DATA STRUCTURES

MINI PROJECT

BRANCH: COMPUTER

CLASS: SE

DIV: B

MEMBERS:

ROLL NO. NAME

307 ANAGHA PATIL 316 SHIVPRASAD POOJARY 318 ABHISHEK PRAJAPATI 320 AAKASH RAJAWAT

PROBLEM DEFINITION:

A Graphical User Interface showing Bubble Sort Technique using linear data structures: Array in C programming.

Data Structures:

A **data structure** is a particular way of organizing data in a computer so that it can be used effectively. Data structures serve as the basis for abstract data types (ADT). The ADT defines the logical form of the data type. The data structure implements the physical form of the data type. Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services.

Types of Data Structures:

ARRAYS:

An **array**, is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index. The simplest type of data structure is a linear array, also called one-dimensional array. The memory address of the first element of an array is called first address, foundation address, or base address.

Arrays are useful mostly because the element indices can be computed at run time. Among other things, this feature allows a single iterative statement to process arbitrarily many elements of an array. For that reason, the elements of an array data structure are required to have the same size and should use the same data representation.

Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables. Many databases, small and large, consist of (or include) one-dimensional arrays whose elements are records.

Arrays are used to implement other data structures, such as lists, heaps, hash tables, deques, queues, stacks, strings. Array-based implementations of other data structures are frequently simple and space-efficient (implicit data structures), requiring little space overhead, but may have poor space complexity, particularly when modified, compared to tree-based data structures (compare a sorted array to a search tree).

REPRESENTATION OF ARRAY:

BUBBLE SORT:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Let us try to understand with an example: A [ ] = { 7, 4, 5, 2}

In step 1, 7 is compared with 4. Since 7>4, 7 is moved ahead of 4. Since all the other elements are of a lesser value than 7, 7 is moved to the end of the array.

Now the array is A[]={4,5,2,7}.

In step 2, 4 is compared with 5. Since 5>4 and both 4 and 5 are in ascending order, these elements are not swapped. However, when 5 is compared with 2, 5>2 and these elements are in descending order. Therefore, 5 and 2 are swapped.

Now the array is A[]={4,2,5,7}.

In step 3, the element 4 is compared with 2. Since 4>2 and the elements are in descending order, 4 and 2 are swapped.

The sorted array is A[]={2,4,5,7}.

Complexity: The complexity of bubble sort is O(n2) in both worst and average cases, because the entire array needs to be iterated for every element.

FUNCTIONS USED :

- **initgraph()**

  The method initgraph() initializes the graphics system by loading the graphics driver from disk(or validating a registered driver) then putting the system into graphics mode. The method initgraph() also resets all graphics settings (color,palette,current position,viewport) to their defaults.

- **cleardevice()**

  The header file graphics.h contains cleardevice() function which clears the screen in graphics mode and sets the current position to (0,0). Clearing the screen consists of filling the screen with current background color.

- **setbkcolor()**

Declaration: void setbkcolor(int color); setbkcolor function changes current background color e.g. setbkcolor(YELLLOW) changes the current background color to YELLOW.

- **circle()**

  The header file graphics.h contains circle() function which draws a circle with center at (x, y) and given radius. Syntax : circle(x, y, radius);

  where, (x, y) is center of the circle. 'radius' is the Radius of the circle.

- **sprintf()**

The sprintf() works just like printf() but instead of sending output to console it returns the formatting string.

Syntax: int sprintf(char *str, const char *control_string, [arg_1, arg_2, ...]);

➢ **outtextxy()**

The header file graphics.h contains outtextxy() function which displays the text or string at a specified point p(x,y) on the screen.
Syntax : void outtextxy(int x, int y, char *string);

where, x, y are coordinates of the point and, third argument contains the address of string to be displayed.

➢ **settextstyle()**

The header file graphics.h contains settextstyle() function which is used to change the way in which text appears. Using it we can modify the size of text, change direction of text and change the font of text. Syntax : void settextstyle(int font, int direction,
int font_size);

where, font argument specifies the font of text, Direction can be HORIZ_DIR (Left to right) or VERT_DIR (Bottom to top).

➢ **closegraph()**

closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you calle initgraph.

Declaration: void closegraph();

SOURCE CODE

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{ int

gd=DETECT;

int
swap,gm,i,j=60,c[6],m,e,d,col=1;

char a[5],r[20];

initgraph(&gd,&gm,"c:\\turboC3\\bgi");

for(i=0;i<5;i++)

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

} cleardevice();

setbkcolor(RED);

circle(80,200,20);

circle(140,200,20);
circle(200,200,20);

circle(260,200,20);

circle(320,200,20);

for(i=0;i<5;i++)

{ sprintf(a,"%d",c[i]);

outtextxy(15+j,200,a);

j=j+60;

} getch();

cleardevice();
setbkcolor(RED);

for(e=0;e<4;e++)

{

settextstyle(BOLD_FONT,HORIZ_DIR

,2);

sprintf(r,"round %d",col);

col++;

outtextxy(200,400,r);

settextstyle(DEFAULT_FONT,HORIZ_DI
R,1);

for(d=0;d<5-e-1;d++)
```

```
{ if(c[d]>c[d+1])

{ delay(1000);

if(d==0)

{

for(m=0;m<60;m++)

{ sprintf(a,"%d",c[2]);

outtextxy(195,200,a);

sprintf(a,"%d",c[3]);

outtextxy(255,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

circle(320,200,20);

circle(260,200,20);

circle(200,200,20);

if(m!=59)

{ delay(30);
cleardevice();

setbkcolor(RED);

sprintf(a,"%d",c[0]);
outtextxy(75+m,200,a);

sprintf(a,"%d",c[1]);

outtextxy(135-m,200,a);

circle(80+m,200,20);

circle(140-m,200,20);

}

circle(200,200,20);

circle(260,200,20);

circle(320,200,20);

sprintf(a,"%d",c[2]);

outtextxy(195,200,a);

sprintf(a,"%d",c[3]);

outtextxy(255,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

} swap=c[0];

c[0]=c[1];

c[1]=swap;

} if(d==1)
```

```
{

for(m=0;m<60;m++)

{ sprintf(a,"%d",c[0]);

outtextxy(75,200,a);
sprintf(a,"%d",c[3]);

outtextxy(255,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

circle(320,200,20);

circle(260,200,20);

circle(80,200,20);

if(m!=99)

{ delay(30);

cleardevice();

setbkcolor(RED);

sprintf(a,"%d",c[1]);

outtextxy(135+m,200,a);

sprintf(a,"%d",c[2]);

outtextxy(195-m,200,a);

circle(140+m,200,20);
circle(200-m,200,20);

}

circle(80,200,20);

circle(260,200,20);

circle(320,200,20);

sprintf(a,"%d",c[0]);

outtextxy(75,200,a);

sprintf(a,"%d",c[3]);

outtextxy(255,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

}
swap=c[2];

c[2]=c[1];

c[1]=swap;

} if(d==2)

{

for(m=0;m<60;m++)

{ sprintf(a,"%d",c[0]);

outtextxy(75,200,a);
```

```
sprintf(a,"%d",c[1]);

outtextxy(135,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

circle(320,200,20);

circle(140,200,20);

circle(80,200,20);

if(m!=59)

{ delay(30);

cleardevice();

setbkcolor(RED);

sprintf(a,"%d",c[2]);

outtextxy(195+m,200,a);

sprintf(a,"%d",c[3]);

outtextxy(255-m,200,a);

circle(200+m,200,20);

circle(260-m,200,20);

}
circle(80,200,20);

circle(140,200,20);
circle(320,200,20);

sprintf(a,"%d",c[0]);

outtextxy(75,200,a);

sprintf(a,"%d",c[1]);

outtextxy(135,200,a);

sprintf(a,"%d",c[4]);

outtextxy(315,200,a);

} swap=c[3];

c[3]=c[2];

c[2]=swap;

} if(d==3)

{

for(m=0;m<60;m++)

{ sprintf(a,"%d",c[0]);

outtextxy(75,200,a);

sprintf(a,"%d",c[1]);

outtextxy(135,200,a);

sprintf(a,"%d",c[2]);

outtextxy(195,200,a);
```

```
circle(200,200,20);

circle(140,200,20);

circle(80,200,20);

if(m!=59)

{
delay(30);

cleardevice();

setbkcolor(RED);

sprintf(a,"%d",c[4]);

outtextxy(315-m,200,a);

sprintf(a,"%d",c[3]);

outtextxy(255+m,200,a);

circle(320-m,200,20);

circle(260+m,200,20);

}

circle(80,200,20);

circle(140,200,20);

circle(200,200,20);

sprintf(a,"%d",c[0]);

outtextxy(75,200,a);
```

```
sprintf(a,"%d",c[1]);

outtextxy(135,200,a);

sprintf(a,"%d",c[2]);

outtextxy(195,200,a);

} swap=c[4];

c[4]=c[3];

c[3]=swap;

} } } }

getch();

closegraph();}
```

OUTPUT

REFERENCES

➢ Let Us C by Yashavant Kanetkar

➢ Data Structures Using C by Reema Thareja

➢ geeksforgeeks.org

➢ codeforwin.org