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");

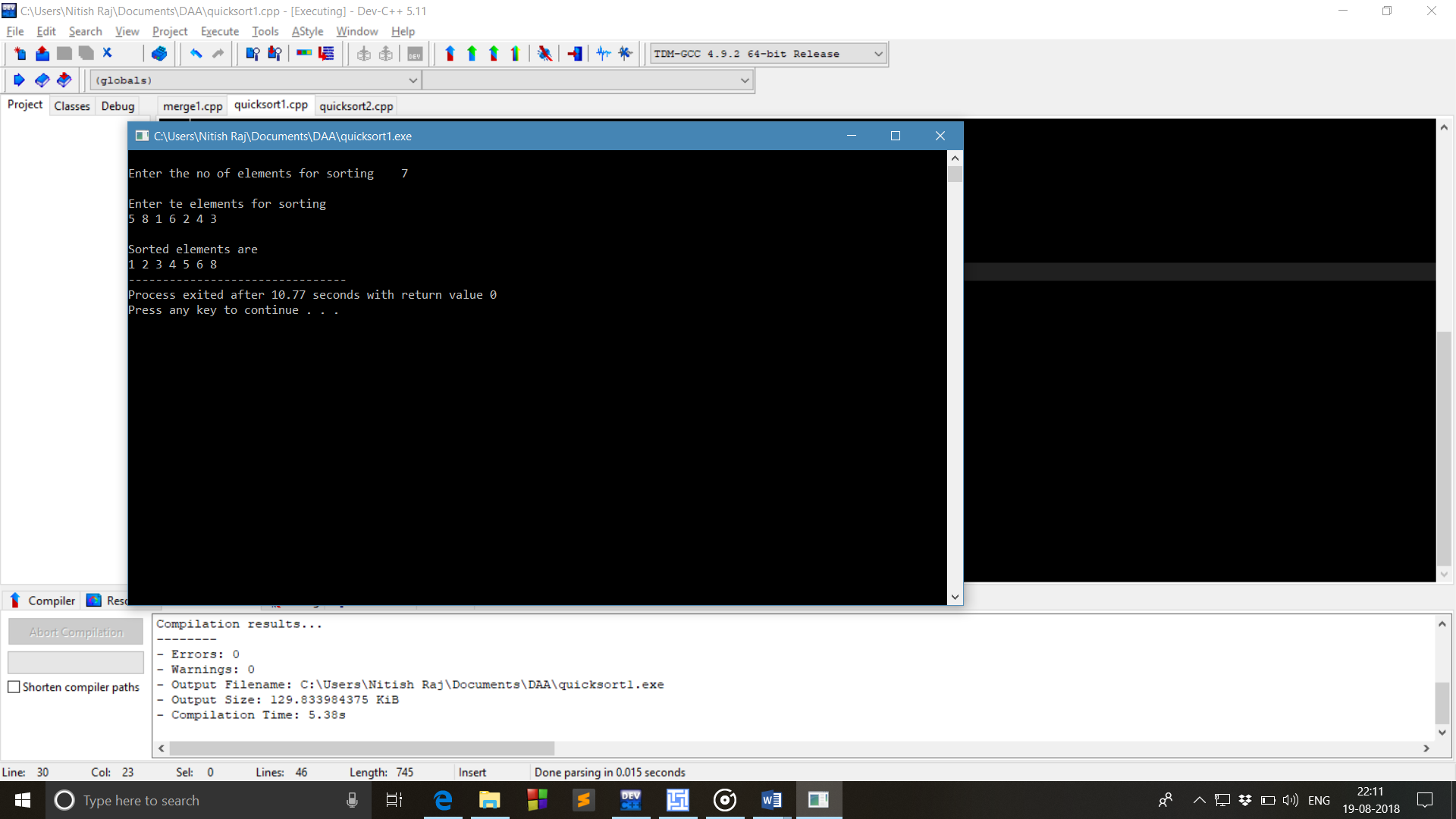
for(int i=0;i<n;i++)

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++)

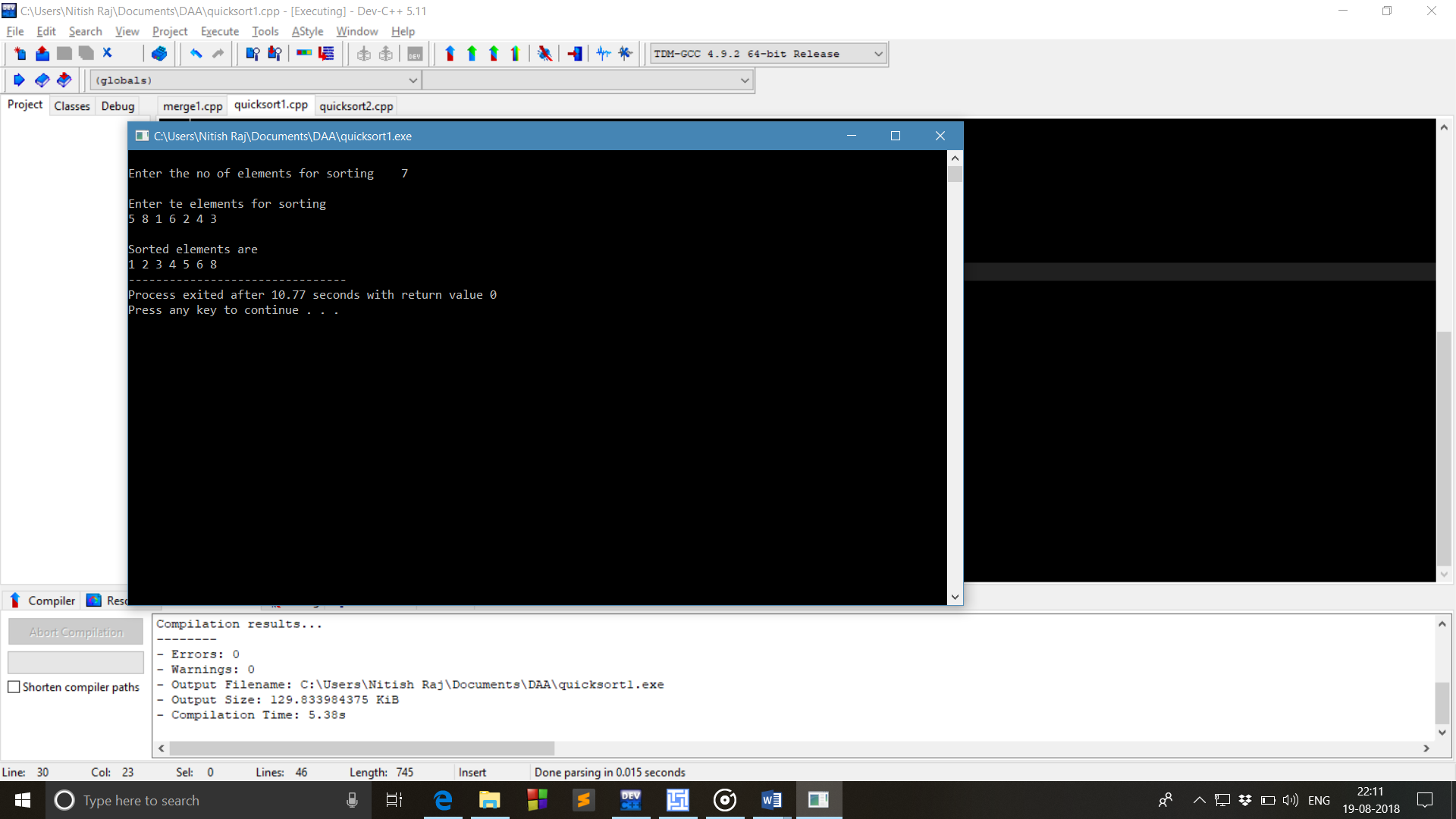
scanf("%d",&ar[i]);

quicksort(ar,0,n-1);

print(ar,n);

return 0;

}



1. **Program for the implementation of Merge sort**

#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);

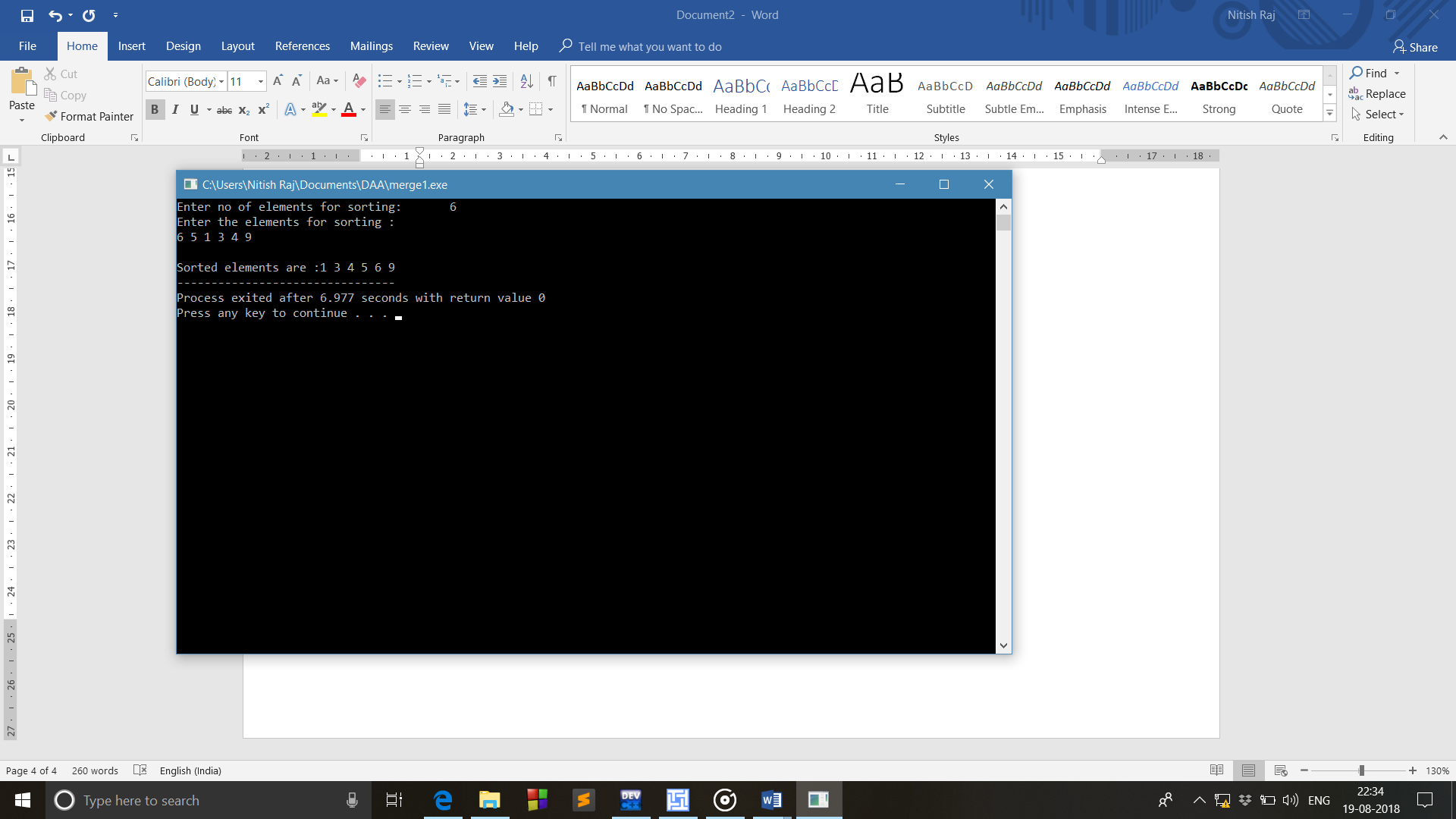
printf("\nSorted elements are :");

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

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

return 0;

}



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];

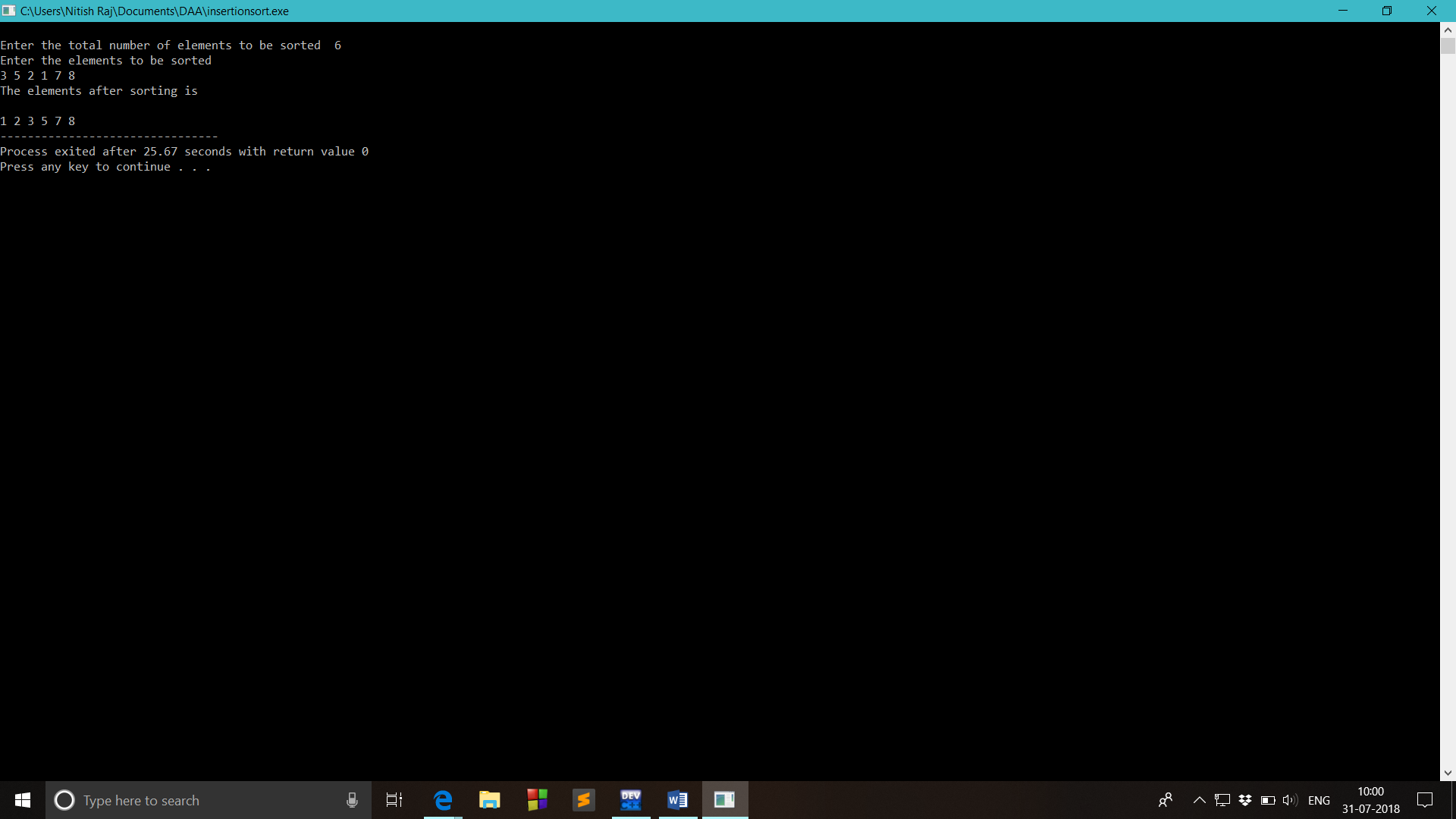
printf("Enter the elements to be sorted\n");

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++)

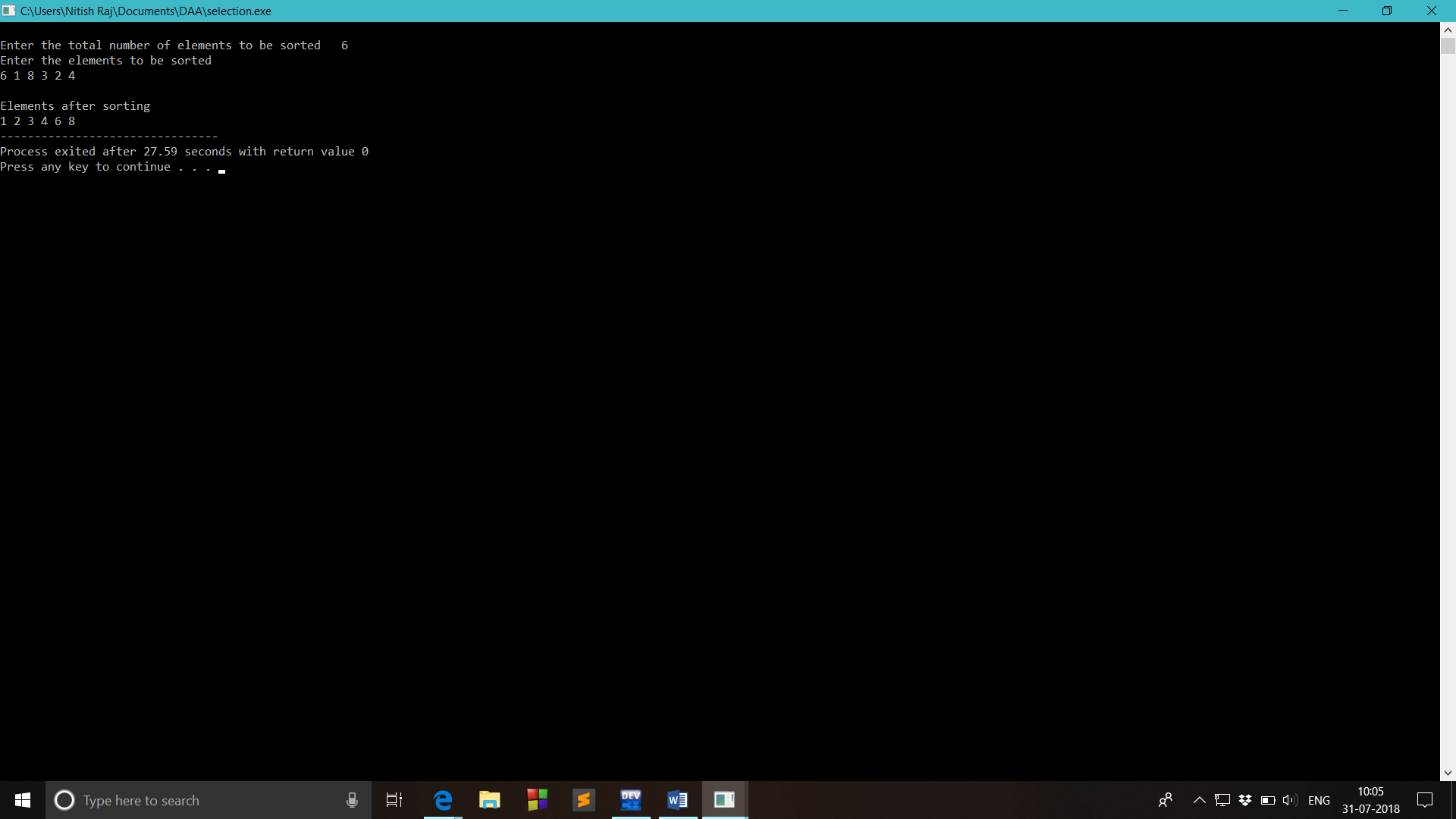
{

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]);

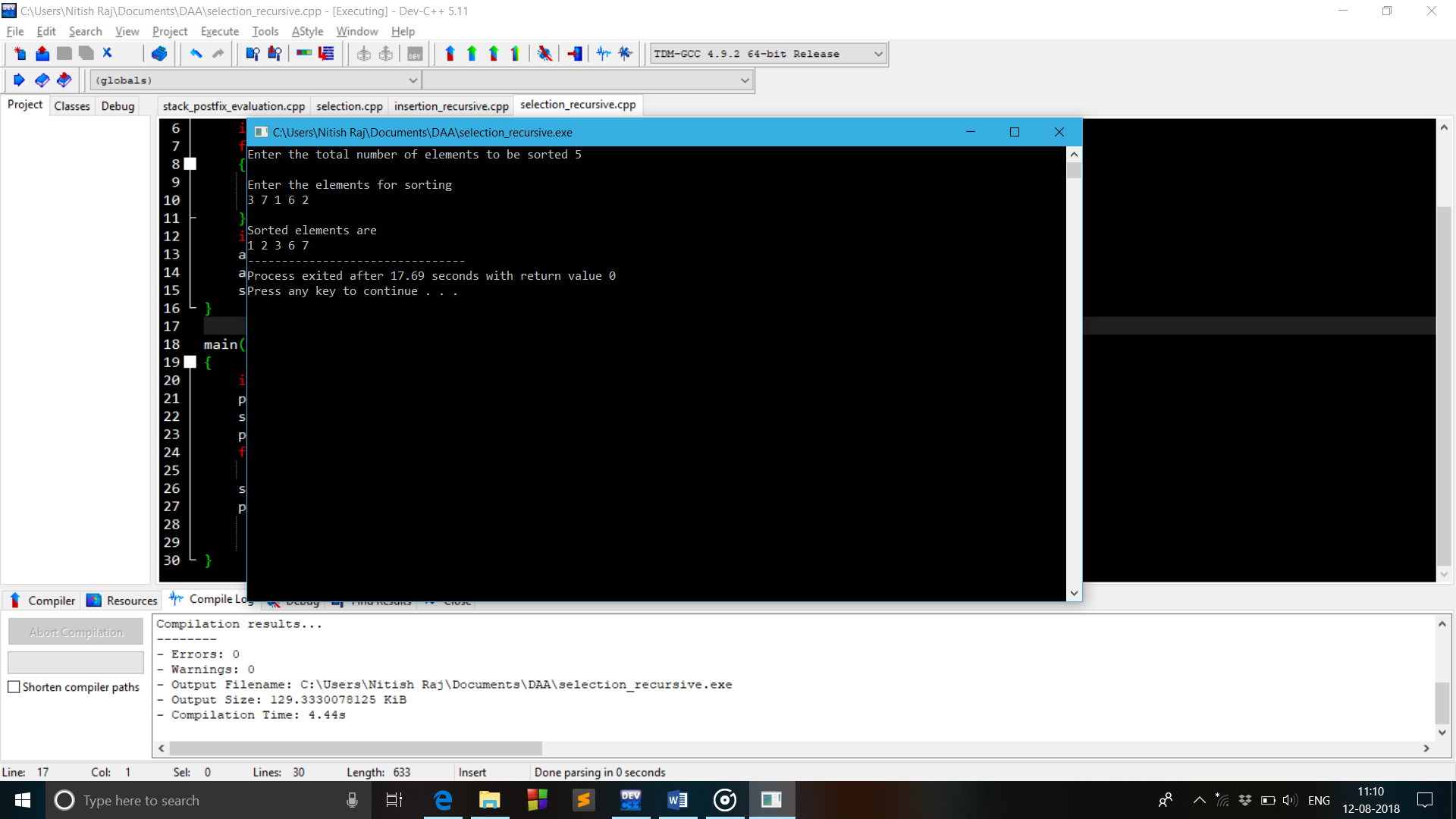
selectionsort(ar,0,n);

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]);

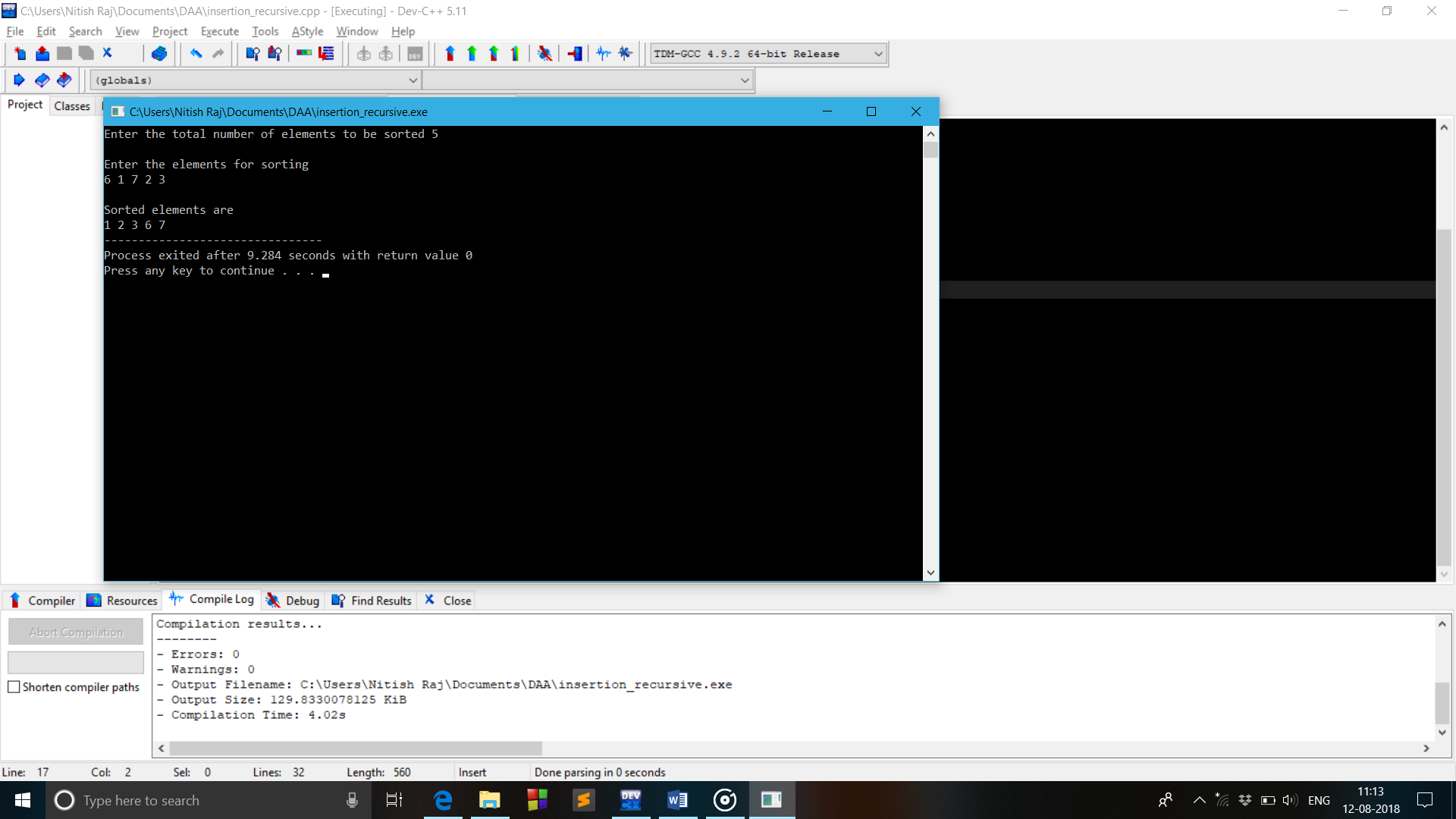
insertionsort(ar,1,n);

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

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

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

}



1. **Program for the Conversion of Infix to Postfix expression**

#include<stdio.h>

#include<ctype.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

void conversion(char e[])

{

printf(“\nExpression after converting from Infix to Postfix is \n”);

char x;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c",pop());

}

}

main()

{

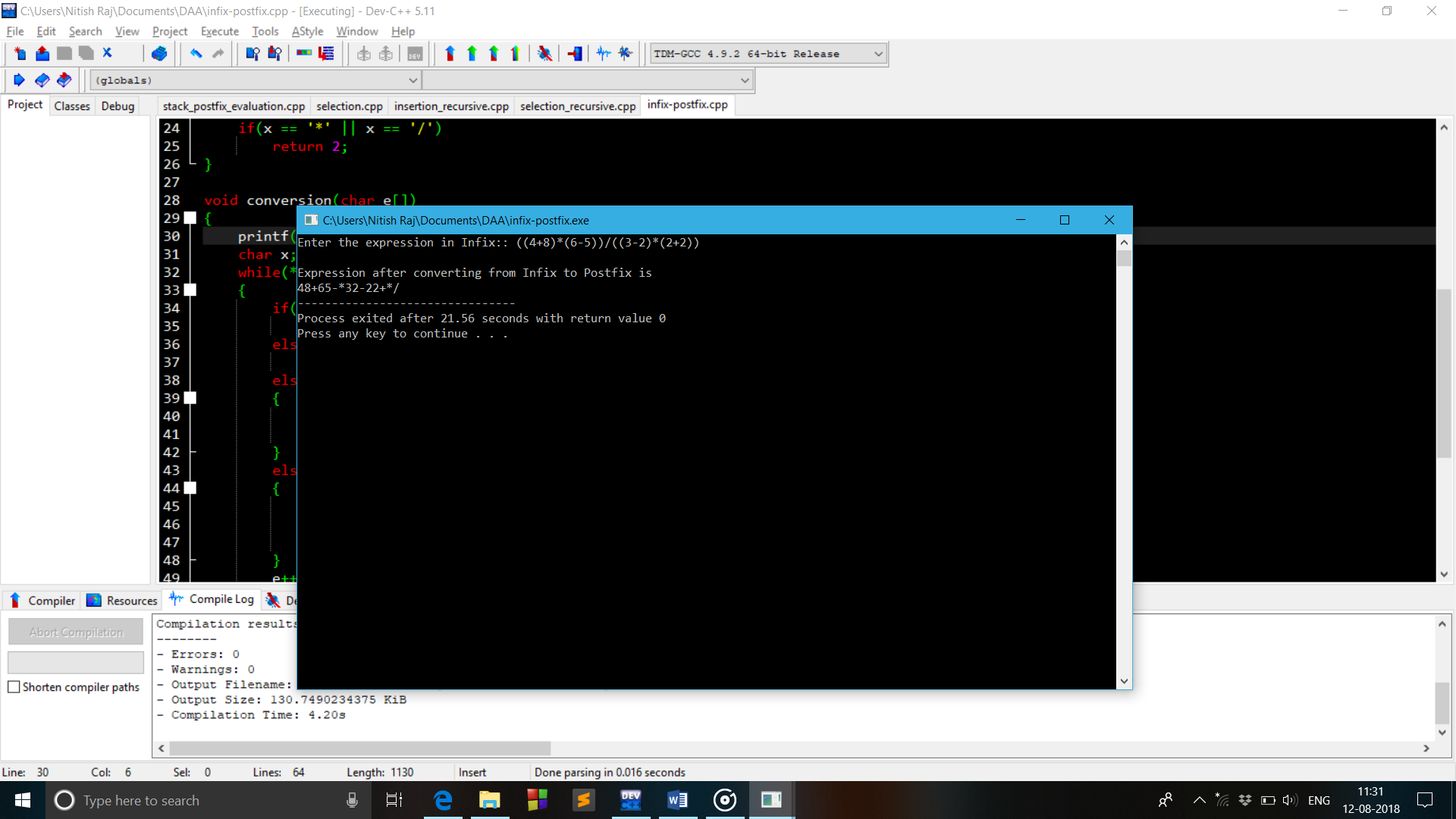
char exp[20];

printf("Enter the expression in Infix :: ");

scanf("%s",exp);

conversion(exp);

}



1. **Program for Evaluation of Postfix Expression**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

const int size=20;

int stack[size];

int top=-1;

int push(int x)

{

stack[++top]=x;

return 0;

}

int pop()

{

return(stack[top--]);

}

int postfixeval(char ar[],int size)

{

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

{

if(isdigit(ar[i]))

push(ar[i]-'0');

else

{

int n1=pop();

int n2=pop();

switch(ar[i])

{

case '+':push(n1+n2);

break;

case '-':push(n2-n1);

break;

case '\*':push(n2\*n1);

break;

case '/':push(n2/n1);

break;

default : printf("\nError!!");

break;

}

}

}

return stack[top];

}

int main()

{

char ar[size];

printf("Enter the postfix ");

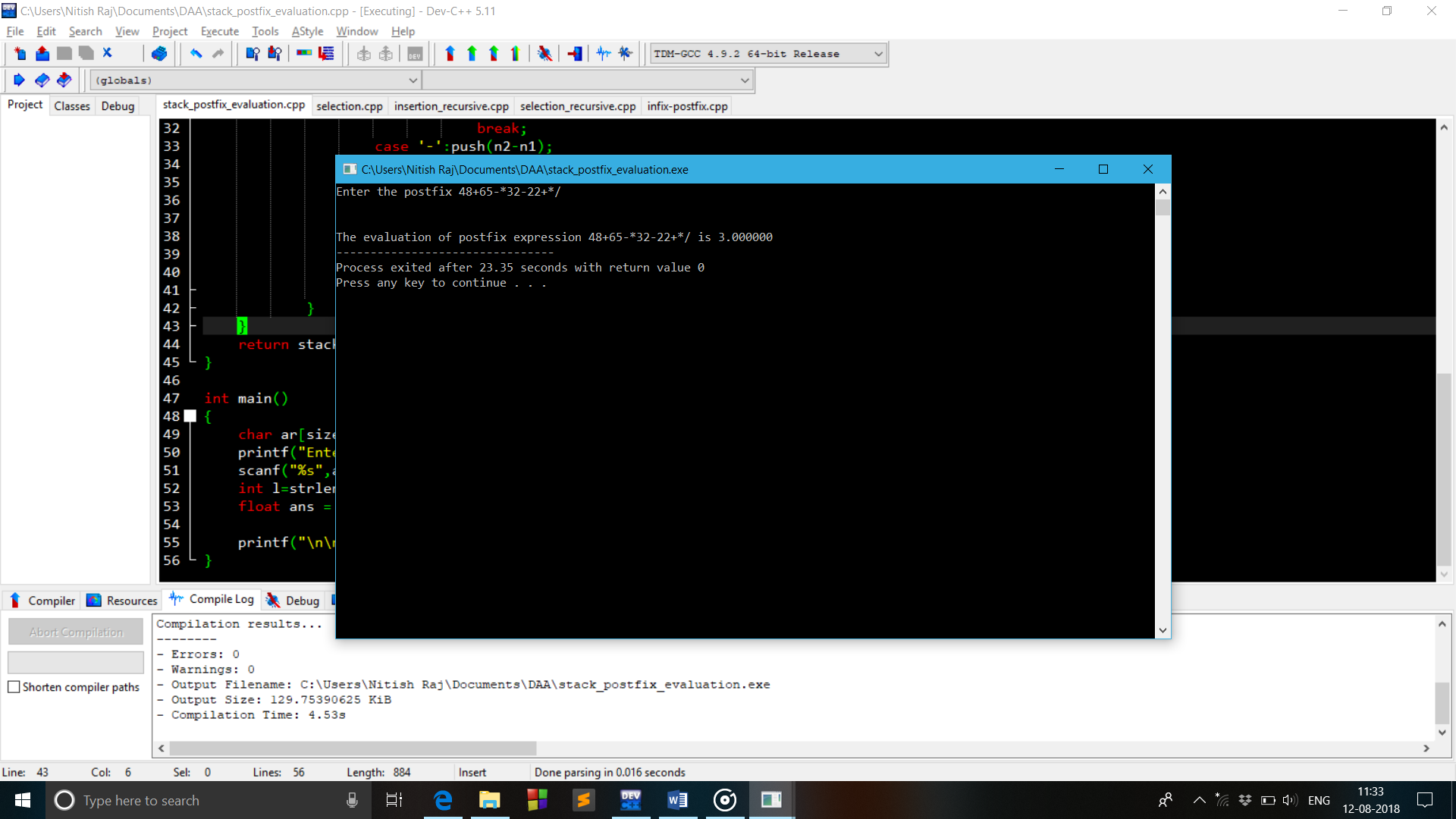
scanf("%s",ar);

int l=strlen(ar);

float ans = postfixeval(ar,l);

printf("\n\nThe evaluation of postfix expression %s is %f",ar,ans);

}



1. **Program for Checking of Correctness of Parenthesis.**

#include<stdio.h>

#include<string.h>

const int size=20;

int top=-1;

char stack[size];

void pop()

{

if(top==-1)

printf("\nUNDERFLOW...");

top--;

}

int push(char x)

{

if(top==size-1)

{

printf("\nOVERFLOW...");

return 0;

}

stack[++top]=x;

return 0;

}

int main()

{

char ar[size];

printf("\nEnter the string ");

scanf("%s",ar);

int l=strlen(ar);

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

{

if(ar[i]=='(')

push(ar[i]);

else if(ar[i]==')')

pop();

}

if(top==-1)

{

printf("\nString is correctly parenthesized");

return 0;

}

else

printf("\nString is Not parenthesized");

return 0;

}

