**Experiment no 1**

1. **Program to implement Insertion and Selection Sort**
2. Implementation of Insertion Sort

#include<stdio.h>

void inssort(int ar[],int n)

{

int i,j;

for(i=1;i<=n;i++)

{

int temp=ar[i];

j=i-1;

while(temp<ar[j])

{

ar[j+1]=ar[j];

j--;

}

ar[j+1]=temp;

}

printf("The elements after sorting is\n\n");

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

printf("%d ",ar[i]);

}

main()

{

int n;

printf("\nEnter the total number of elements to be sorted ");

scanf("%d",&n);

int ar[n];

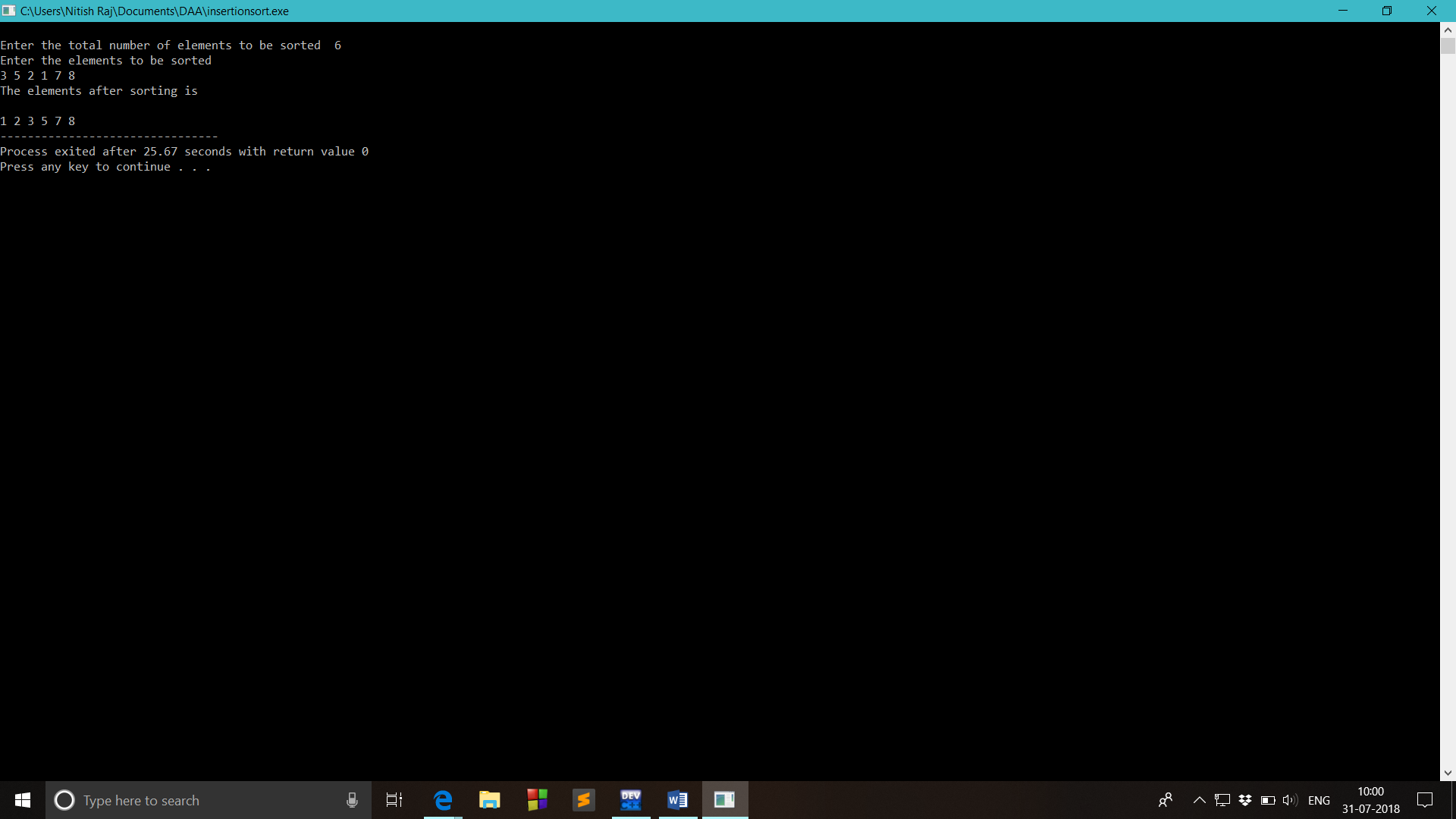
printf("Enter the elements to be sorted\n");

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

scanf("%d",&ar[i]);

inssort(ar,n);

}



1. **Implementation of Selection Sort**

#include<stdio.h>

void selection(int ar[],int n)

{

for(int i=0;i<n-1;i++)

{

for(int j=i+1;j<n;j++)

{

if(ar[j]<ar[i])

{

int temp=ar[i];

ar[i]=ar[j];

ar[j]=temp;

}

}

}

printf("\nElements after sorting\n");

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

printf("%d ",ar[i]);

}

main()

{

int n;

printf("\nEnter the total number of elements to be sorted ");

scanf("%d",&n);

printf("Enter the elements to be sorted\n");

int ar[n];

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

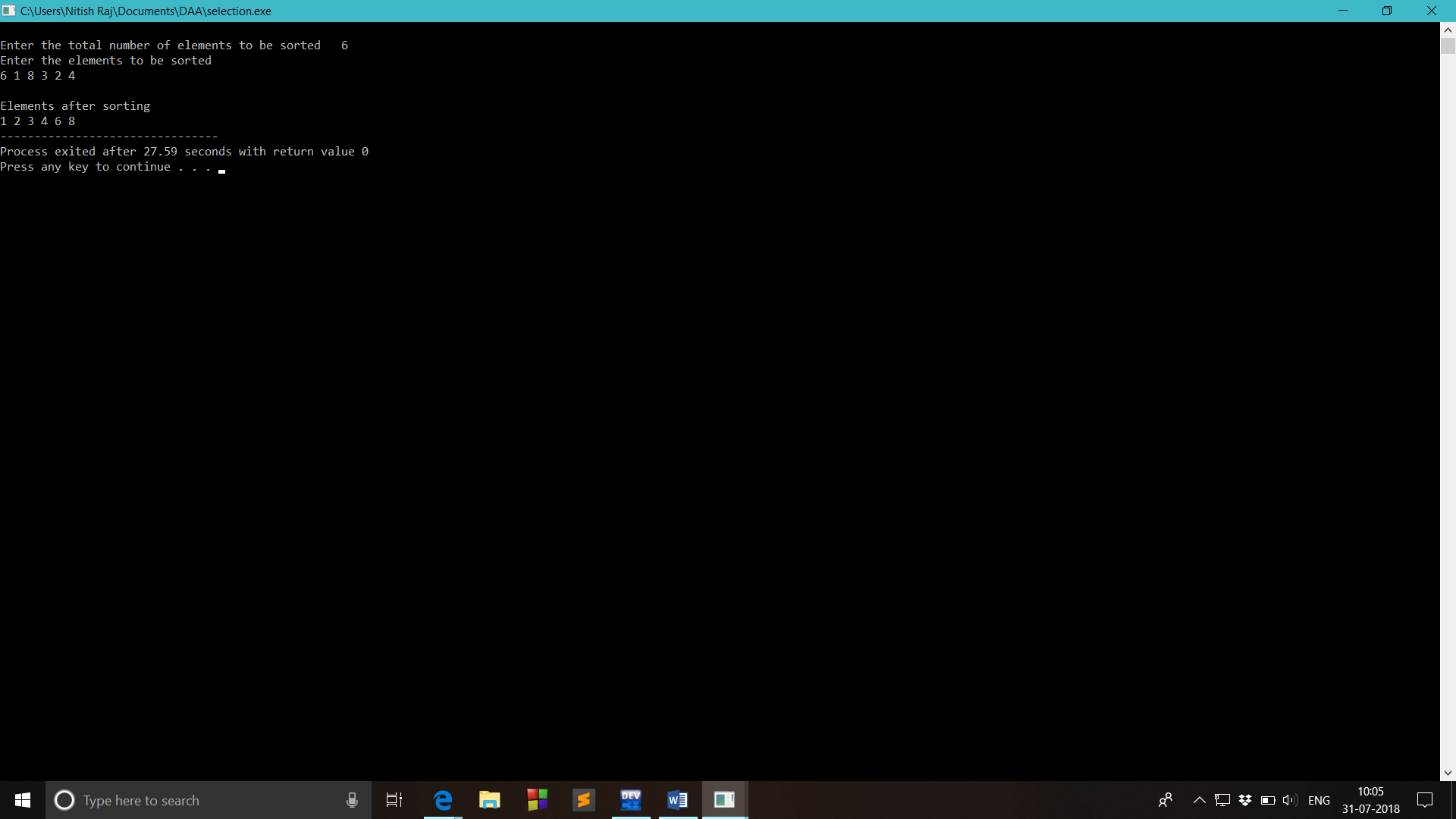
{

scanf("%d",&ar[i]);

}

selection(ar,n);

}



1. **Program to implement Selection and Insertion Sort recursively.**
2. **Implementation of Selection Sort Recursively**

#include<stdio.h>

void selectionsort(int ar[], int i,int size)

{

if (i>=size-1)

return;

int min=i;

for (int j=i+1;j<size;j++ )

{

if (ar[j]<ar[min])

min=j;

}

int temp = ar[i];

ar[i] = ar[min];

ar[min]=temp;

selectionsort(ar,i+1,size);

}

main()

{

int ar[20],n;

printf("Enter the total number of elements to be sorted ");

scanf("%d",&n);

printf("\nEnter the elements for sorting\n");

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

scanf("%d",&ar[i]);

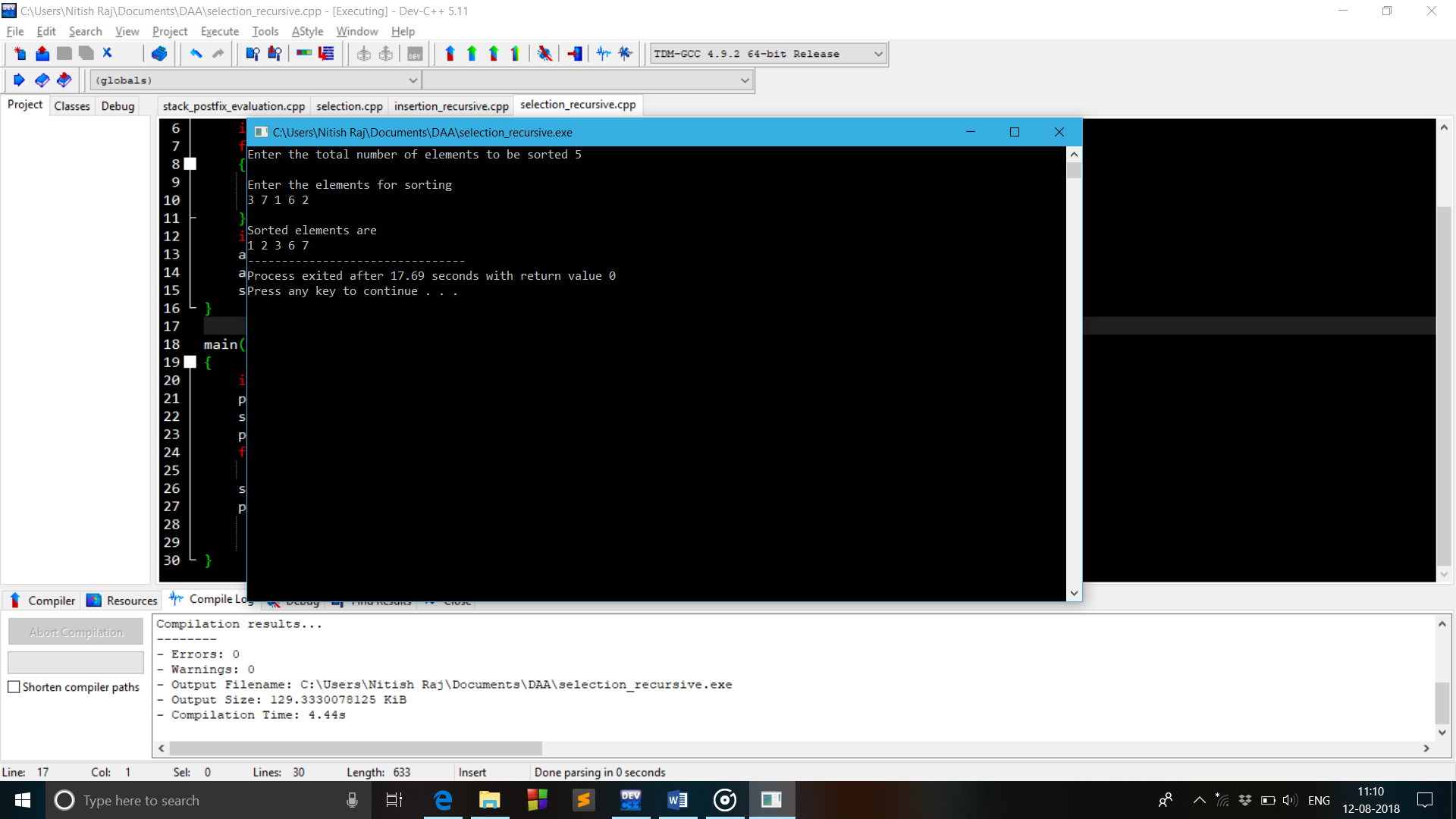
selectionsort(ar,0,n);

printf("\nSorted elements are\n");

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

printf("%d ",ar[i]);

}



1. **Implementation of Insertion Sort Recursively**

#include<stdio.h>

void insertionsort(int ar[], int i,int size)

{

if(i>size)

return;

int beg=ar[i];

int j=i-1;

while((j>=0)&&(ar[j]>beg))

{

ar[j+1]=ar[j];

j--;

}

ar[j+1]=beg;

insertionsort(ar,i+1,size);

}

main()

{

int n;

printf("Enter the total number of elements to be sorted ");

scanf("%d",&n);

int ar[n];

printf("\nEnter the elements for sorting\n");

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

scanf("%d",&ar[i]);

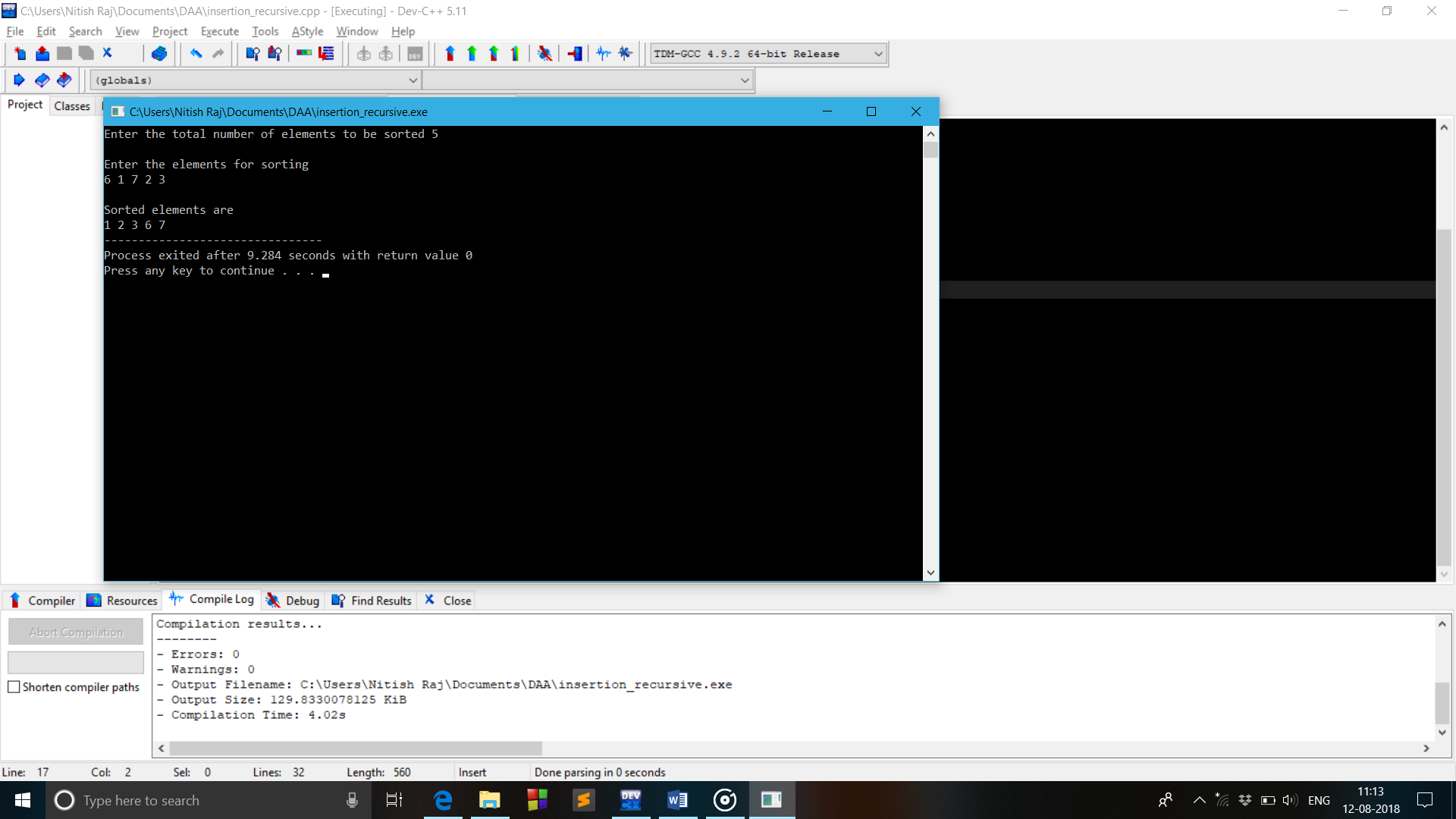
insertionsort(ar,1,n);

printf("\nSorted elements are\n");

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

printf("%d ",ar[i]);

}



**Experiment no 2**

1. **Program to implement Quick sort.**

**Method 1:**

#include<stdio.h>

void quicksort(int ar[],int start, int end)

{

if(start>end)

return;

int i=start-1;

int pindex=end;

for(int j=start;j<end;j++)

{

if(ar[j]<ar[pindex])

{

i=i+1;

int temp=ar[j];

ar[j]=ar[i];

ar[i]=temp;

}

}

i=i+1;

int flag=ar[i];

ar[i]=ar[end];

ar[end]=flag;

quicksort(ar,start,i-1);

quicksort(ar,i+1,end);

}

void print(int ar[],int n)

{

printf("\nSorted elements are\n");

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

printf("%d ",ar[i]);

}

main()

{

int n;

printf("\nEnter the no of elements for sorting ");

scanf("%d",&n);

int ar[n];

printf("\nEnter the elements for sorting\n");

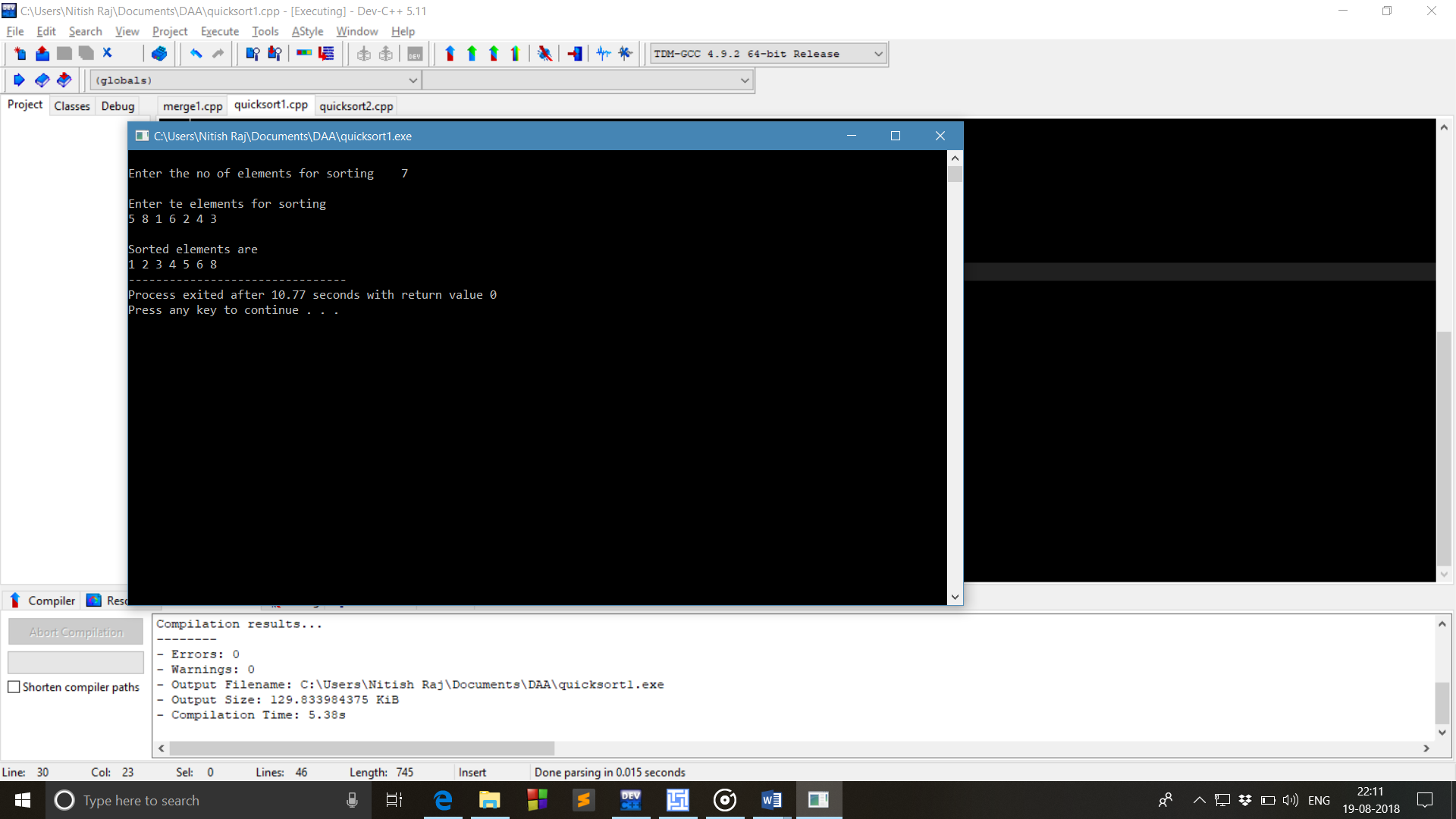
for(int i=0;i<n;i++)

scanf("%d",&ar[i]);

quicksort(ar,0,n-1);

print(ar,n);

}



**Method 2:**

#include<stdio.h>

int partition(int ar[],int low,int high)

{

int i,j,pivot,temp;

if(low<high)

{

pivot=low;

i=low;

j=high;

while(i<j)

{

while(ar[i]<ar[pivot]&&i<high)

i++;

while(ar[j]>ar[pivot])

j--;

if(i<j)

{

temp=ar[i];

ar[i]=ar[j];

ar[j]=temp;

}

}

temp=ar[pivot];

ar[pivot]=ar[j];

ar[j]=temp;

return j;

}

}

quicksort(int ar[],int low,int high)

{

int pivotloc;

if(low>high)

return 0;

pivotloc=partition(ar,low,high);

quicksort(ar,low,pivotloc-1);

quicksort(ar,pivotloc+1,high);

}

void print(int ar[],int n)

{

printf("\nSorted elements are\n");

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

printf("%d ",ar[i]);

}

int main()

{

int i,n;

printf("How much numbers you want to sort ");

scanf("%d",&n);

int ar[n];

printf("\nEnter the numbers for sorting\n");

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

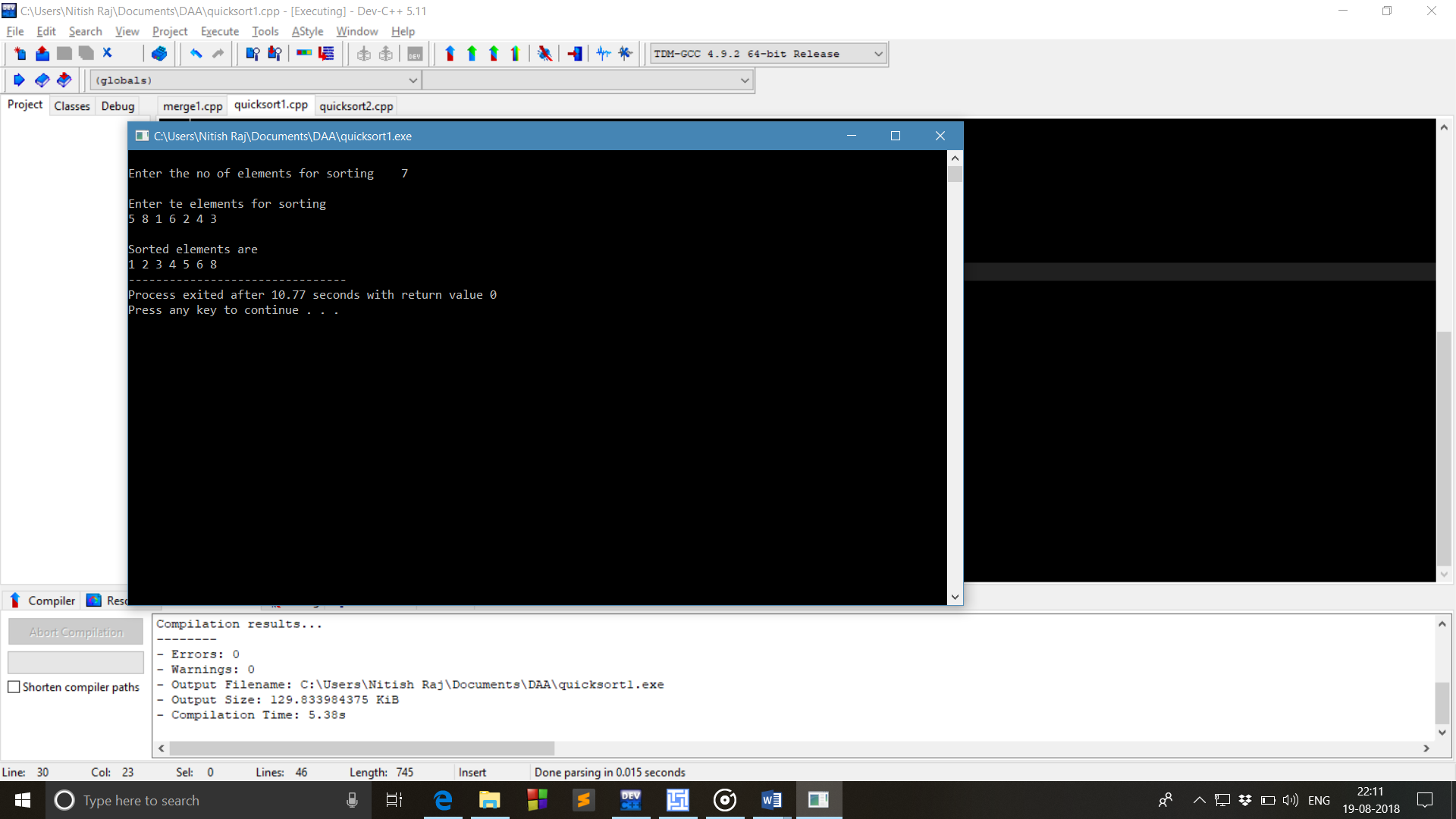
scanf("%d",&ar[i]);

quicksort(ar,0,n-1);

print(ar,n);

return 0;

}



**Experiment no 3**

**Program for the implementation of Merge sort**

**Method 1:**

#include<stdio.h>

void merge(int a[],int l,int m,int h)

{

int i,j,k;

int n1=m-l+1;

int n2=h-m;

int ar1[n1],ar2[n2];

for(i=0;i<n1;i++)

{

ar1[i]=a[l+i];

}

for(j=0;j<n2;j++)

ar2[j]=a[m+1+j];

ar1[i]=9999;

ar2[j]=9999;

i=0;j=0;

for(k=l;k<=h;k++)

{

if(ar1[i]<=ar2[j])

a[k]=ar1[i++];

else

a[k]=ar2[j++];

}

}

void mergesort(int a[],int start,int end)

{

int mid;

if(start<end)

{

mid=(start+end)/2;

mergesort(a,start,mid);

mergesort(a,mid+1,end);

merge(a,start,mid,end);

}

}

int main()

{

int n,i;

printf("Enter no of elements for sorting: ");

scanf("%d",&n);

int ar[n];

printf("Enter the elements for sorting :\n");

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

scanf("%d",&a[i]);

mergesort(a,0,n-1);

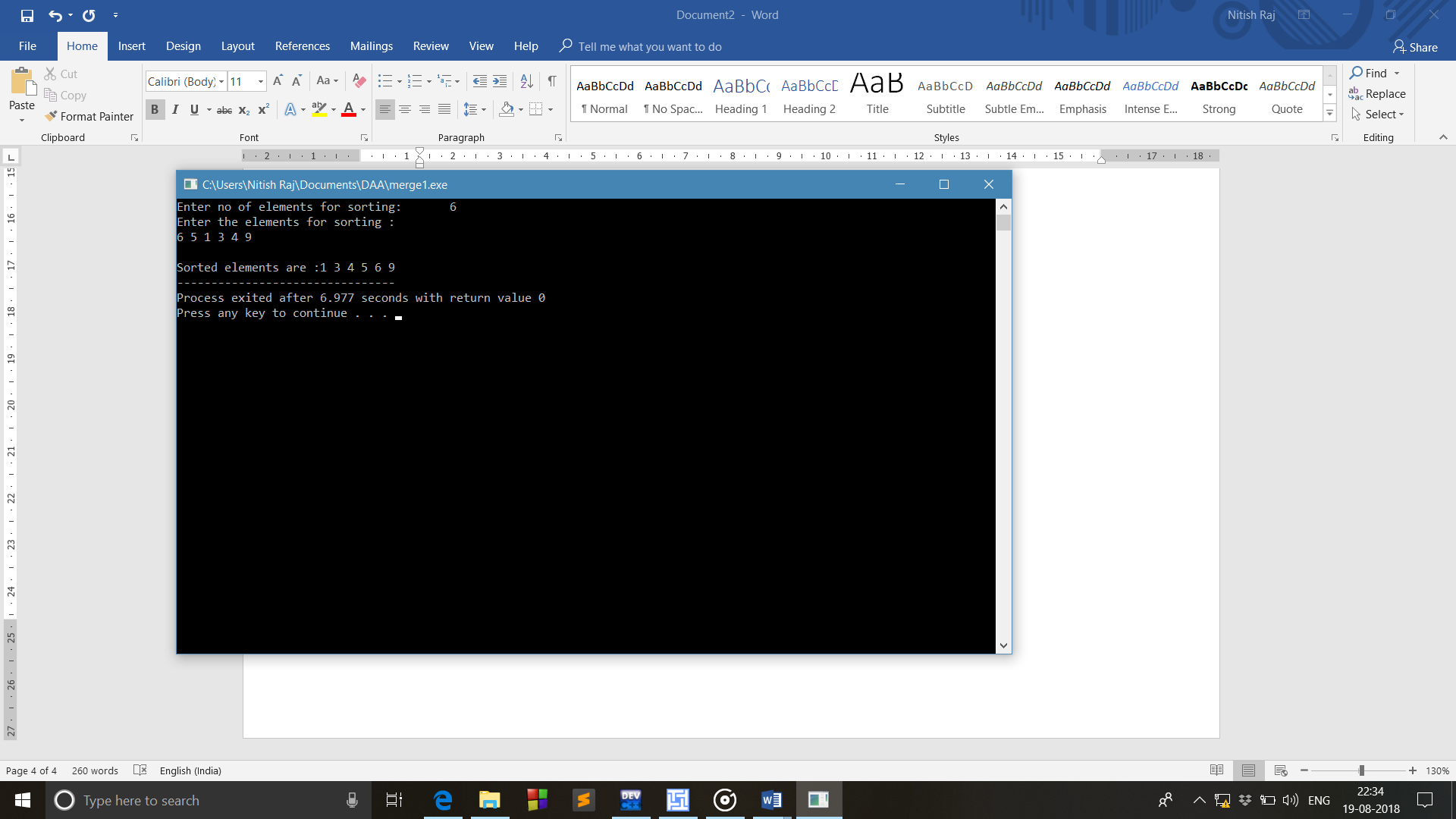
printf("\nSorted elements are :");

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

printf("%d ",a[i]);

return 0;

}



**Method 2:**

#include<stdio.h>

void merge(int arr[],int start,int end,int mid)

{

int i =start;

int j=mid+1;

int k=start;

int temp[end-start];

while(i<=mid && j<=end)

{

if(arr[i]>=arr[j])

{

temp[k]=arr[j];

j=j+1;

}

else{

temp[k]=arr[i];

i=i+1;

}

k=k+1;

}

while(i<=mid)

{

temp[k]=arr[i];

i++;

k++;

}

while(j<=end)

{

temp[k]=arr[j];

j++;

k++;

}

for(int i=start;i<=end;i++)

{

arr[i]=temp[i];

}

}

void mergesort2(int arr[],int start,int end)

{

if(start<end)

{

int mid =(start+end)/2;

mergesort2(arr,start,mid);

mergesort2(arr,mid+1,end);

merge(arr,start,end,mid);

}

}

void print(int arr[],int n)

{

printf("Sorted elements are\n");

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

{

printf("%d ",arr[i]);

}

}

int main()

{

int n;

printf("Enter no of elements for sorting: ");

scanf("%d",&n);

int arr[n];

printf("Enter the elements for Sorting\n”);

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

{

scanf("%d",&arr[i]);

}

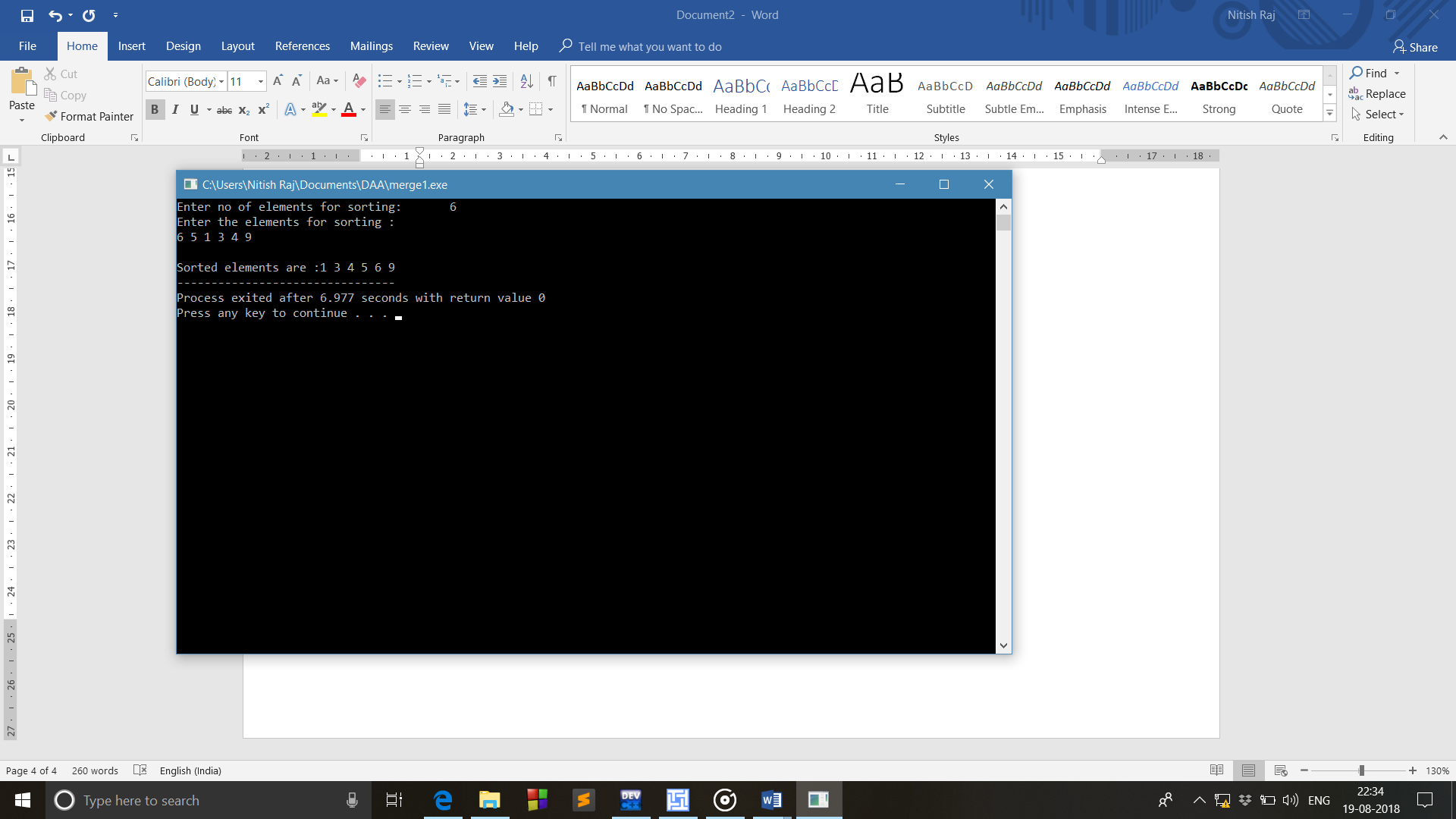
int start =0;

int end =n-1;

mergesort2(arr,start,end);

print(arr,n);

}



**Experiment no 4**

1. **Implementation of Heap Sort without Recursion**

#include<stdio.h>

void swap(int ar[],int value1,int value2)

{

int temp=ar[value1];

ar[value1]=ar[value2];

ar[value2]=temp;

}

void maxheapify(int ar[],int index,int n)

{

int highindex=index;

if((2\*index<=n)&&ar[2\*index]>ar[highindex])

highindex=2\*index;

if((2\*index+1<=n)&&ar[2\*index+1]>ar[highindex])

highindex=2\*index+1;

if(highindex!=index)

{

swap(ar,index,highindex);

maxheapify(ar,highindex,n);

}

}

void buildmaxheap(int ar[],int n)

{

if(n<1)

return ;

for(int i=n/2;i>=1;i--)

maxheapify(ar,i,n);

swap(ar,1,n);

buildmaxheap(ar,n-1);

}

void print(int ar[],int n)

{

for(int i=1;i<=n;i++)

{

printf("%d ",ar[i]);

}

}

main()

{

int n;

printf("\nHow many numbers you want to sort ");

scanf("%d",&n);

int ar[n+1];

printf("\nEnter the numbers in heap\n");

for(int i=1;i<=n;i++)

{

scanf("%d",&ar[i]);

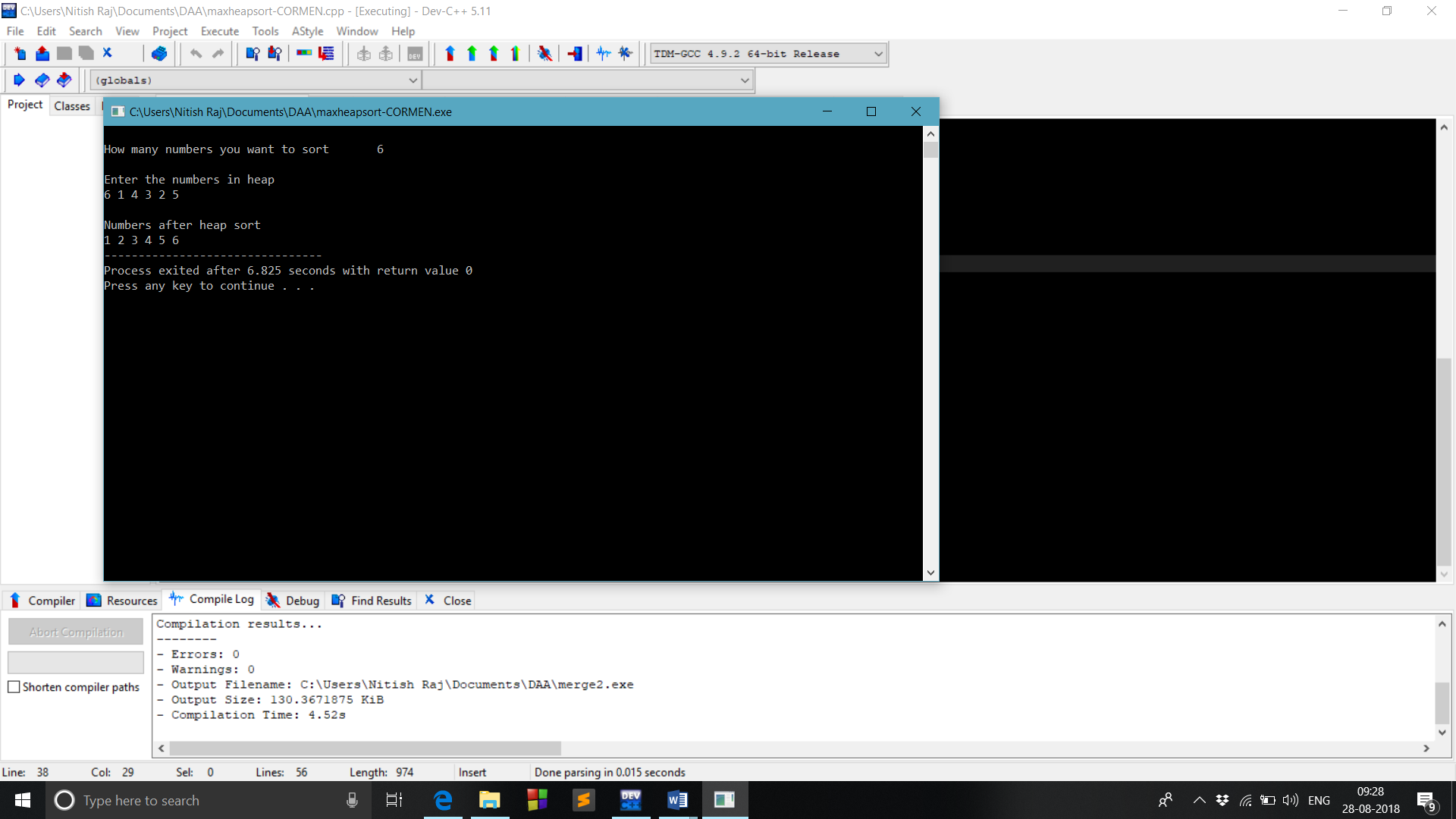
}

printf("\nNumbers after heap sort\n");

buildmaxheap(ar,n);

print(ar,n);

}



1. **Implementation of heap Sort Recursively.**

#include<stdio.h>

void swap(int ar[],int value1,int value2)

{

int temp=ar[value1];

ar[value1]=ar[value2];

ar[value2]=temp;

}

void maxheapify(int ar[],int index,int n)

{

int highindex=index;

if((2\*index<=n)&&ar[2\*index]>ar[highindex])

highindex=2\*index;

if((2\*index+1<=n)&&ar[2\*index+1]>ar[highindex])

highindex=2\*index+1;

if(highindex!=index)

{

swap(ar,index,highindex);

maxheapify(ar,highindex,n);

}

}

void buildmaxheap(int ar[],int n)

{

if(n<1)

return ;

for(int i=n/2;i>=1;i--)

maxheapify(ar,i,n);

for(int i=n;i>=1;i--)

{

swap(ar,1,i);

maxheapify(ar,1,i-1);

}

}

void print(int ar[],int n)

{

for(int i=1;i<=n;i++)

{

printf("%d ",ar[i]);

}

}

main()

{

int n;

printf("\nHow many numbers you want to sort ");

scanf("%d",&n);

int ar[n+1];

printf("\nEnter the numbers in heap\n");

for(int i=1;i<=n;i++)

{

scanf("%d",&ar[i]);

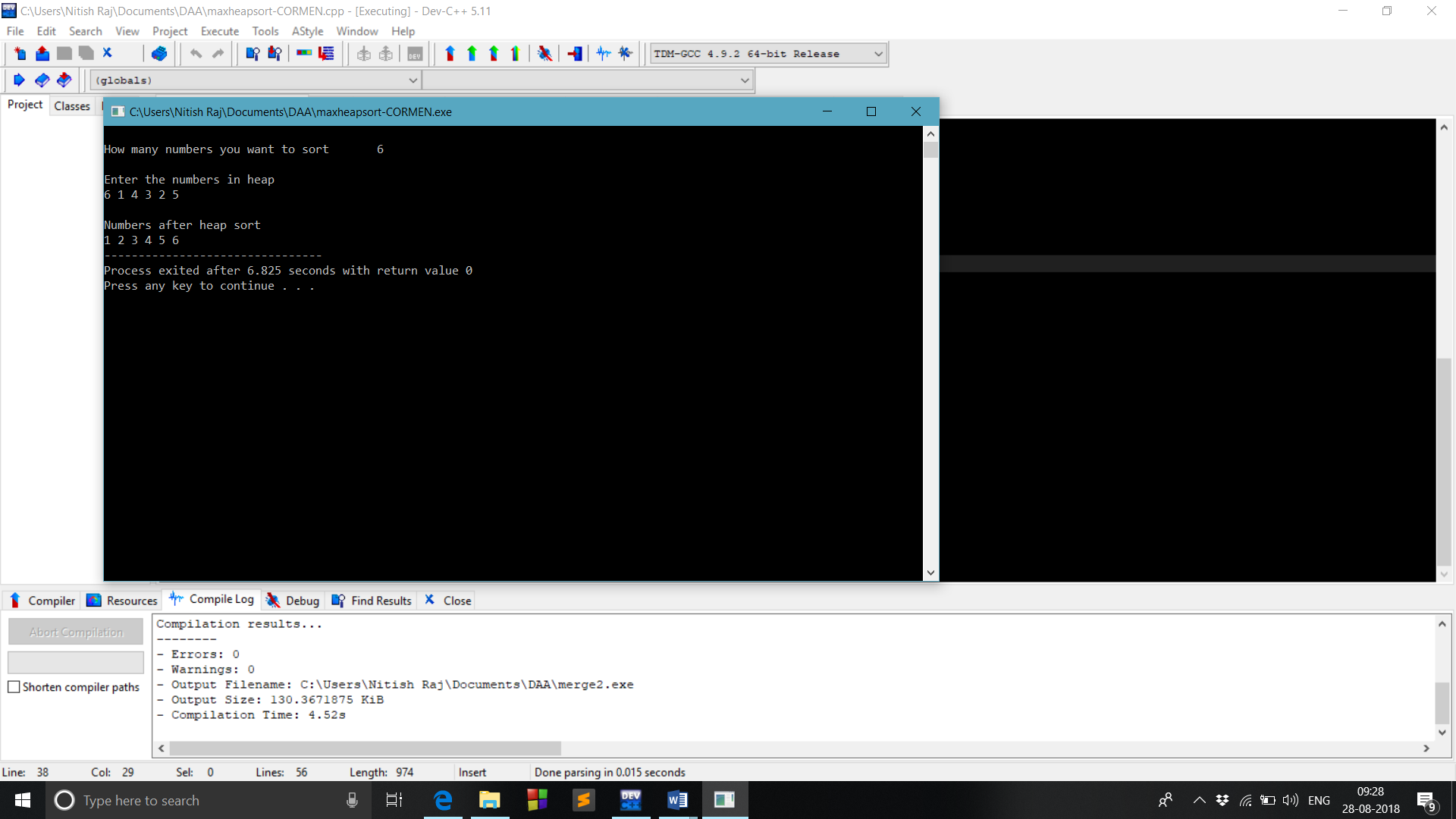
}

printf("\nNumbers after heap sort\n");

buildmaxheap(ar,n);

print(ar,n);

}



**Experiment no 5**

1. **Conversion of Infix to postfix**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

struct Stack

{

int top;

int cap;

int \*array;

};

struct Stack\* creatstack(int cap)

{

struct Stack\* stack=(struct Stack\*)malloc(6);

stack->top=-1;

stack->cap=cap;

stack->array=(int \*)malloc(sizeof(int)\*cap);

return stack;

}

int isfull(struct Stack \*stack)

{

if(stack->top==stack->cap-1)

return 1;

return 0;

}

int isempty(struct Stack \*stack)

{

if(stack->top==-1)

return 1;

return 0;

}

void push(struct Stack \*stack,char data)

{

if(!isfull(stack))

stack->array[++stack->top]=data;

}

char pop(struct Stack \*stack)

{

if(!isempty(stack))

return stack->array[stack->top--] ;

}

char peak(struct Stack \*stack)

{

if(!isempty(stack))

return stack->array[stack->top] ;

}

int pre(char ch)

{

switch(ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int isoperand(char ch)

{

return ch>='a'&&ch<='z' ||ch>='A'&&ch<='Z';

}

int isMatchingPair(char character1, char character2)

{

if (character1 == '(' && character2 == ')')

return 1;

else if (character1 == '{' && character2 == '}')

return 1;

else if (character1 == '[' && character2 == ']')

return 1;

else

return 0;

}

int intopost(struct Stack \*s,int l,char exp[])

{

int i,k=-1;

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

{

if(isoperand(exp[i]))

{

exp[++k]=exp[i];

}

else if (exp[i]=='(' || exp[i]=='{' || exp[i]=='[')

push(s,exp[i]);

else if(exp[i] == ')' || exp[i]=='}' || exp[i]==']')

{

while(!isempty(s) && peak(s)!='(' && peak(s)!='{' && peak(s)!='[' )

exp[++k]=pop(s);

if(isempty(s))

return 0;

else

if(!isMatchingPair(peak(s),exp[i] ))

return 0;

else

pop(s);

}

else

{

while(!isempty(s) && pre(exp[i]) <= pre(peak(s)))

exp[++k]=pop(s);

push(s,exp[i]);

}

}

while(!isempty(s) && peak(s)!='(' && peak(s)!='[' && peak(s)!='{')

exp[++k]=pop(s);

if(peak(s)=='(' || peak(s)=='{' || peak(s)=='[')

return 0;

exp[++k]='\0';

}

int main()

{

struct Stack \*s=creatstack(20);

char exp[20];

printf("enter the exression = ");

scanf("%s",&exp);

int l=strlen(exp);

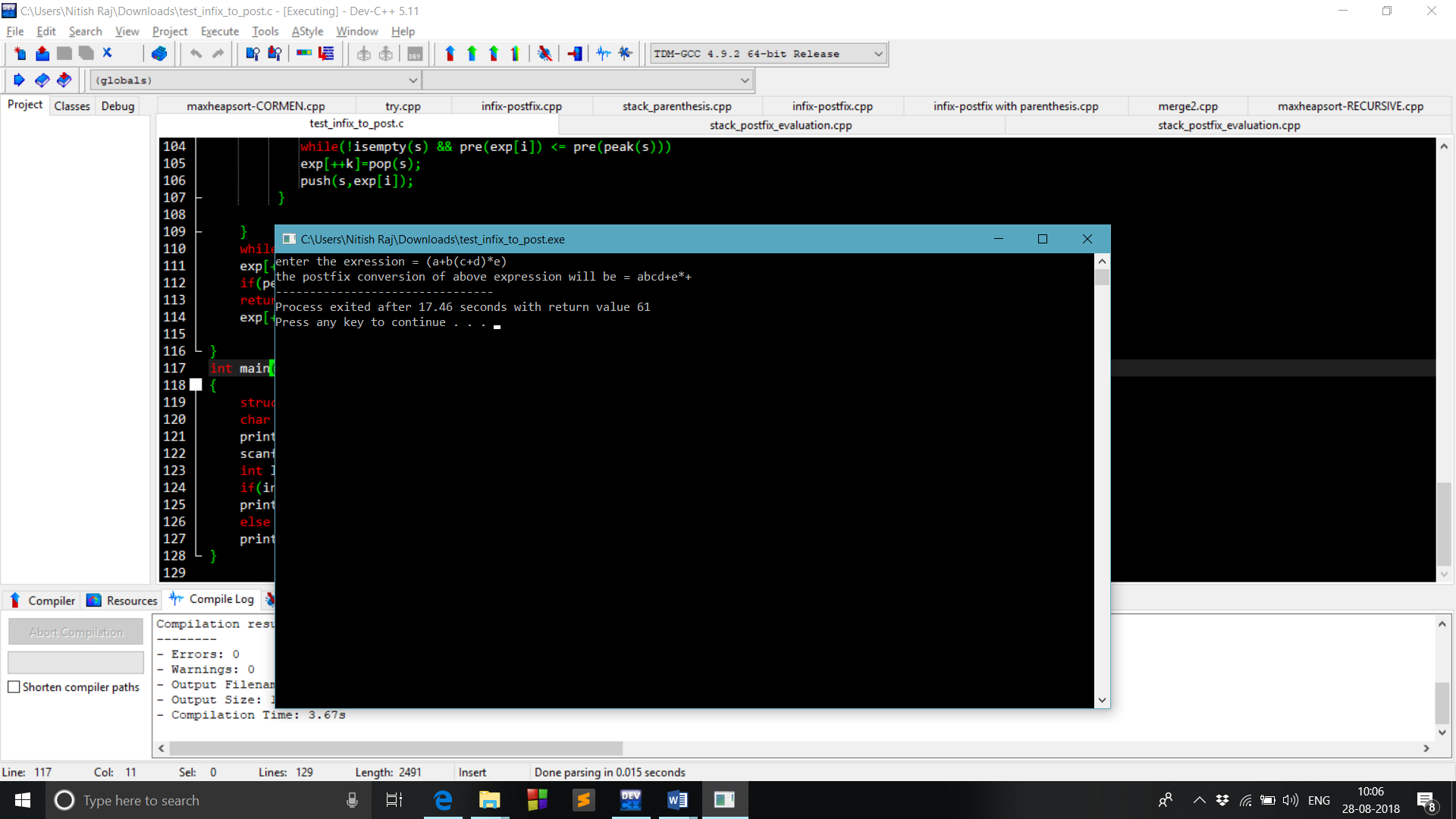
if(intopost(s,l,exp))

printf("the postfix conversion of above expression will be = %s",exp);

else

printf("invalid ");

}



1. **Postfix Evaluation**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct Stack

{

int top;

int cap;

char \*array;

};

struct Stack\* creatstack(int cap)

{

struct Stack \*stack=(struct Stack\*)malloc(sizeof(struct Stack));

stack->cap=cap;

stack->top=-1;

stack->array=(char \*)malloc(sizeof(char)\*cap);

return stack;

}

int isfull(struct Stack \*stack)

{

return stack->top==stack->cap-1;

}

int isempty(struct Stack \*stack)

{

return stack->top==-1;

}

void push(struct Stack \*stack,int data)

{

if(!isfull(stack))

stack->array[++stack->top]=data;

else

printf("\nstack full");

}

char pop(struct Stack \*stack)

{

if(!isempty(stack))

return stack->array[stack->top--];

else

printf("\nstack empty");

}

int post(struct Stack \*stack,char exp[],int l)

{

int i,a,b,c;

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

{

if(isdigit(exp[i]))

push(stack,exp[i]-'0');

else

{

a=pop(stack);

b=pop(stack);

switch(exp[i])

{

case '+':

c=a+b;

break;

case '-':

c=b-a;

break;

case '\*':

c=b\*a;

break;

case '/':

c=b/a;

break;

case '^' :

c =b^a;

default :

printf("\nInvalid Expression");

exit(0);

}

push(stack,c);

}

}

a=pop(stack);

if(!isempty(stack))

return 0;

return a;

}

int main()

{

struct Stack \*stack=creatstack(20);

char exp[20];

printf("\nenter the expression=");

scanf("%s",exp);

int l=strlen(exp);

int x=post(stack,exp,l);

if(x)

printf("the post fix evaluation of above expresson will be = %d",x);

else

printf("invalid expression");

}

