 100
void mergesort(int a[], int low, int high);
void main() {
int array[MAXARRAY],n;
int i = 0;
clrscr();
/* reading the elements form the users*/
printf("\n\t\t\t\tMERGE SORT");
printf("\n\t\t\t\t**********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE INPUT ELEMENTS:\n\n");
for(i = 0; i < n; i++ ){
        scanf("%d",&array[i]);
}
/* array before mergesort */
 printf("\nBEFORE SORTING ELEMENTS ARE:");
for(i = 0; i < n; i++)
    printf(" %d", array[i]);
    printf("\n");
mergesort(array, 0, n - 1);
/* array after mergesort */
printf("\nAFTER SORTING ELEMENTS ARE:");
for(i = 0; i < n; i++)
    printf(" %d", array[i]);
    printf("\n");
getch();
```

```
}
void mergesort(int a[], int low, int high) {
int i = 0;
int length=high-low+1;
int pivot=0;
int merge1=0;
int merge2=0;
int working[100];
if(low == high)
     return;
pivot  = (low + high) / 2;
mergesort(a, low, pivot);
mergesort(a, pivot + 1, high);
for(i = 0; i < length; i++)
working[i] = a[low + i];
merge1 = 0;
merge2 = pivot - low + 1;
for(i = 0; i < length; i++) {
          if(merge2 <= high - low)
          if(merge1 <= pivot - low)
          if(working[merge1] > working[merge2])
          a[i + low] = working[merge2++];
          else
          a[i + low] = working[merge1++];
          else
          a[i + low] = working[merge2++];
          else
          a[i + low] = working[merge1++];
}
}
```

6. 1.9. QUICK SORT

```
#include "stdio.h"
#include<conio.h>
int split ( int*,int,int) ;
void quicksort (int*,int,int) ;
void main( )
{
    int i,n,arr[100];
    clrscr();
    printf("\n\t\t\t\tQUICK SORT");
    printf("\n\t\t\t\t**********\n\n\n");
    printf("\n\nENTER THE NUMBER OF ELEMENTS:");
    scanf("%d",&n);
    printf("\nENTER THE INPUT ELEMENTS:\n\n");
        for(i=0;i<n;i++)
        {
        scanf("%d",&arr[i]);
        }
    printf("\nBEFORE SORTING ELEMENTS ARE:");
    for(i=0;i<n;i++)
        printf("%d ",arr[i]);
quicksort(arr,0,n) ;
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for(i=0; i<n; i++)
    printf("%d ",arr[i]);
getch();
}
void quicksort ( int a[ ], int lower, int upper )
{
    int i ;
```

```
    if ( upper > lower )
        {
        i = split ( a, lower, upper ) ;
        quicksort ( a, lower, i - 1 ) ;
        quicksort ( a, i + 1, upper ) ;
        }
}
int split ( int a[ ], int lower, int upper )
    {
    int i, p, q, t ;
    p = lower + 1 ;
    q = upper ;
    i = a[lower] ;
    while ( q >= p )
    {
        while ( a[p] < i )
                p++ ;
        while ( a[q] > i )
                q-- ;
        if ( q > p )
         {
         t = a[p] ;
         a[p] = a[q] ;
         a[q] = t ;
         }
    }
    t = a[lower] ;
    a[lower] = a[q] ;
    a[q] = t ;
    return q ;
    }
```

6. 1.10. RADIX SORT

```
#include <stdio.h>
#define MAX 20
#define SHOWPASS
void print(int *a, int n)
{
int i;
for (i = 0; i < n; i++)
     printf("%d\t", a[i]);
}
void radixsort(int *a, int n)
{
       int i, b[MAX], m = 0, exp = 1;
       for (i = 0; i < n; i++)
       {
        if (a[i] > m)
        m = a[i];
       }
       while (m / exp > 0)
       {
        int bucket[10] =
        {
          0
         };
     for (i = 0; i < n; i++)
          bucket[a[i] / exp % 10]++;
     for (i = 1; i < 10; i++)
          bucket[i] += bucket[i - 1];
     for (i = n - 1; i >= 0; i--)
          b[--bucket[a[i] / exp % 10]] = a[i];
```

```
    for (i = 0; i < n; i++)
        a[i] = b[i];
        xp *= 10;
/*  #ifdef SHOWPASS
printf("\n\nNUMBER OF PASS REQUIRED: ");
print(a, n);
#endif  */
}
}
void main()
{
      int arr[MAX];
      int i, n;
      clrscr();
      printf("\n\t\t\t\tRADIX SORT");
      printf("\n\t\t\t\t**********\n\n\n");
      printf("\n\nENTER THE NUMBER OF ELEMENTS < %d: ",MAX);
      scanf("%d", &n);
      printf("\nENTER THE %d INPUT ELEMENTS:\n\n",n);
      for (i = 0; i < n; i++)
         scanf("%d", &arr[i]);
         printf("\nBEFORE SORTING ELEMENTS ARE:");
         print(&arr[0], n);
radixsort(&arr[0], n);
printf("\n\nAFTER SORTING ELEMENTS ARE:");
print(&arr[0], n);
printf("\n");
getch();
}
```

6. 1.11. SELECTION SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int array[100], n, c, d, position,swap;
clrscr();
printf("\n\t\t\t\tSELECTION SORT");
printf("\n\t\t\t\t**************\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE %d INPUT ELEMENTS:\n\n",n);
for ( c = 0 ; c < n ; c++ )
        scanf("%d", &array[c]);
        printf("\nBEFORE SORTING ELEMENTS ARE:");
for ( c = 0 ; c < n ; c++ )
        printf("%d ",array[c]);
for ( c = 0 ; c < ( n - 1 ) ; c++ ) {
        position = c;
for ( d = c + 1 ; d < n ; d++ ) {
    if ( array[position] > array[d] )
    position = d;
    }
    if ( position != c ) {
         swap = array[c];
         array[c] = array[position];
         array[position] = swap;
    }
}
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for ( c = 0 ; c < n ; c++ )
printf("%d ", array[c]);
getch();
}
```

6. 1.12. SHELL SORT

```
#include<stdio.h>
#include<conio.h>
void shellsort(int a[],int n) {
int j,i,k,m,mid;
for(m = n/2;m>0;m/=2) {
for(j = m;j<n;j++) {
for(i=j-m;i>=0;i-=m) {
if(a[i+m]>=a[i])
break;
else
{
mid = a[i];
a[i] = a[i+m];
a[i+m] = mid;
}
} } } }
void main() {
int a[10],i,n;
clrscr();
printf("\n\t\t\t\tSELL SORT");
printf("\n\t\t\t\t*********\n\n\n");
printf("\n\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d",&n);
for(i=0;i<n;i++) {
printf("\nENTER THE %d INPUT ELEMENT:",i+1);
scanf("%d",&a[i]);
}
printf("\nBEROFE SORTING ELEMENTS ARE:");
for(i=0;i<n;i++)
printf("%2d ",a[i]);
shellsort(a,n);
printf("\n\nAFTER SORTING ELEMENTS ARE:");
for(i=0;i<n;i++)
printf("%2d ",a[i]);
getch();
return 0;
}
```

CHAPTER 7

INPUT-OUTPUT DESIGN

IN THIS CHAPTER

7. 1. SCREENSHOTS

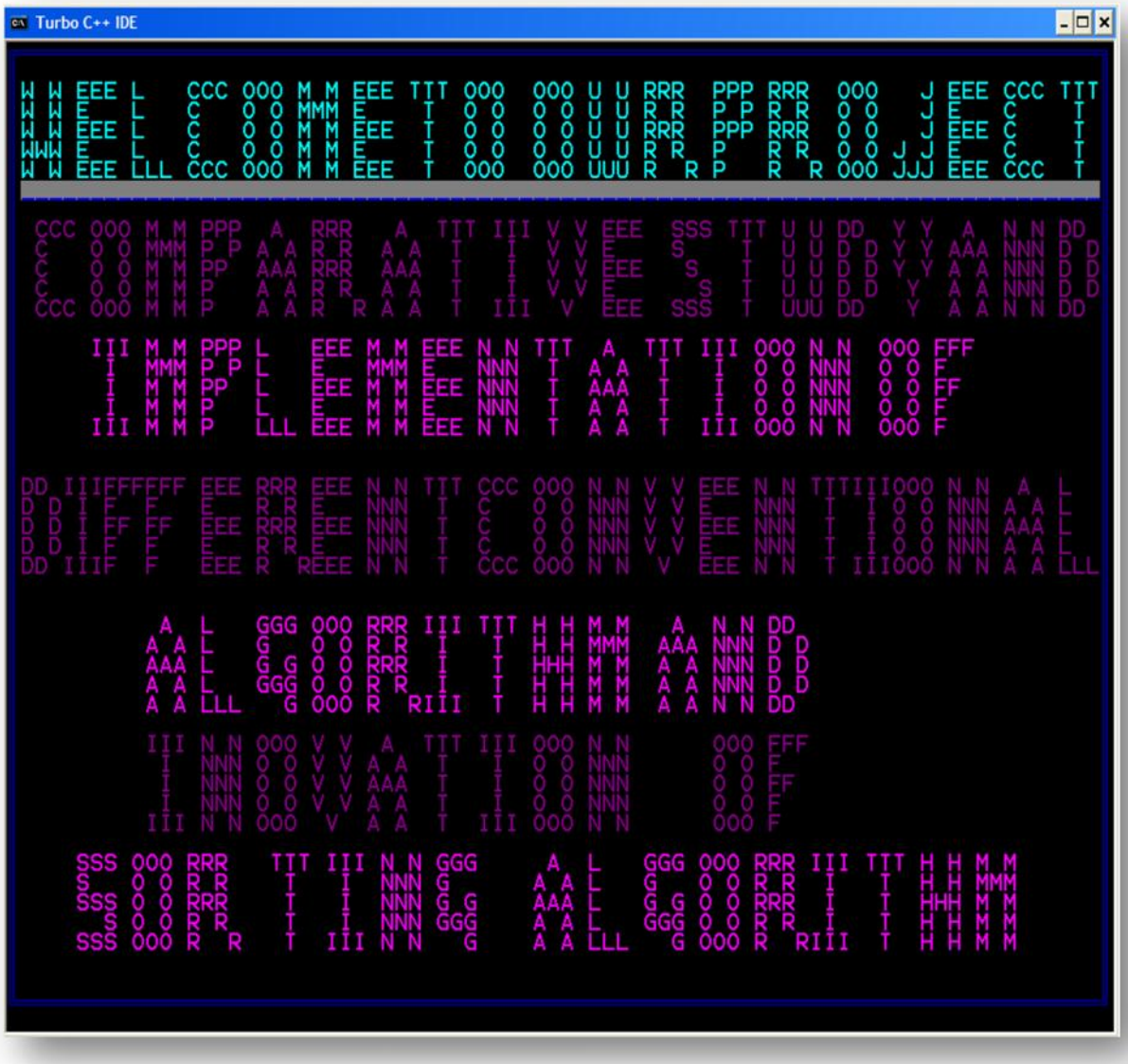
7. 1.1. LOGIN

UNSUCCESSFUL

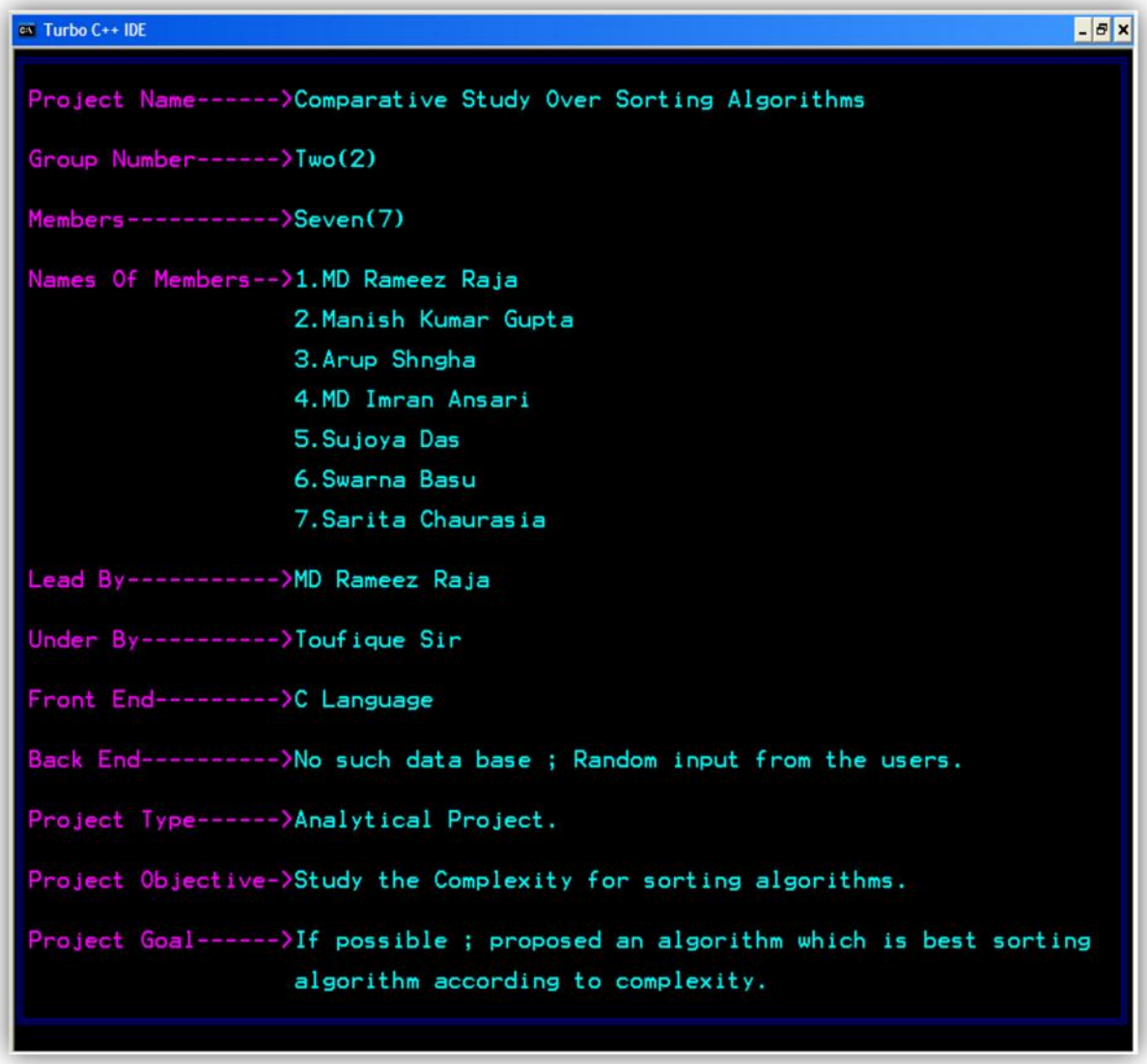
SUCCESSFUL


Turbo C++ IDE
--- WELCOME TO OUR PROJECT ---
COMPARATIVE STUDY OVER SORTING ALGORITHMS
PLEASE ENTER PASSWORD:
CORRECT ! YOU ARE AUTHENTICATED TO USE
PLEASE PRESS ANY KEY..

7. 1.2. WELCOME

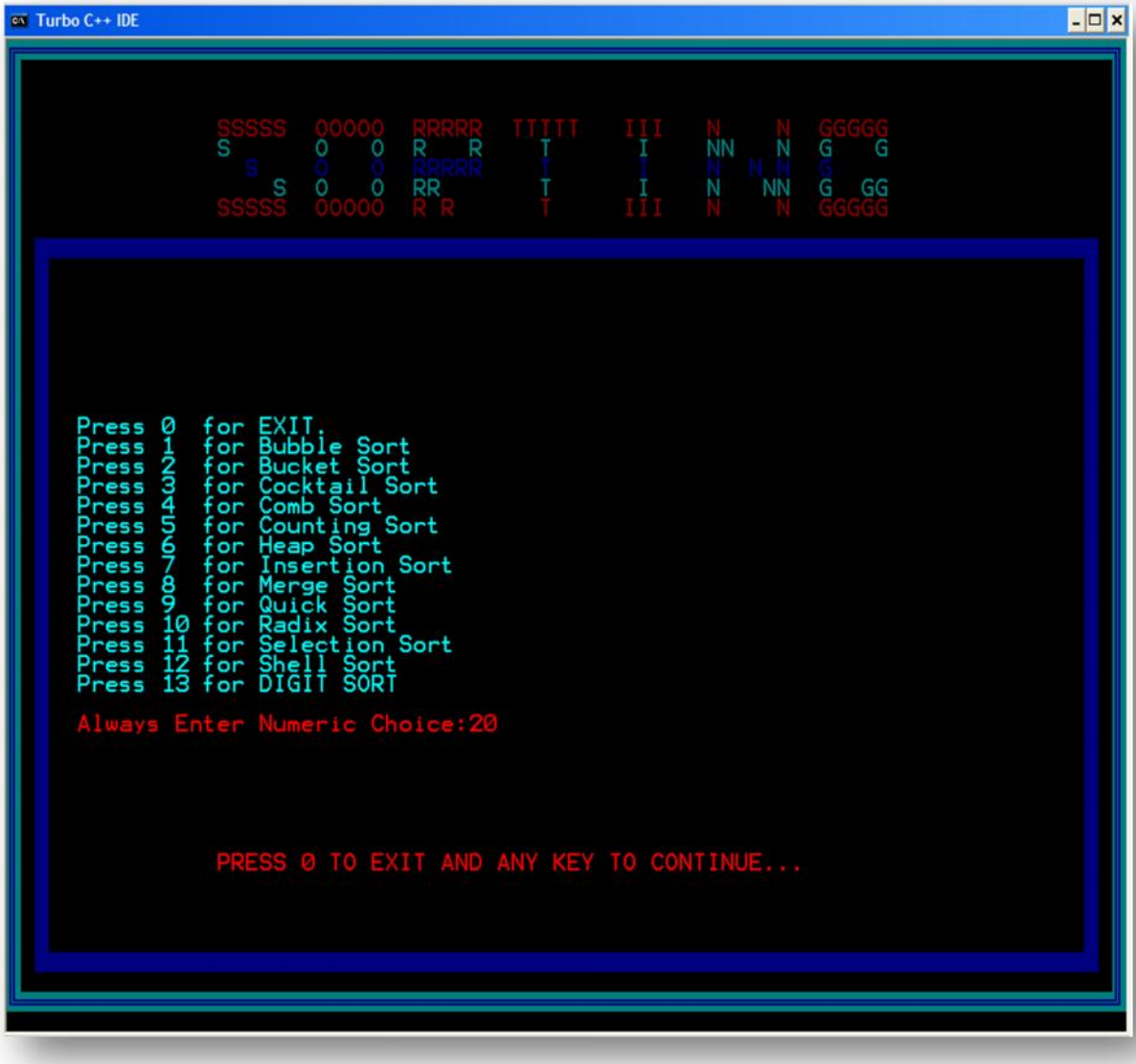


7. 1.3. TEAM MEMBERS NAME

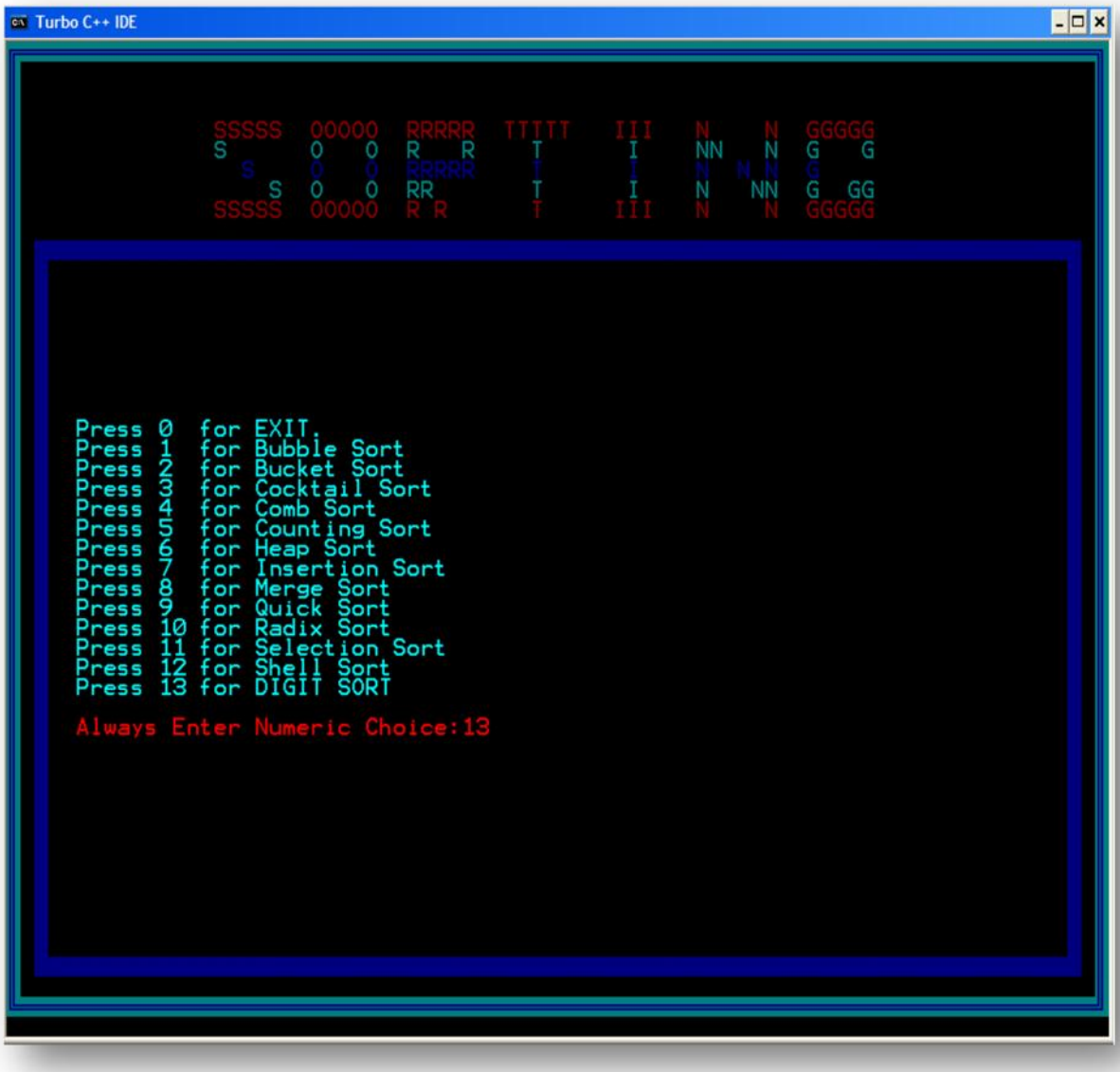


7. 1.4. APPLICATION

ENTERED WRONG CHOICE

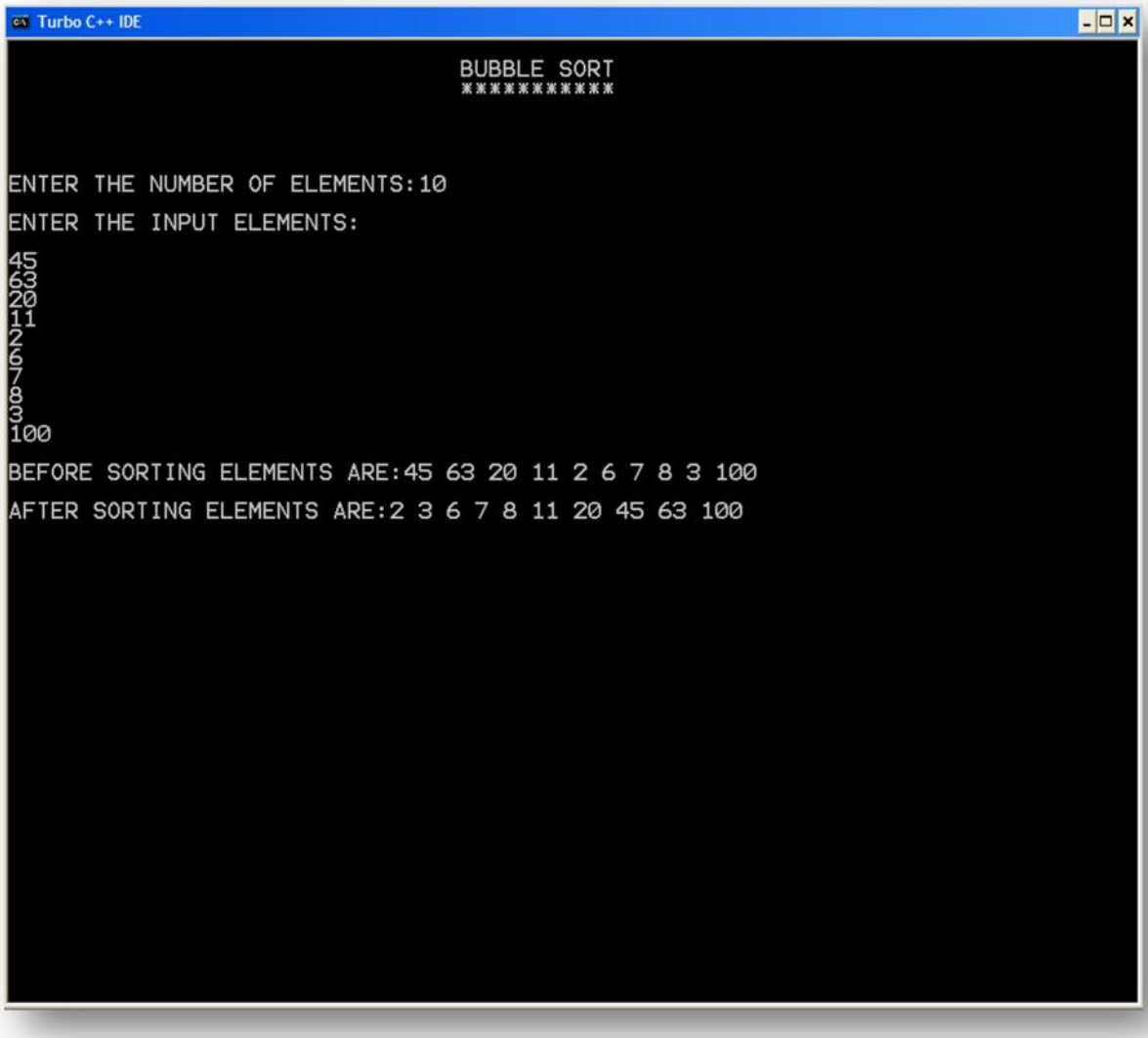


ENTERED RIGHT CHOICE

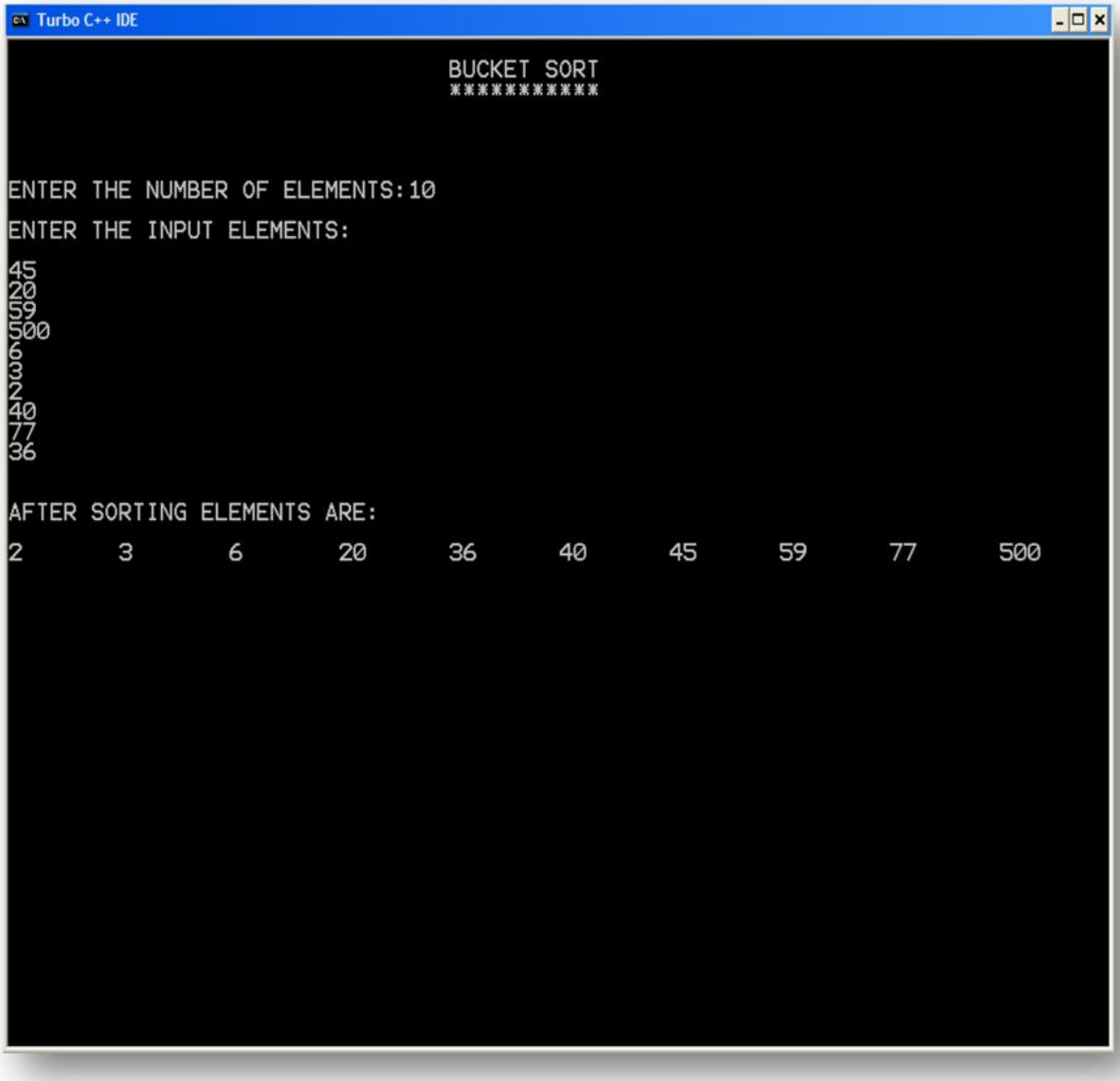


7. 1.5. SORTING ALGORITHMS

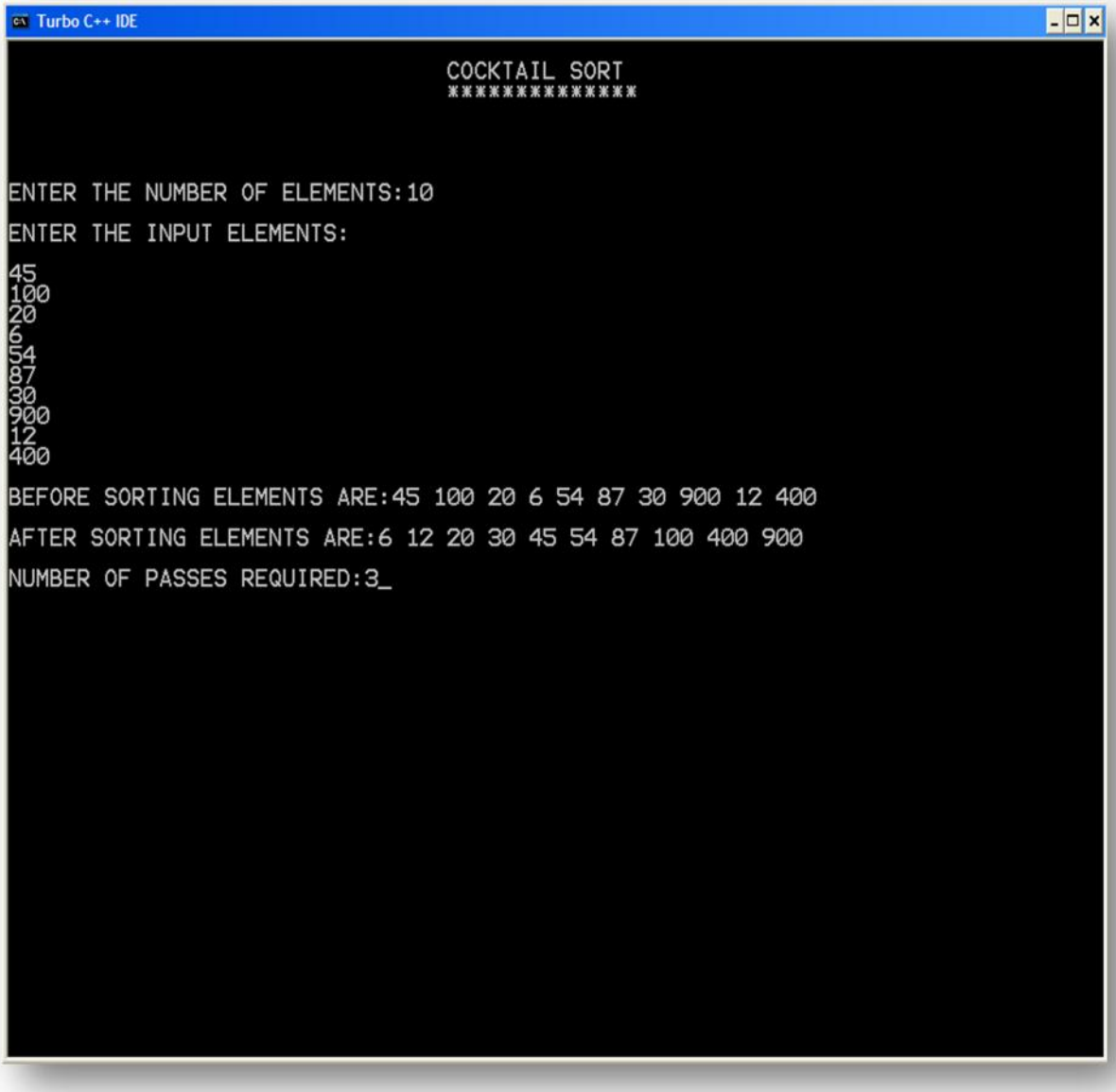
BUBBLE SORT



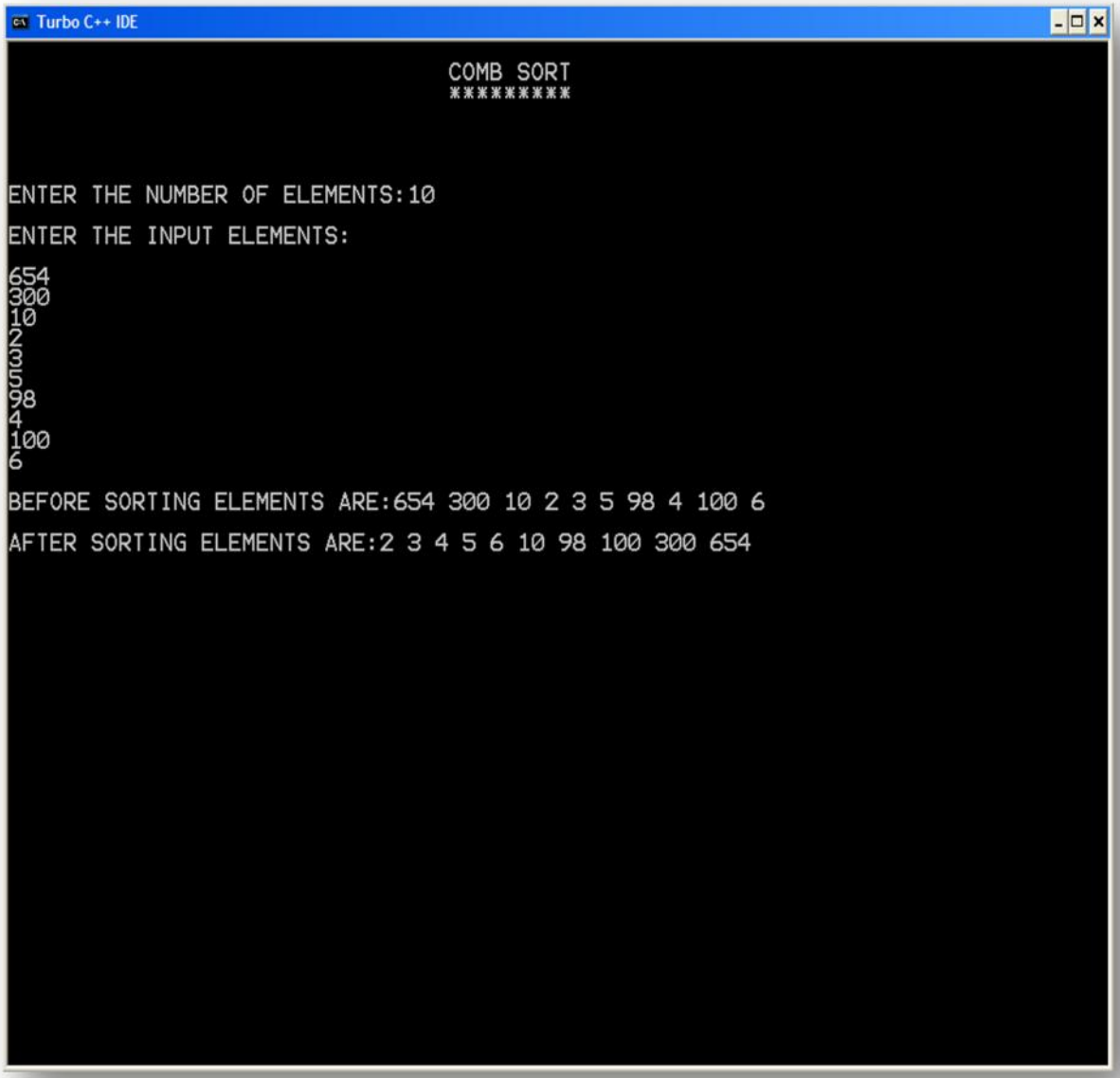
BUCKET SORT


Turbo C++ IDE
BUCKET SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE INPUT ELEMENTS:
45
20
59
500
6
3
2
40
77
36
AFTER SORTING ELEMENTS ARE:
2 3 6 20 36 40 45 59 77 500

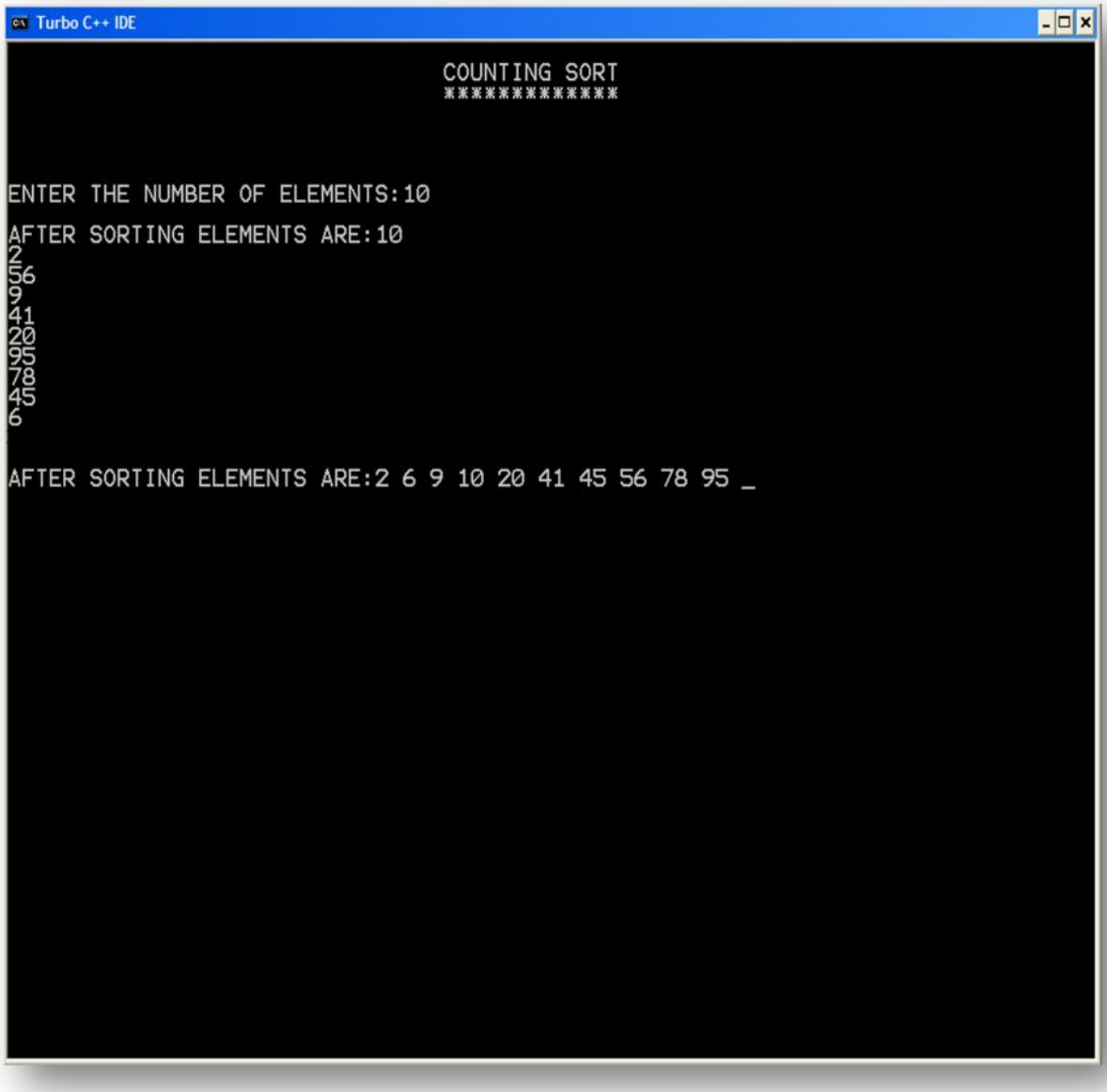
COCKTAIL SORT



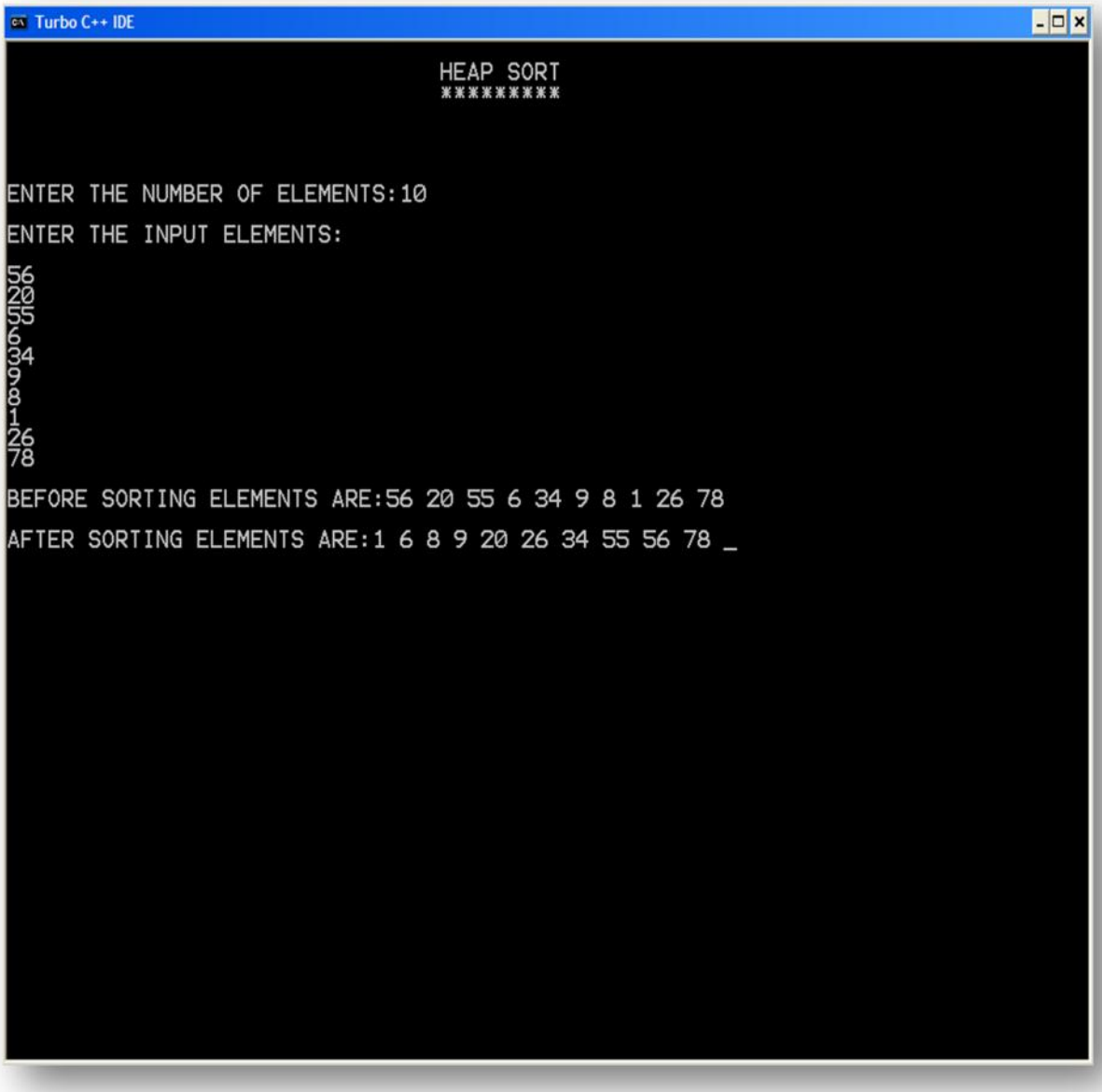
COMB SORT


Turbo C++ IDE
COMB SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE INPUT ELEMENTS:
654
300
10
2
3
5
98
4
100
6
BEFORE SORTING ELEMENTS ARE:654 300 10 2 3 5 98 4 100 6
AFTER SORTING ELEMENTS ARE:2 3 4 5 6 10 98 100 300 654

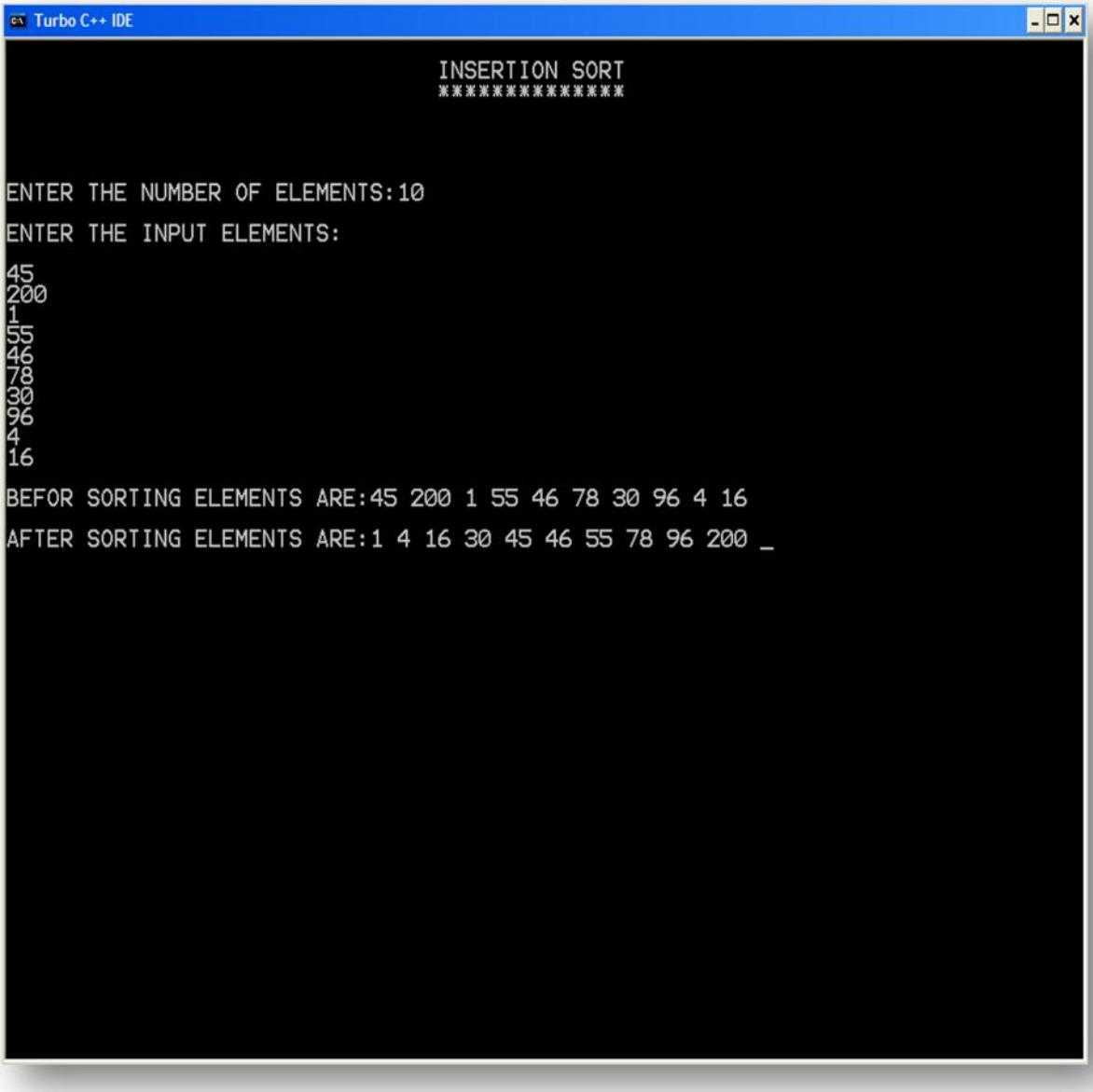
COUNTING SORT



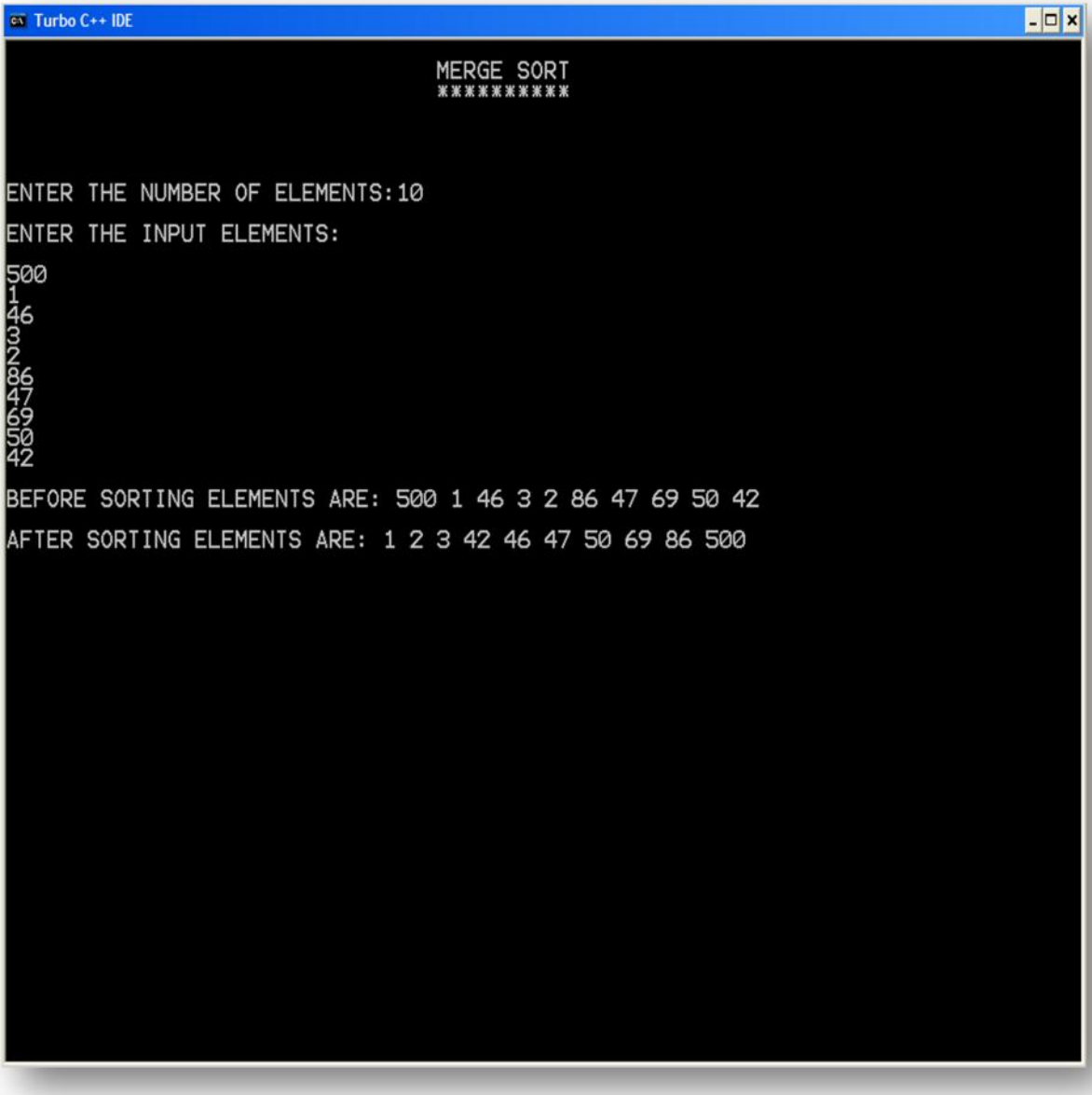
HEAP SORT


Turbo C++ IDE
HEAP SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE INPUT ELEMENTS:
56
20
55
6
34
9
8
1
26
78
BEFORE SORTING ELEMENTS ARE:56 20 55 6 34 9 8 1 26 78
AFTER SORTING ELEMENTS ARE:1 6 8 9 20 26 34 55 56 78 _

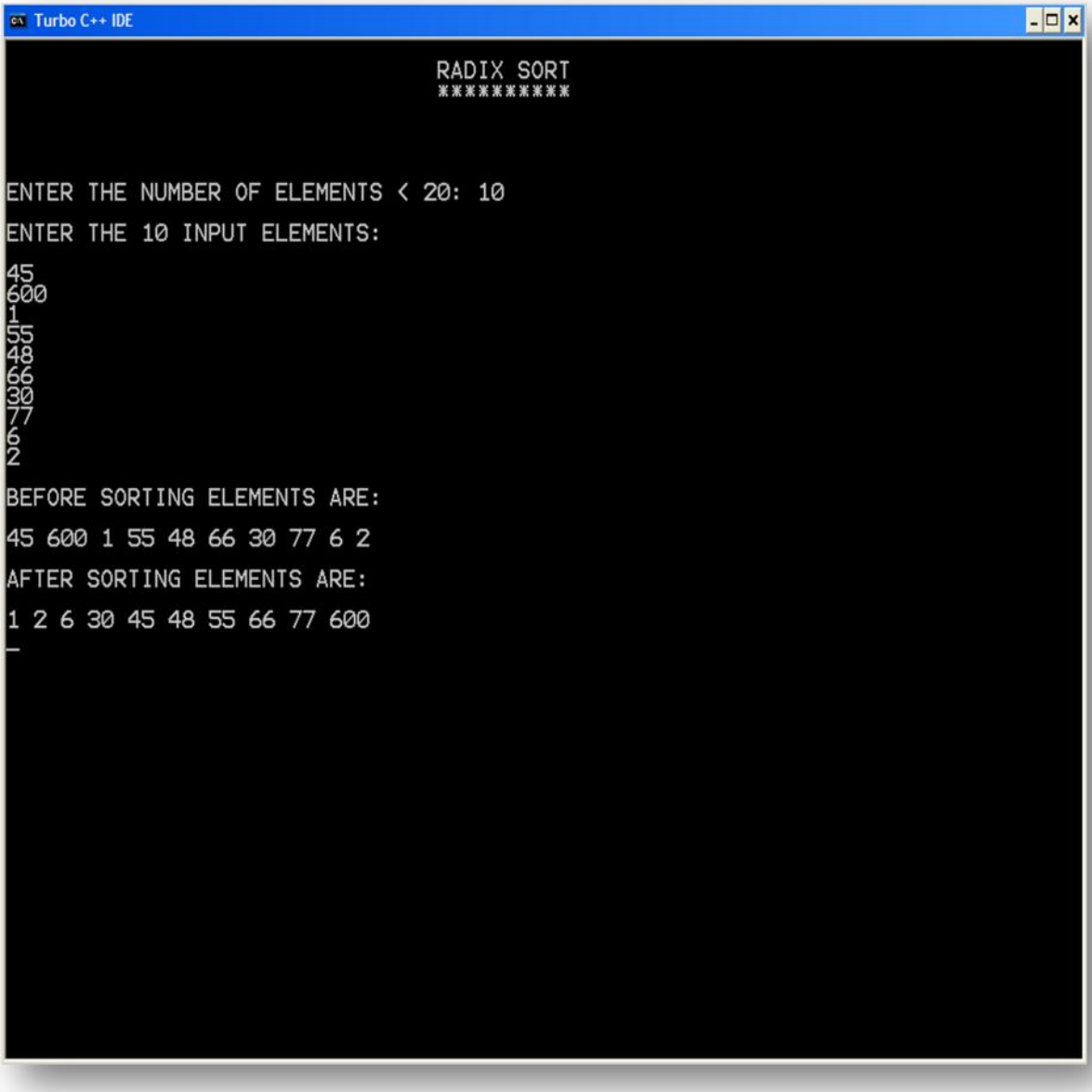
INSERTION SORT



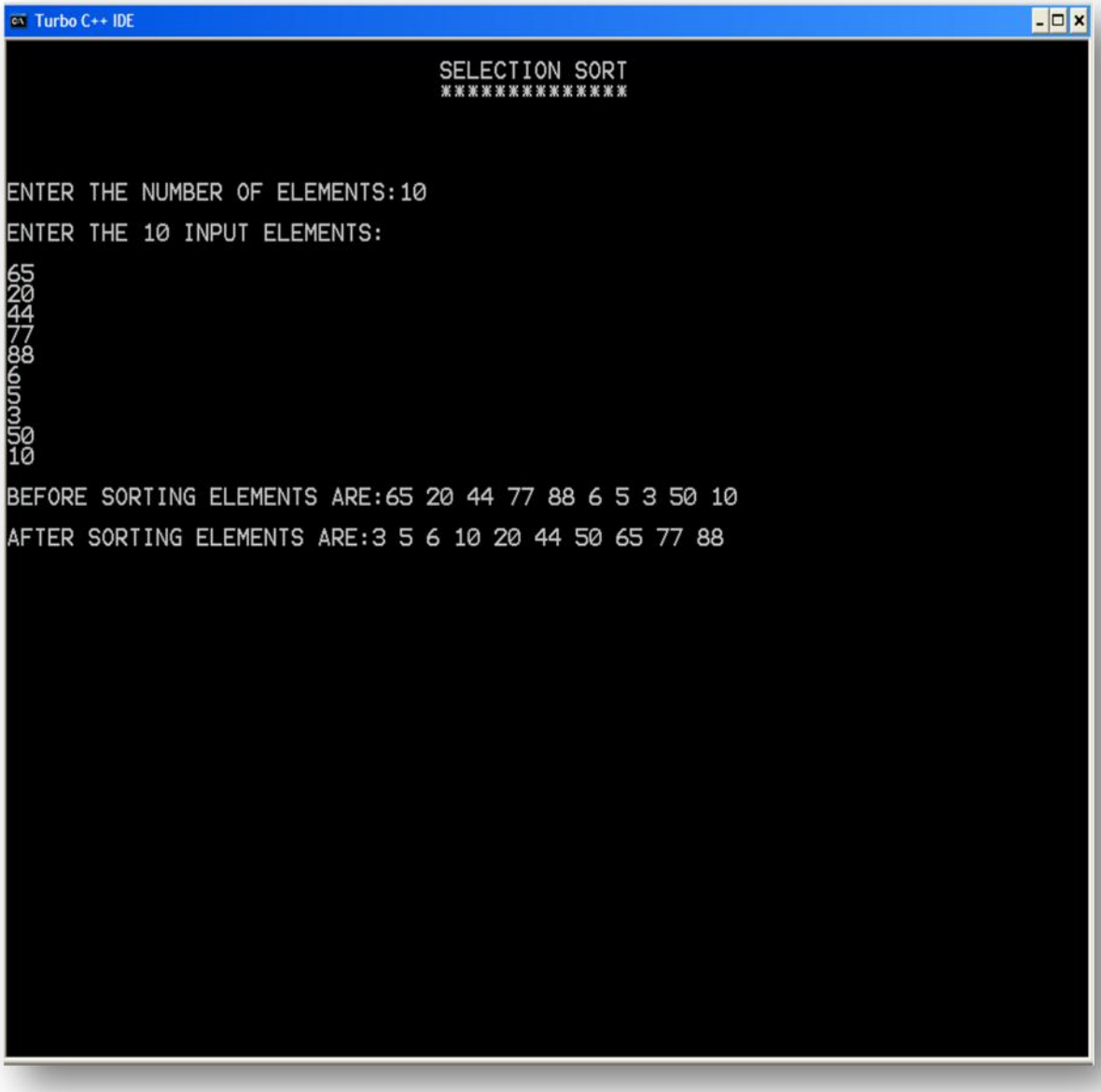
MERGE SORT


Turbo C++ IDE
MERGE SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE INPUT ELEMENTS:
500
1
46
3
2
86
47
69
50
42
BEFORE SORTING ELEMENTS ARE: 500 1 46 3 2 86 47 69 50 42
AFTER SORTING ELEMENTS ARE: 1 2 3 42 46 47 50 69 86 500

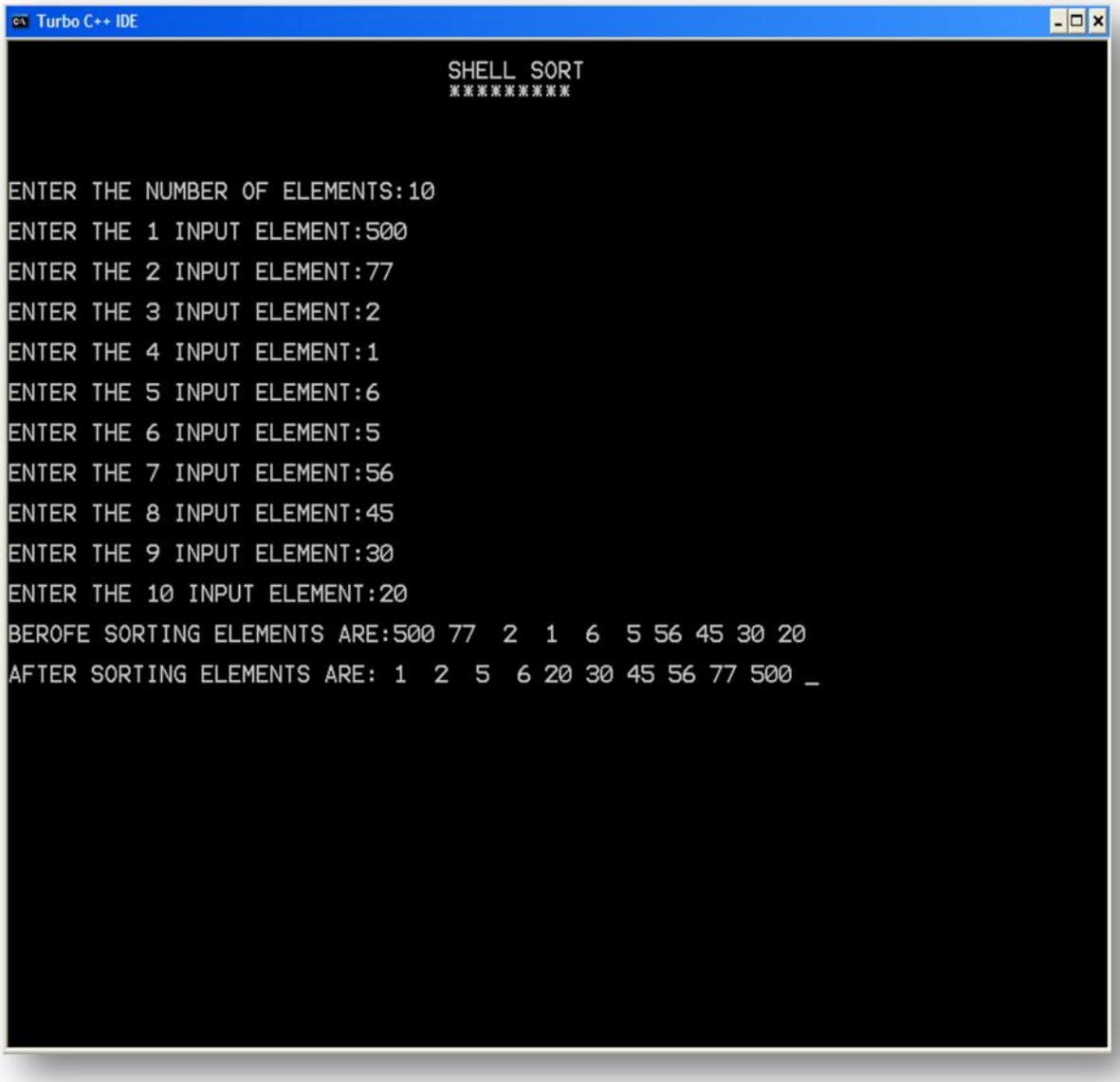
RADIX SORT


Turbo C++ IDE
RADIX SORT
****
ENTER THE NUMBER OF ELEMENTS < 20: 10
ENTER THE 10 INPUT ELEMENTS:
45
600
1
55
48
66
30
77
6
2
BEFORE SORTING ELEMENTS ARE:
45 600 1 55 48 66 30 77 6 2
AFTER SORTING ELEMENTS ARE:
1 2 6 30 45 48 55 66 77 600

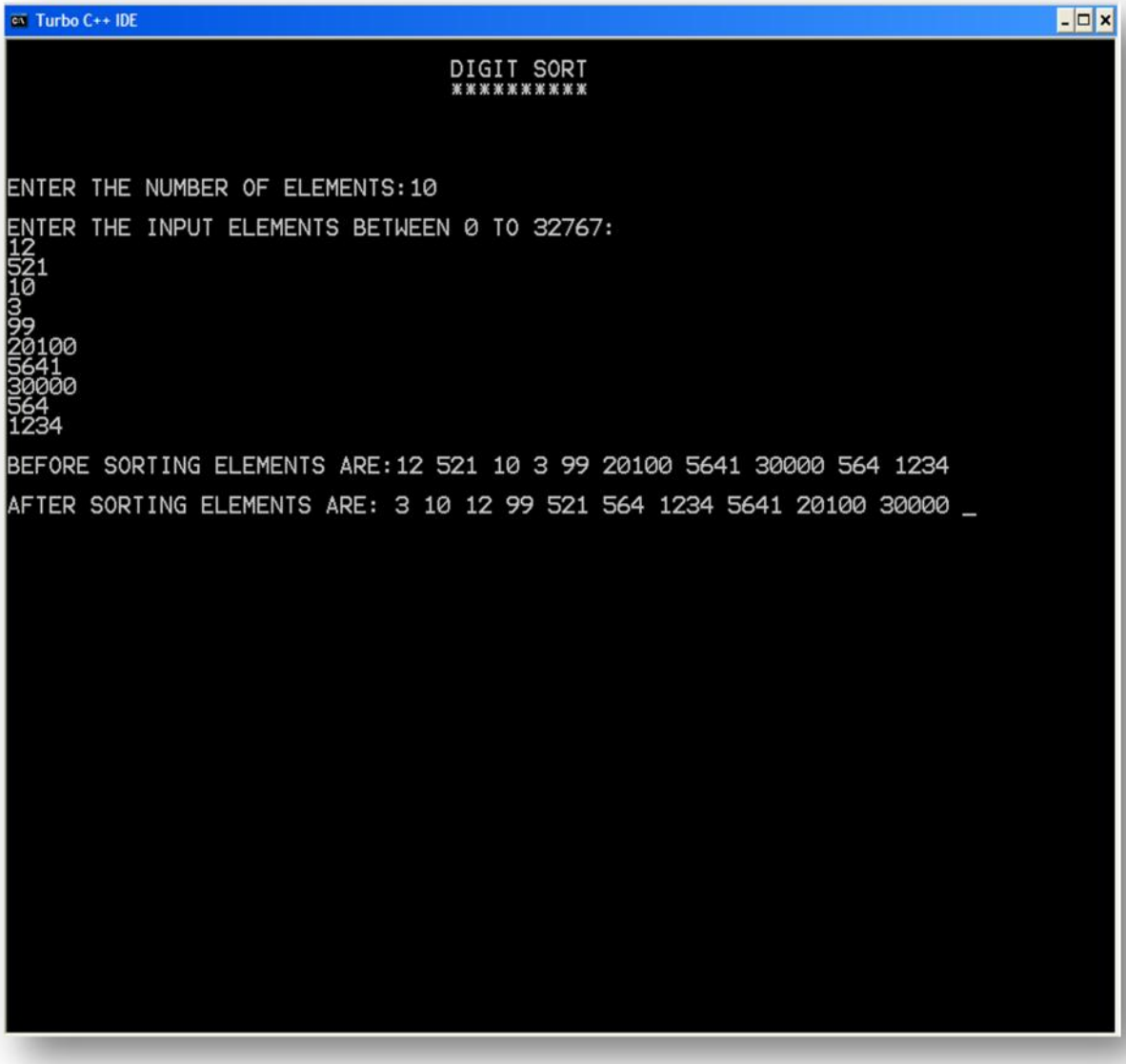
SELECTION SORT



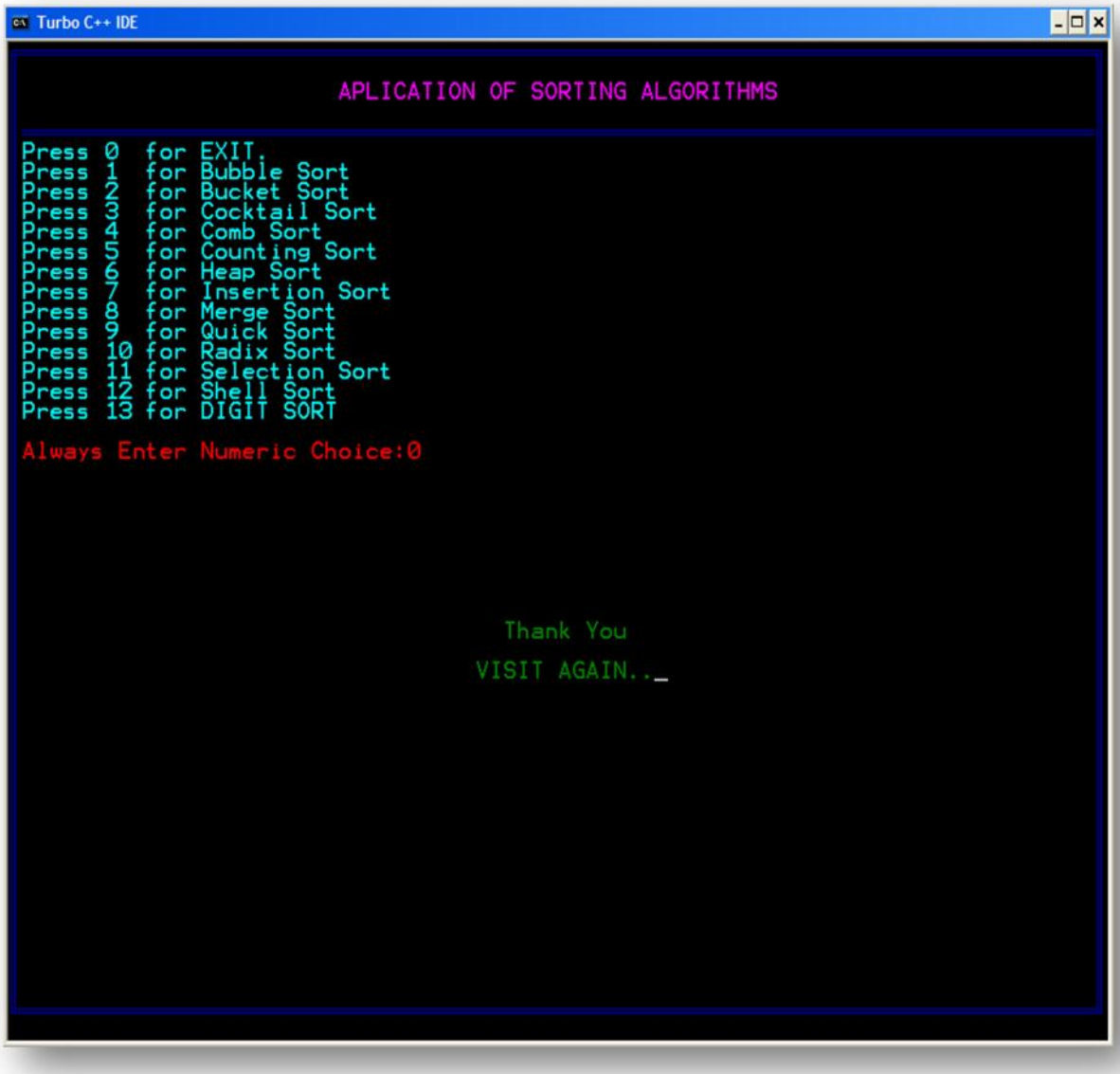
SHELL SORT


Turbo C++ IDE
SHELL SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE 1 INPUT ELEMENT:500
ENTER THE 2 INPUT ELEMENT:77
ENTER THE 3 INPUT ELEMENT:2
ENTER THE 4 INPUT ELEMENT:1
ENTER THE 5 INPUT ELEMENT:6
ENTER THE 6 INPUT ELEMENT:5
ENTER THE 7 INPUT ELEMENT:56
ENTER THE 8 INPUT ELEMENT:45
ENTER THE 9 INPUT ELEMENT:30
ENTER THE 10 INPUT ELEMENT:20
BEROFE SORTING ELEMENTS ARE:500 77 2 1 6 5 56 45 30 20
AFTER SORTING ELEMENTS ARE: 1 2 5 6 20 30 45 56 77 500 _

DIGIT SORT


Turbo C++ IDE
DIGIT SORT
****
ENTER THE NUMBER OF ELEMENTS:10
ENTER THE INPUT ELEMENTS BETWEEN 0 TO 32767:
12
521
10
3
99
20100
5641
30000
564
1234
BEFORE SORTING ELEMENTS ARE:12 521 10 3 99 20100 5641 30000 564 1234
AFTER SORTING ELEMENTS ARE: 3 10 12 99 521 564 1234 5641 20100 30000 _

7. 1.6. EXIT


Turbo C++ IDE
APLICATION OF SORTING ALGORITHMS
Press 0 for EXIT.
Press 1 for Bubble Sort
Press 2 for Bucket Sort
Press 3 for Cocktail Sort
Press 4 for Comb Sort
Press 5 for Counting Sort
Press 6 for Heap Sort
Press 7 for Insertion Sort
Press 8 for Merge Sort
Press 9 for Quick Sort
Press 10 for Radix Sort
Press 11 for Selection Sort
Press 12 for Shell Sort
Press 13 for DIGIT SORT
Always Enter Numeric Choice:0
Thank You
VISIT AGAIN..

7. 1.7. THANK YOU