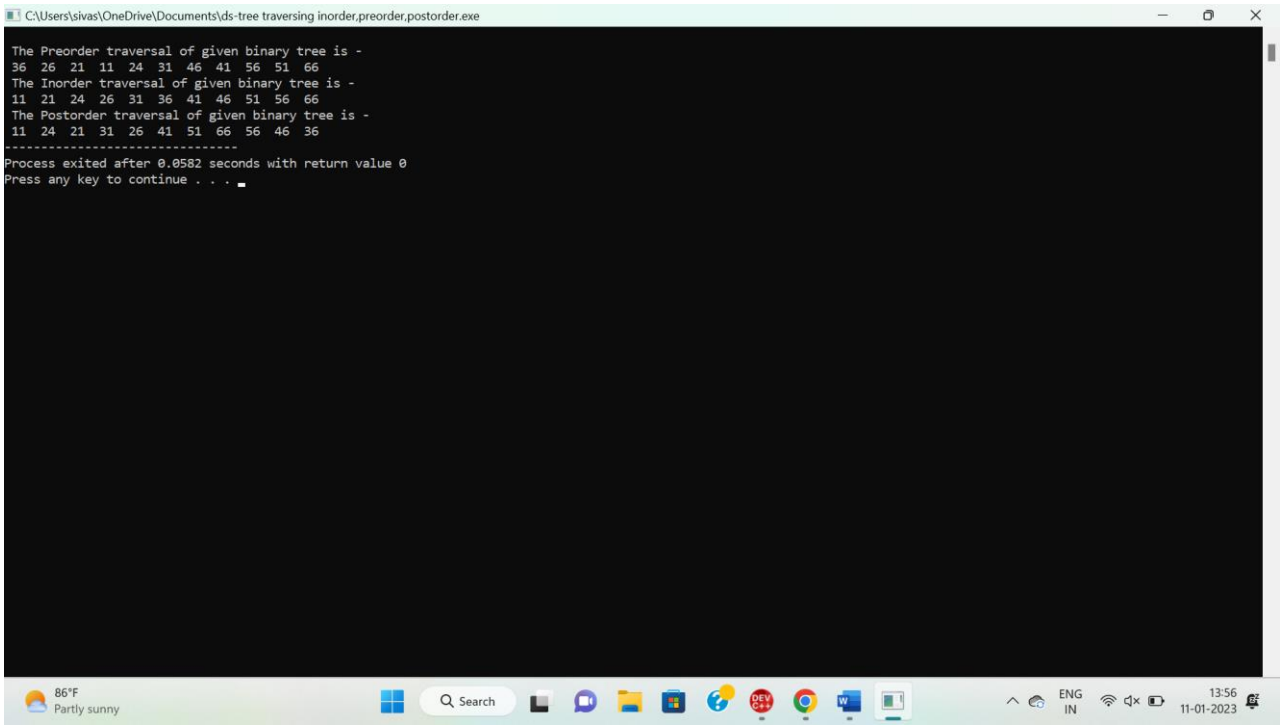
Traversing a tree - inorder,preorder,postorder:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int element;
   struct node* left;
   struct node* right;
};
struct node* createNode(int val)
{
   struct node* Node = (struct node*)malloc(sizeof(struct node));
   Node->element = val;
   Node->left = NULL;
   Node->right = NULL;
   return (Node);
}
void traversePreorder(struct node* root)
{
   if (root == NULL)
      return;
   printf(" %d ", root->element);
   traversePreorder(root->left);
   traversePreorder(root->right);
}
void traverseInorder(struct node* root)
{
   if (root == NULL)
      return;
   traverseInorder(root->left);
   printf(" %d ", root->element);
```

```
    traverseInorder(root->right);
}
void traversePostorder(struct node* root)
{
    if (root == NULL)
        return;
    traversePostorder(root->left);
    traversePostorder(root->right);
    printf(" %d ", root->element);
}
int main()
{
    struct node* root = createNode(36);
    root->left = createNode(26);
    root->right = createNode(46);
    root->left->left = createNode(21);
    root->left->right = createNode(31);
    root->left->left->left = createNode(11);
    root->left->left->right = createNode(24);
    root->right->left = createNode(41);
    root->right->right = createNode(56);
    root->right->right->left = createNode(51);
    root->right->right->right = createNode(66);
    printf("\n The Preorder traversal of given binary tree is -\n");
    traversePreorder(root);
    printf("\n The Inorder traversal of given binary tree is -\n");
    traverseInorder(root);
    printf("\n The Postorder traversal of given binary tree is -\n");
    traversePostorder(root);
    return 0;
}
```



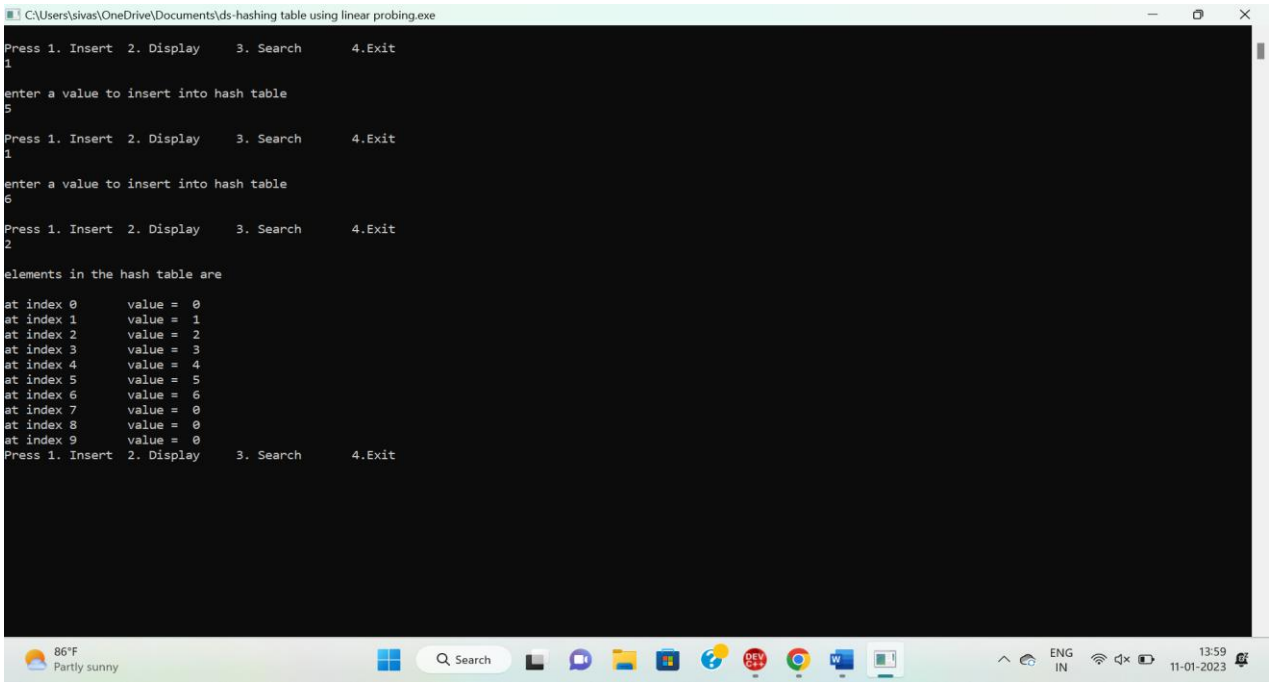
Implementing hashing table using linear probing:

```
 #include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};


void insert()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
   {

    index=(hkey+i)%TABLE_SIZE;

    if(h[index] == NULL)
```

```
    {
       h[index]=key;
        break;
    }

   }

   if(i == TABLE_SIZE)

    printf("\nelement cannot be inserted\n");
}
void search()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
   index=(hkey+i)%TABLE_SIZE;
   if(h[index]==key)
   {
     printf("value is found at index %d",index);
     break;
   }
  }
  if(i == TABLE_SIZE)
   printf("\n value is not found\n");
}
void display()
```

```
{

  int i;

  printf("\nelements in the hash table are \n");

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

  printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
    int opt,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
```

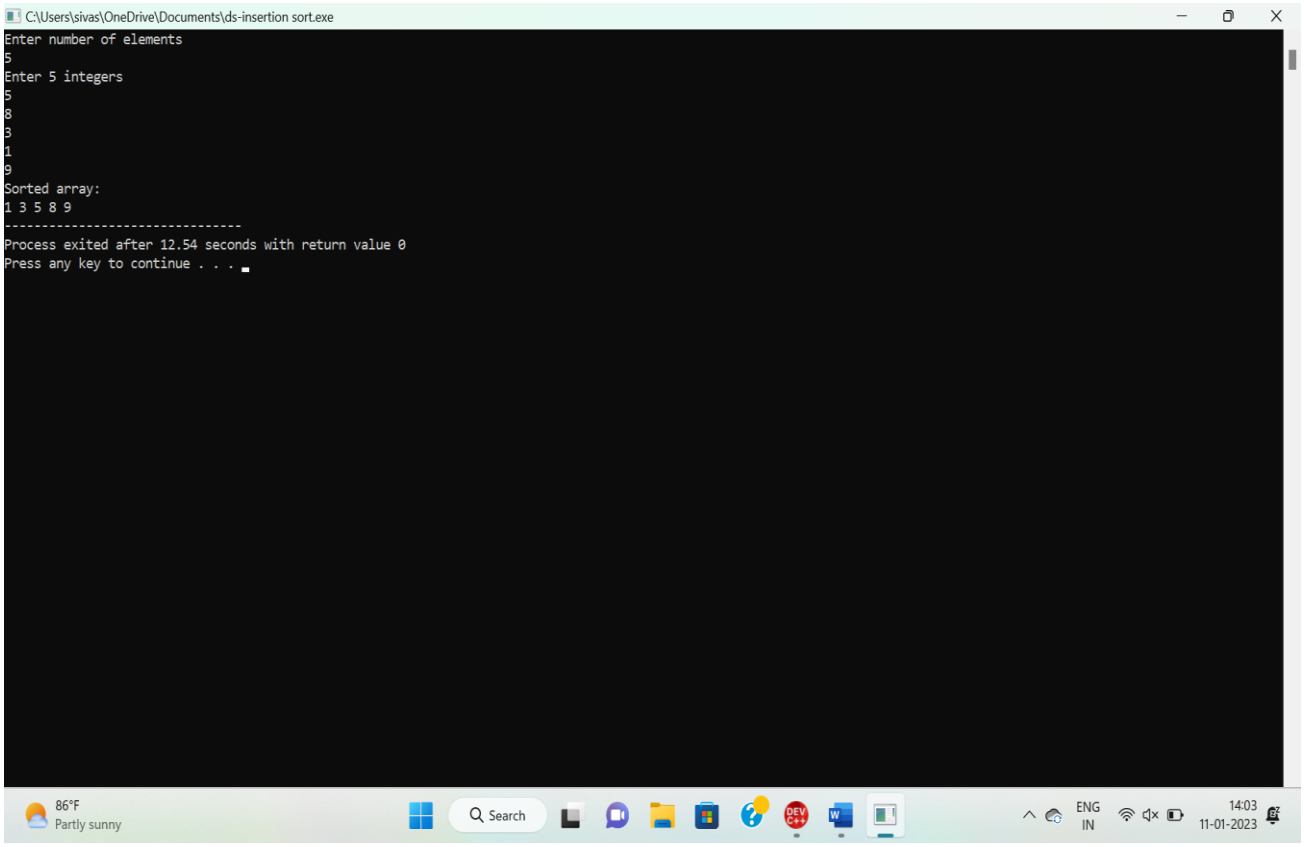
```
    }
}
```



Insertion sort:

```
#include <stdio.h>
int main(void)
{
   int n,i,j,temp;
   int arr[60];
   printf("Enter number of elements\n");
   scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for (i=0;i<n;i++)
   {
      scanf("%d",&arr[i]);
   }
   for (i=1;i<n;i++)
   {
      j = i;
```

```
    while (j>0&&arr[j-1]>arr[j])
    {
        temp = arr[j];
        arr[j] = arr[j-1];
        arr[j-1] = temp;
        j--;
    }
  }
  printf("Sorted array:\n");
  for (i=0;i<n;i++)
  {
    printf("%d ",arr[i]);
  }
  return 0;
}
```

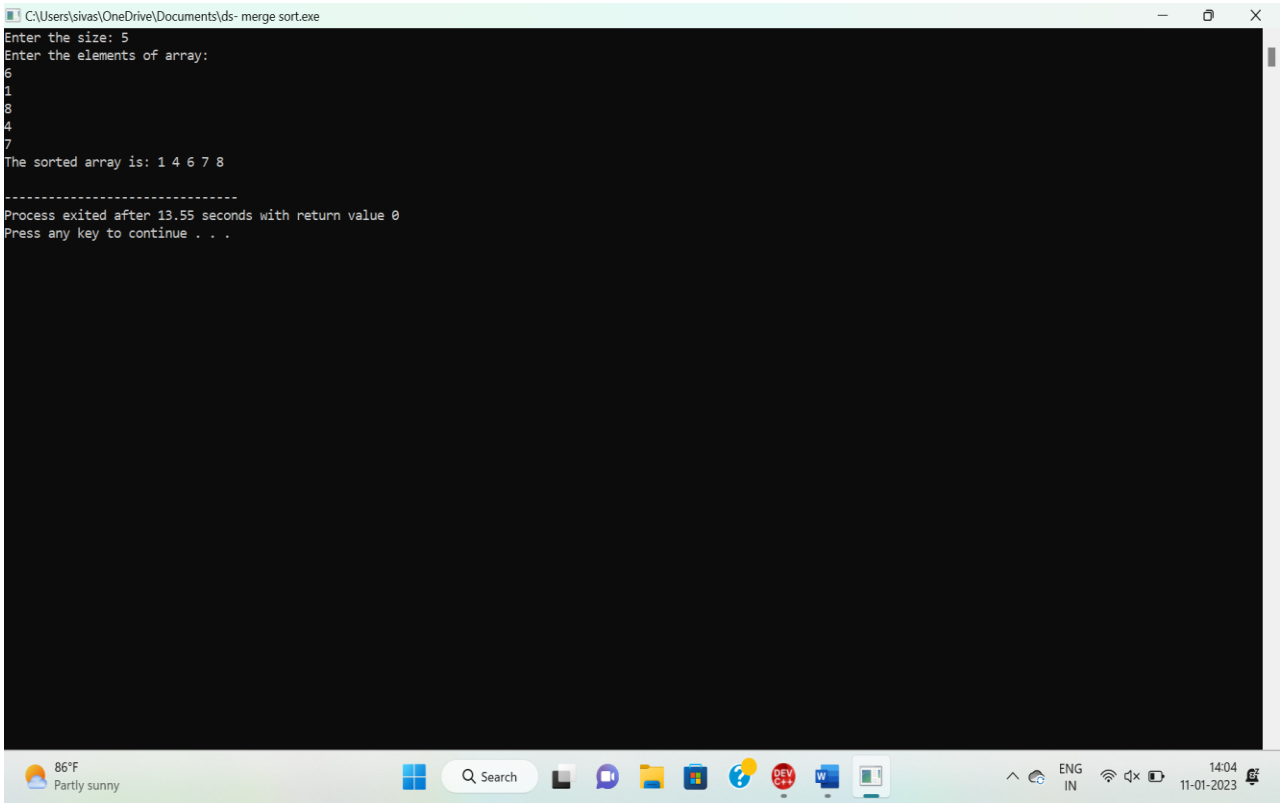


Merge sort:

```
#include <stdio.h>
#include <stdlib.h>
void Merge(int arr[], int left, int mid, int right)
{
   int i, j, k;
   int size1 = mid - left + 1;
   int size2 = right - mid;
   int Left[size1], Right[size2];
   for (i = 0; i < size1; i++)
      Left[i] = arr[left + i];
   for (j = 0; j < size2; j++)
      Right[j] = arr[mid + 1 + j];
   i = 0;
   j = 0;
   k = left;
   while (i < size1 && j < size2)
   {
      if (Left[i] <= Right[j])
      {
         arr[k] = Left[i];
         i++;
      }
      else
      {
         arr[k] = Right[j];
         j++;
      }
      k++;
```

```
    }
    while (i < size1)
    {
        arr[k] = Left[i];
        i++;
        k++;
    }
    while (j < size2)
    {
        arr[k] = Right[j];
        j++;
        k++;
    }
}
void Merge_Sort(int arr[], int left, int right)
{
    if (left < right)
    {

        int mid = left + (right - left) / 2;
        Merge_Sort(arr, left, mid);
        Merge_Sort(arr, mid + 1, right);

        Merge(arr, left, mid, right);
    }
}
int main()
{
    int size;
    printf("Enter the size: ");
    scanf("%d", &size);
```

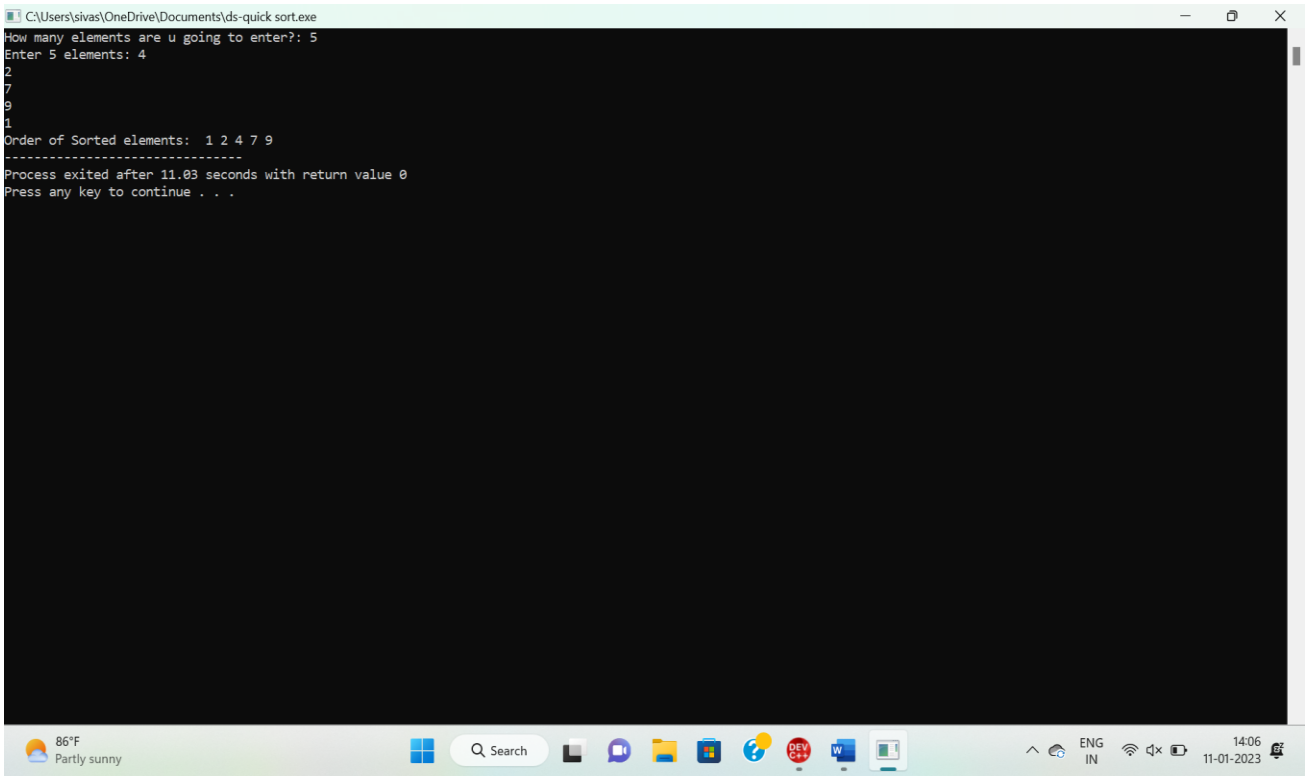
```
    int arr[size];
    printf("Enter the elements of array:\n");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &arr[i]);
    }
    Merge_Sort(arr, 0, size - 1);
    printf("The sorted array is: ");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```



Quick sort:

```
#include<stdio.h>
void quicksort(int number[25],int first,int last)
{
  int i,j, pivot,temp;
  if(first<last)
  {
    pivot=first;
    i=first;
    j=last;
    while(i<j)
          {
      while(number[i]<=number[pivot]&&i<last)
      i++;
      while(number[j]>number[pivot])
      j--;
      if(i<j)
                  {
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
    }
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);
  }
}
```

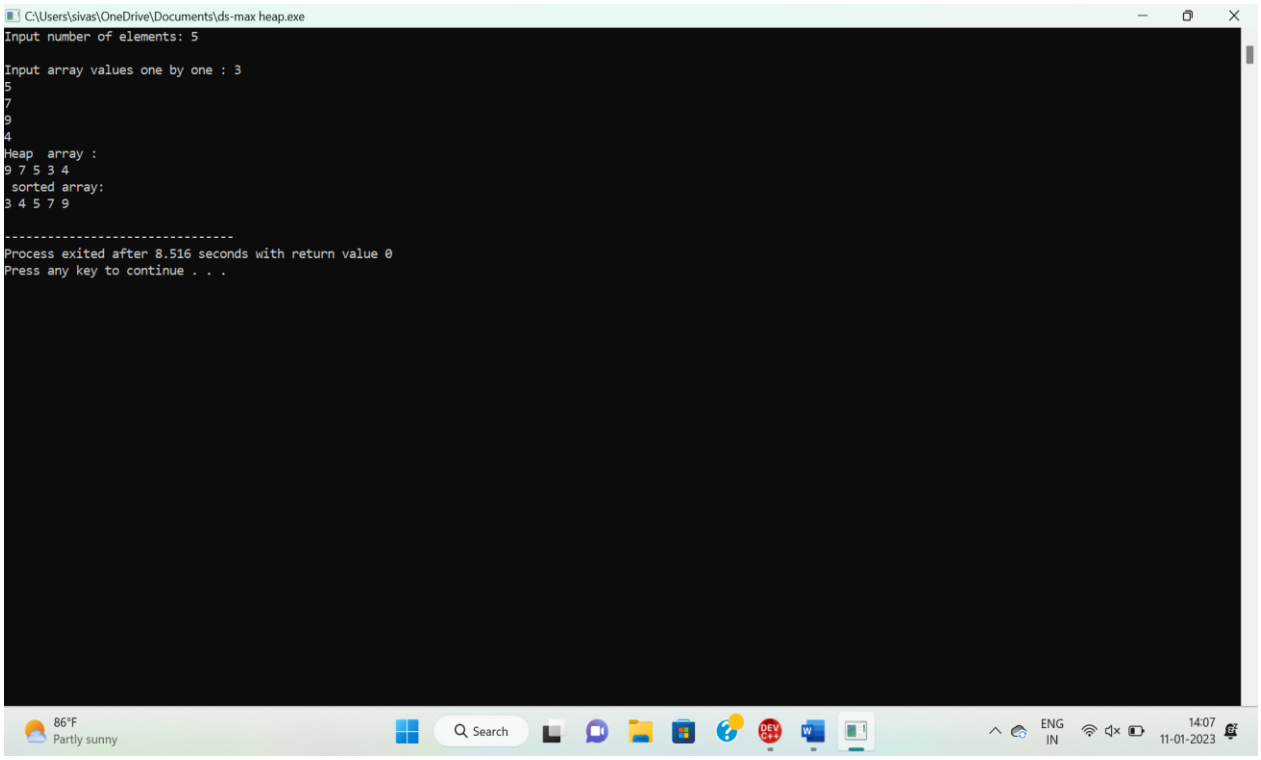
```
int main()
{
  int i, count, number[25];
  printf("How many elements are u going to enter?: ");
  scanf("%d",&count);
  printf("Enter %d elements: ", count);
  for(i=0;i<count;i++)
  scanf("%d",&number[i]);
  quicksort(number,0,count-1);
  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
  printf(" %d",number[i]);
  return 0;
}
```



Heap sort:

```
#include <stdio.h>
 int main()
 {
 int arr[10],no,i,j,c,heap_root,temp;
 printf("Input number of elements: ");
 scanf("%d", &no);
 printf("\nInput array values one by one : ");
 for (i = 0; i <  no; i++)
 scanf("%d", &arr[i]);
 for (i=1;i<no;i++)
 {
 c = i;
 do
 {
 heap_root  = (c - 1) / 2;
 if  (arr[heap_root] < arr[c])
 {
 temp =  arr[heap_root];
 arr[heap_root] = arr[c];
 arr[c]  = temp;
 }
 c =  heap_root;
 } while (c !=  0);
 }
 printf("Heap  array :\n");
 for (i = 0; i <  no; i++)
 printf("%d ", arr[i]);
 for (j = no - 1; j  >= 0; j--)
 {
```

```
temp = arr[0];
arr[0] = arr[j];
arr[j] = temp;
heap_root = 0;
do
{
c = 2 *  heap_root + 1;
if  ((arr[c] < arr[c + 1]) && c < j-1)
c++;
if  (arr[heap_root]<arr[c] && c<j)
{
temp =  arr[heap_root];
arr[heap_root] = arr[c];
arr[c]  = temp;
}
heap_root  = c;
} while (c  < j);
}
printf("\n sorted array:\n");
for (i = 0; i <  no; i++)
printf("%d ", arr[i]);
printf("\n");
}
```


C:\Users\sivas\OneDrive\Documents\ds-max heap.exe
Input number of elements: 5
Input array values one by one : 3
5
7
9
4
Heap array :
9 7 5 3 4
sorted array:
3 4 5 7 9
Process exited after 8.516 seconds with return value 0
Press any key to continue . . .
86°F
Partly sunny
Search
ENG
IN
14:07
11-01-2023