

KHUSHI MALIK

500119018

R2142230031

BATCH -2, SEM-3

DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

ALL LABS

● PATTERNS

1. RIGHT HALF PYRAMID(STAR)

The screenshot shows a code editor interface with a dark theme. On the left, there is a file navigation sidebar with icons for Python, C, C++, Java, JavaScript, Go, PHP, and CMake. The main area displays a C program named 'main.c'. The code uses nested loops to print a right half pyramid of stars. The output window shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j <= i; j++) {
9             printf("* ");
10        }
11    }
12    printf("\n");
13 }
14 return 0;
15 }
```

Output

```
/tmp/kb7G64pRne.o
*
* *
* * *
* * * *
* * * * *

== Code Execution Successful ==
```

2. RIGHT HALF PYRAMID(NUMBERS)

The screenshot shows a code editor interface with a dark theme. On the left, there is a file navigation sidebar with icons for Python, C, C++, Java, JavaScript, Go, PHP, and CMake. The main area displays a C program named 'main.c'. The code uses nested loops to print a right half pyramid of numbers. The output window shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j <= i; j++) {
9             printf("%d ", j + 1);
10        }
11    }
12    printf("\n");
13 }
14 return 0;
15 }
```

Output

```
/tmp/PQh6Siosh.o
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

== Code Execution Successful ==
```

3. RIGHT HALF PYRAMID(ALPHABETS)

The screenshot shows a code editor interface with a dark theme. On the left, there is a file navigation sidebar with icons for Python, C, C++, Java, JavaScript, Go, PHP, and CMake. The main area displays a C program named 'main.c'. The code uses nested loops to print a right half pyramid of alphabets. The output window shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j <= i; j++) {
9             printf("%c ", 'A' + j);
10        }
11    }
12    printf("\n");
13 }
14 return 0;
15 }
```

Output

```
/tmp/XuibEHIGed.o
A
A B
A B C
A B C D
A B C D E

== Code Execution Successful ==
```

4. INVERTED RIGHT HALF PYRAMID(STAR)

The screenshot shows a C code editor interface with a dark theme. The code in the main.c file is as follows:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i; j++) {
9             printf("* ");
10        }
11        printf("\n");
12    }
13    return 0;
14 }
```

The output window shows the execution results:

```
/tmp/UH6bEIMsTJ.o
* * * *
* * *
* *
*
*** Code Execution Successful ***
```

5. INVERTED RIGHT HALF PYRAMID(NUMBERS)

The screenshot shows a C code editor interface with a dark theme. The code in the main.c file is as follows:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i; j++) {
9             printf("%d ", j+1);
10        }
11        printf("\n");
12    }
13    return 0;
14 }
```

The output window shows the execution results:

```
/tmp/5a4PXqb7N.o
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
*** Code Execution Successful ***
```

6. INVERTED RIGHT HALF PYRAMID(ALPHABETS)

The screenshot shows a C code editor interface with a dark theme. The code in the main.c file is as follows:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i; j++) {
9             printf("%c ", 'A' + j);
10        }
11        printf("\n");
12    }
13    return 0;
14 }
```

The output window shows the execution results:

```
/tmp/bxQzN3GrPs.o
A B C D E
A B C D
A B C
A B
A
*** Code Execution Successful ***
```

7. LEFT HALF PYRAMID(STAR)

The screenshot shows a C code editor interface with a dark theme. The code in main.c prints a left half pyramid of stars. The output window shows the resulting star pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11        for (int k=0; k<=i; k++){
12            printf("* ");
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/5TzNuYMNw6.o
*
 *
 *
 *
=====
== Code Execution Successful ==
```

8. LEFT HALF PYRAMID(NUMBERS)

The screenshot shows a C code editor interface with a dark theme. The code in main.c prints a left half pyramid of numbers. The output window shows the resulting number pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11        for (int k=0; k<=i; k++){
12            printf("%d ",k+1);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/m5KsXwmZLV.o
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
=====
== Code Execution Successful ==
```

9. LEFT HALF PYRAMID(ALPHABETS)

The screenshot shows a C code editor interface with a dark theme. The code in main.c prints a left half pyramid of alphabets. The output window shows the resulting alphabet pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11        for (int k=0; k<=i; k++){
12            printf("%c ",'A'+k);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/EfZSDXygel.o
A
A B
A B C
A B C D
A B C D E
=====
== Code Execution Successful ==
```

10. INVERTED LEFT HALF PYRAMID(STAR)

The screenshot shows a code editor interface with a dark theme. On the left is a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11    }
12
13    for (int k = 0; k < rows - i ; k++){
14        printf("* ");
15    }
16    printf("\n");
17 }
18 return 0;
19 }
```

The output window on the right shows the execution results:

```
/tmp/kCZWrJ3q1K.o
* * * *
* * *
* *
*
*
*** Code Execution Successful ***
```

11. INVERTED LEFT HALF PYRAMID(NUMBERS)

The screenshot shows a code editor interface with a dark theme. On the left is a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11    }
12
13    for (int k = 0; k < rows - i ; k++){
14        printf("%d ",k+1);
15    }
16    printf("\n");
17 }
18 return 0;
19 }
```

The output window on the right shows the execution results:

```
/tmp/wGBxzL9YMu.o
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
*** Code Execution Successful ***
```

12. INVERTED LEFT HALF PYRAMID(ALPHABETS)

The screenshot shows a code editor interface with a dark theme. On the left is a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11    }
12
13    for (int k = 0; k < rows - i ; k++){
14        printf("%c ",'A'+k);
15    }
16    printf("\n");
17 }
18 return 0;
19 }
```

The output window on the right shows the execution results:

```
/tmp/2GhH192CZT.o
A B C D E
A B C D
A B C
A B
A
*** Code Execution Successful ***
```

13. FULL PYRAMID(STAR)

The screenshot shows a dark-themed online IDE interface. On the left, there's a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11    }
12    for (int k=0; k<2*i+1; k++){
13        printf("*");
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

On the right, the output window shows the execution results:

```
/tmp/qQg07qnS1i.o
*
* *
* * *
* * * *
* * * * *
* * * * * *

== Code Execution Successful ==
```

14. FULL PYRAMID(NUMBERS)

The screenshot shows a dark-themed online IDE interface. On the left, there's a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11    }
12    for (int k=0; k<2*i+1; k++){
13        printf("%d",k+1);
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

On the right, the output window shows the execution results:

```
/tmp/lCtQCKrlHx.o
1
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9

== Code Execution Successful ==
```

15. FULL PYRAMID(ALPHABETS)

The screenshot shows a dark-themed online IDE interface. On the left, there's a file browser with icons for Python, C, C++, Java, and JavaScript. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2*(rows-i)-1; j++) {
9             printf(" ");
10        }
11    }
12    for (int k=0; k<2*i+1; k++){
13        printf("%c ", 'A'+k);
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

On the right, the output window shows the execution results:

```
/tmp/mRpMK9Wa1W.o
A
A B C
A B C D E
A B C D E F G
A B C D E F G H I

== Code Execution Successful ==
```

16. INVERTED PYRAMID (STAR)

The screenshot shows a code editor interface with a dark theme. On the left is the code editor pane containing a file named 'main.c'. The code prints a 5-row inverted pyramid of asterisks. The output pane on the right shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < 2*(rows - i)-1 ; k++){
12            printf("*");
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/BGhAjaOmI0.o
* * * * *
* * * *
* * *
*
=====
== Code Execution Successful ==
```

17. INVERTED PYRAMID (NUMBERS)

The screenshot shows a code editor interface with a dark theme. On the left is the code editor pane containing a file named 'main.c'. The code prints a 5-row inverted pyramid of numbers from 1 to 5. The output pane on the right shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < 2*(rows - i)-1 ; k++){
12            printf("%c ", 'A'+k);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/yeB3YT1K3F.o
A B C D E F G H I
A B C D E F G
A B C D E
A B C
A
=====
== Code Execution Successful ==
```

18. INVERTED PYRAMID (ALPHABETS)

The screenshot shows a code editor interface with a dark theme. On the left is the code editor pane containing a file named 'main.c'. The code prints a 5-row inverted pyramid of uppercase letters from A to E. The output pane on the right shows the resulting pattern and a success message.

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < 2 * i; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < 2*(rows - i)-1 ; k++){
12            printf("%c ", 'A'+k);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

Output

```
/tmp/QEL3Knsi2G.o
A B C D E F G H I
A B C D E F G
A B C D E
A B C
A
=====
== Code Execution Successful ==
```

19. RHOMBUS(STAR)

The screenshot shows an online IDE interface with a dark theme. On the left is a file tree with icons for C, C++, Java, Python, and Go. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i-1; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < rows; k++){
12            printf("*");
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

The output window on the right shows the execution results:

```
/tmp/h4ERYqaH0D.o
*****
*** Code Execution Successful ===
```

20. RHOMBUS(NUMBERS)

The screenshot shows an online IDE interface with a dark theme. On the left is a file tree with icons for C, C++, Java, Python, and Go. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i-1; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < rows; k++){
12            printf("%d ",k+1);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

The output window on the right shows the execution results:

```
/tmp/JXDvnbBZbX.o
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
*** Code Execution Successful ===
```

21. RHOMBUS(ALPHABETS)

The screenshot shows an online IDE interface with a dark theme. On the left is a file tree with icons for C, C++, Java, Python, and Go. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows-i-1; j++) {
9             printf(" ");
10        }
11        for (int k = 0; k < rows; k++){
12            printf("%c ", 'A'+k);
13        }
14        printf("\n");
15    }
16    return 0;
17 }
```

The output window on the right shows the execution results:

```
/tmp/l1lLbRso07.o
A B C D E
A B C D E
A B C D E
A B C D E
A B C D E
*** Code Execution Successful ===
```

22. RECTANGLE(STARS)

The screenshot shows a C code editor interface with a dark theme. On the left is a sidebar with various icons for file operations like copy, paste, share, run, and save. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows; j++) {
9             printf("*");
10        }
11    }
12    for (int k = 0; k < rows; k++){
13        printf("\n");
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

The output window on the right shows the execution results:

```
/tmp/c16146gyIL.o
*****
*** Code Execution Successful ***
```

23. RECTANGLE(NUMBERS)

The screenshot shows a C code editor interface with a dark theme. On the left is a sidebar with various icons for file operations like copy, paste, share, run, and save. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows; j++) {
9             printf(" ");
10        }
11    }
12    for (int k = 0; k < rows; k++){
13        printf("%d ",k+1);
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

The output window on the right shows the execution results:

```
/tmp/fIV7QTIn2y.o
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
*** Code Execution Successful ***
```

24. RECTANGLE(ALPHABETS)

The screenshot shows a C code editor interface with a dark theme. On the left is a sidebar with various icons for file operations like copy, paste, share, run, and save. The main area contains a code editor with the following C code:

```
main.c
1 #include <stdio.h>
2 int main()
3 {
4     int rows = 5;
5
6     for (int i = 0; i < rows; i++) {
7
8         for (int j = 0; j < rows; j++) {
9             printf(" ");
10        }
11    }
12    for (int k = 0; k < rows; k++){
13        printf("%c ",'A'+k);
14    }
15    printf("\n");
16 }
17 return 0;
18 }
```

The output window on the right shows the execution results:

```
/tmp/M0B1Ft83vF.o
A B C D E
A B C D E
A B C D E
A B C D E
A B C D E
*** Code Execution Successful ***
```

25. DIAMOND(STARS)

The screenshot shows an online C compiler interface. The left sidebar lists various programming languages with their respective icons: C, C++, C#, Java, JavaScript, Go, PHP, Python, Swift, and R. The main area has tabs for "main.c" and "Output". The "Run" button is highlighted in blue. The "Output" tab shows the execution results.

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 6;
7
8     // first loop to print all rows
9     for (int i = 1; i <= rows; i++) {
10
11         // inner loop 1 to print white spaces
12         for (int j = 1; j <= (rows - i); j++) {
13             printf(" ");
14         }
15
16         // inner loop 2 to print star * character
17         for (int k = 1; k <= 2 * i - 1; k++) {
18             printf("*");
19         }
20         printf("\n");
21     }
22     for (int i = rows-1; i >=1; i--) {
23
24         // first inner loop for printing leading white
25         // spaces
26         for (int j = 1; j <=rows- i; j++) {
27             printf(" ");
28         }
29
30         // second inner loop for printing stars *
31         for (int k = 1; k <= 2 * i - 1; k++) {
32             printf("*");
33         }
34         printf("\n");
35     }
36     return 0;
37 }
```

The output window displays the printed diamond shape of stars:

```
*
 ***
 *****
 ******
 *****
 ****
 ***
 *
 === Code Execution Successful ===
```

26. DIAMOND(NUMBERS)

The screenshot shows an online C compiler interface. The left pane contains the source code for a C program named `main.c`. The right pane shows the output of the program's execution.

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 5;
7
8     for (int i = 1; i <= rows; i++) {
9
10        for (int j = 1; j <= (rows - i); j++) {
11            printf(" ");
12        }
13
14        for (int k = 1; k <= 2 * i - 1; k++) {
15            printf("%d", k+1);
16        }
17
18        printf("\n");
19    }
20
21    for (int i = rows-1; i >=1; i--) {
22
23        for (int j = 1; j <=rows- i; j++) {
24            printf(" ");
25        }
26
27    }
28
29
30    for (int k = 1; k <= 2 * i - 1; k++) {
31        printf("%d", k+1);
32    }
33
34    printf("\n");
35
36 }
```

The output window displays the following sequence of numbers, forming a diamond shape:

```
2
234
23456
2345678
2345678910
2345678
23456
234
2
```

Below the output, a message indicates the execution was successful:

```
== Code Execution Successful ==
```

27. DIAMOND(ALPHABETS)

The screenshot shows an online C compiler interface. The left pane contains the source code for a C program named `main.c`. The right pane shows the output of the program's execution.

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 5;
7
8     for (int i = 0; i <= rows; i++) {
9
10        for (int j = 0; j <= (rows - i); j++) {
11            printf(" ");
12        }
13
14        for (int k = 0; k <= 2 * i - 1; k++) {
15            printf("%c", 'A'+k);
16        }
17
18        printf("\n");
19    }
20
21    for (int i = rows-1; i >=1; i--) {
22
23        for (int j = 0; j <=rows- i; j++) {
24            printf(" ");
25        }
26
27    }
28
29
30    for (int k = 0; k <= 2 * i - 1; k++) {
31        printf("%c", 'A'+k);
32    }
33
34    printf("\n");
35
36 }
```

The output window displays the following sequence of alphabets, forming a diamond shape:

```
AB
ABCD
ABCDE
ABCDEF
ABCDEFH
ABCDEFHJ
ABCDEFH
ABCDE
ABCD
AB
```

Below the output, a message indicates the execution was successful:

```
== Code Execution Successful ==
```

28. HOURGLASS(STAR)

The screenshot shows an online C compiler interface. The left pane displays the source code in a text editor, and the right pane shows the output of the compiled program.

Code (main.c):

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 5;
7
8     for (int i = rows; i >= 1; i--) {
9
10        for (int j =0; j < (rows - i) ; j++) {
11            printf(" ");
12        }
13
14        for (int k = 1; k <= 2 * i - 1; k++) {
15            printf("*");
16        }
17        printf("\n");
18    }
19
20    for (int i = 2; i <=rows; i++) {
21
22        for (int j = 0; j <rows- i; j++) {
23            printf(" ");
24        }
25
26        for (int k = 1; k <= 2 * i - 1; k++) {
27            printf("*");
28        }
29
30        for (int k = 1; k <= 2 * i - 1; k++) {
31            printf("*");
32        }
33        printf("\n");
34    }
35    return 0;
36 }
```

Output:

```
/tmp/ZC36DjIDIO.o
*****
*****
*****
*
***
*****
*****
*****
*****
=====
== Code Execution Successful ==|
```

29. HOURGLASS(NUMBERS)

The screenshot shows an online C compiler interface. The code in the editor is:

```
main.c
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 5;
7
8     for (int i = rows; i >= 1; i--) {
9
10        for (int j = 0; j < (rows - i); j++) {
11            printf(" ");
12        }
13
14        for (int k = 0; k <= 2 * i - 1; k++) {
15            printf("%d", k+1);
16        }
17        printf("\n");
18    }
19
20    for (int i = 2; i <=rows; i++) {
21
22        for (int j = 0; j <rows- i; j++) {
23            printf(" ");
24        }
25
26        for (int j = 0; j <rows- i; j++) {
27            printf(" ");
28        }
29
30        for (int k = 0; k <= 2 * i - 1; k++) {
31            printf("%d", k+1);
32        }
33        printf("\n");
34    }
35
36 }
```

The output window shows the generated numbers:

```
/tmp/zwafxbft4n.o
12345678910
12345678
123456
1234
123
1234
123456
12345678
12345678910
== Code Execution Successful ==
```

30. HOURGLASS(ALPHABETS)

The screenshot shows an online C compiler interface. The code in the editor is:

```
main.c
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 int main()
5 {
6     int rows = 5;
7
8     for (int i = rows; i >= 1; i--) {
9
10        for (int j = 0; j < (rows - i); j++) {
11            printf(" ");
12        }
13
14        for (int k = 0; k <= 2 * i - 1; k++) {
15            printf("%c", 'A'+k);
16        }
17        printf("\n");
18    }
19
20    for (int i = 2; i <=rows; i++) {
21
22        for (int j = 0; j <rows- i; j++) {
23            printf(" ");
24        }
25
26        for (int k = 0; k <= 2 * i - 1; k++) {
27            printf("%c", 'A'+k);
28        }
29
30        for (int k = 0; k <= 2 * i - 1; k++) {
31            printf("%c", 'A'+k);
32        }
33        printf("\n");
34    }
35
36 }
```

The output window shows the generated alphabets:

```
/tmp/H15ZmZSzZH.o
ABCDEFGHIJ
ABCDEFGHI
ABCDEFGHI
ABCD
AB
ABCD
ABCDEF
ABCDEFGHIJ
== Code Execution Successful ==
```

31. PASCAL'S TRIANGLE

The screenshot shows a code editor interface with a dark theme. On the left, the code file 'main.c' is displayed with line numbers from 3 to 25. The code implements a function to print Pascal's Triangle. It uses nested loops to calculate binomial coefficients and prints them using printf. The output window on the right shows the execution of the program, prompting for 'Enter number of rows:' and then displaying the triangle for 5 rows.

```
main.c
3 int main() {
4     int rows, coef = 1, i, j;
5     printf("Enter number of rows: ");
6     scanf("%d", &rows);
7
8     for(i = 0; i < rows; i++) {
9         for(j = 1; j <= rows - i; j++) {
10            printf(" ");
11        }
12        for(j = 0; j <= i; j++) {
13            if (j == 0 || i == j) {
14                coef = 1;
15            } else {
16                coef = coef * (i - j + 1) / j;
17            }
18            printf("%4d", coef);
19        }
20        printf("\n");
21    }
22
23    return 0;
24 }
25
```

/tmp/Eogx4tMCMz.o
Enter number of rows: 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
== Code Execution Successful ==

32. HOLLOW DIAMOND(STARS)

33. HOLLOW DIAMOND(NUMBERS)

```
main.c
```

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, n;
5     printf("Enter the number of rows: ");
6     scanf("%d", &n);
7
8     // Upper half
9     for (i = 1; i <= n; i++) {
10        for (j = 1; j <= n - i; j++) {
11            printf(" ");
12        }
13        for (j = 1; j <= 2 * i - 1; j++) {
14            if (j == 1 || j == 2 * i - 1) {
15                printf("%d", i);
16            } else {
17                printf(" ");
18            }
19        }
20        printf("\n");
21    }
22
23    // Lower half
24    for (i = n - 1; i >= 1; i--) {
25        for (j = 1; j <= n - i; j++) {
26            printf(" ");
27        }
28        for (j = 1; j <= 2 * i - 1; j++) {
29            if (j == 1 || j == 2 * i - 1) {
30                printf("%d", i);
31            } else {
32                printf(" ");
33            }
34        }
35        printf("\n");
36    }
37
38    return 0;
39 }
40
```

RunShare

```
/tmp/hJHqoe0jv.o
Enter the number of rows: 10
      1
    2 2
  3 3
4 4
  5 5
    6 6
      7
    8 8
  9 9
10 10
  9 9
    8 8
  7 7
    6 6
  5 5
    4 4
  3 3
    2 2
      1

==== Code Execution Successful ====

```

```
      5 5
    6 6
      7 7
    8 8
  9 9
10 10
  9 9
    8 8
  7 7
    6 6
  5 5
    4 4
  3 3
    2 2
      1

==== Code Execution Successful ====

```

34. HOLLOW DIAMOND(ALPHABETS)

```
main.c
```

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, n;
5     char ch;
6     printf("Enter the number of rows: ");
7     scanf("%d", &n);
8
9     // Upper half
10    for (i = 1; i <= n; i++) {
11        ch = 'A' + i - 1;
12        for (j = 1; j <= n - i; j++) {
13            printf(" ");
14        }
15        for (j = 1; j <= 2 * i - 1; j++) {
16            if (j == 1 || j == 2 * i - 1) {
17                printf("%c", ch);
18            } else {
19                printf(" ");
20            }
21        }
22        printf("\n");
23    }
24
25    // Lower half
26    for (i = n - 1; i >= 1; i--) {
27        ch = 'A' + i - 1;
28        for (j = 1; j <= n - i; j++) {
29            printf(" ");
30        }
31        for (j = 1; j <= 2 * i - 1; j++) {
32            if (j == 1 || j == 2 * i - 1) {
33                printf("%c", ch);
34            } else {
35                printf(" ");
36            }
37        }
38        printf("\n");
39    }
40
41    return 0;
42 }
```

```
43
```

RunShareOutput

```
/tmp/bjvcuotbRr.o
Enter the number of rows: 5
      A
      B B
      C   C
      D   D
      E   E
      D   D
      C   C
      B B
      A

==== Code Execution Successful ====

Enter the number of rows: 5
      A
      B B
      C   C
      D   D
      E   E
      D   D
      C   C
      B B
      A

==== Code Execution Successful ====

```

35. HOLLOW PYRAMID(STARS)

The screenshot shows a code editor interface with a dark theme. The left panel displays the source code for 'main.c'. The right panel shows the output of the program execution.

Code (main.c):

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, rows;
5     printf("Enter the number of rows: ");
6     scanf("%d", &rows);
7
8     for (i = 1; i <= rows; i++) {
9         for (j = 1; j <= rows - i; j++) {
10            printf(" ");
11        }
12        for (j = 1; j <= 2 * i - 1; j++) {
13            if (j == 1 || j == 2 * i - 1 || i == rows) {
14                printf("*");
15            } else {
16                printf(" ");
17            }
18        }
19        printf("\n");
20    }
21
22    return 0;
23 }
```

Output:

```
/tmp/CJzFpmZLiV.o
Enter the number of rows: 6
*
* *
*   *
*   *
*****
== Code Execution Successful ==
```

36. HOLLOW PYRAMID(NUMBERS)

The screenshot shows a code editor interface with a dark theme. The left panel displays the source code for 'main.c'. The right panel shows the output of the program execution.

Code (main.c):

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, rows;
5     printf("Enter the number of rows: ");
6     scanf("%d", &rows);
7
8     for (i = 1; i <= rows; i++) {
9         for (j = 1; j <= rows - i; j++) {
10            printf(" ");
11        }
12        for (j = 1; j <= 2 * i - 1; j++) {
13            if (j == 1 || j == 2 * i - 1 || i == rows) {
14                printf("%d", i);
15            } else {
16                printf(" ");
17            }
18        }
19        printf("\n");
20    }
21
22    return 0;
23 }
```

Output:

```
/tmp/M00ire2wwN.o
Enter the number of rows: 5
1
2 2
3 3
4 4
555555555
== Code Execution Successful ==
```

37. HOLLOW PYRAMID(ALPHABET)

The screenshot shows a code editor interface with a dark theme. On the left, the code file 'main.c' is displayed with line numbers from 1 to 24. The code itself is a C program that prints a hollow pyramid pattern of alphabets. It uses nested loops to iterate through rows and columns, and conditional statements to determine whether to print an alphabet or a space based on the row and column indices. On the right, the 'Output' panel shows the execution results. It starts with the command '/tmp/Vd9ydCaKu8.o', followed by the user input 'Enter the number of rows: 8'. The output then displays a hollow pyramid pattern with 8 rows, where each row contains two alphabets ('A' through 'G') separated by spaces. Below the pyramid, the message '===== Code Execution Successful =====' is printed.

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, rows;
5     char ch;
6     printf("Enter the number of rows: ");
7     scanf("%d", &rows);
8
9     for (i = 1; i <= rows; i++) {
10        for (j = 1; j <= rows - i; j++) {
11            printf(" ");
12        }
13        for (j = 1; j <= 2 * i - 1; j++) {
14            if (j == 1 || j == 2 * i - 1 || i == rows) {
15                printf("%c", 'A' + i - 1);
16            } else {
17                printf(" ");
18            }
19        }
20        printf("\n");
21    }
22
23    return 0;
24 }
```

/tmp/Vd9ydCaKu8.o
Enter the number of rows: 8
A
B B
C C
D D
E E
F F
G G
HHHHHHHHHHHHHHHH
===== Code Execution Successful =====

38. HOLLOW FULL PYRAMID(STARS)

The screenshot shows a code editor interface with a dark theme. On the left, the code file 'main.c' is displayed with line numbers from 1 to 23. The code is a C program that prints a hollow full pyramid pattern of asterisks (*). It uses nested loops to iterate through rows and columns, and conditional statements to determine whether to print a star or a space based on the row and column indices. On the right, the 'Output' panel shows the execution results. It starts with the command '/tmp/Ntfdw6azvR.o', followed by the user input 'Enter the number of rows: 5'. The output then displays a hollow full pyramid pattern with 5 rows, where each row contains three stars (*) separated by spaces. Below the pyramid, the message '===== Code Execution Successful =====' is printed.

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j, rows;
5     printf("Enter the number of rows: ");
6     scanf("%d", &rows);
7
8     for (i = 1; i <= rows; i++) {
9        for (j = 1; j <= rows - i; j++) {
10            printf(" ");
11        }
12        for (j = 1; j <= 2 * i - 1; j++) {
13            if (j == 1 || j == 2 * i - 1 || i == rows) {
14                printf("*");
15            } else {
16                printf(" ");
17            }
18        }
19        printf("\n");
20    }
21
22    return 0;
23 }
```

/tmp/Ntfdw6azvR.o
Enter the number of rows: 5
*
* *
* * *

===== Code Execution Successful =====

39. HOLLOW FULL PYRAMID(NUMBERS)

The screenshot shows a code editor interface with a dark theme. On the left, the code file 'main.c' is displayed with line numbers from 1 to 23. The code itself is a C program that prompts the user for the number of rows and then prints a hollow full pyramid of numbers. On the right, there is an 'Output' panel showing the execution results. The output starts with the command 'Run' and ends with '==== Code Execution Successful ==='. The user input 'Enter the number of rows: 7' is followed by the printed pyramid:

```
1  
2 2  
3 3  
4 4  
5 5  
6 6  
7777777777777
```

40. HOLLOW FULL PYRAMID(ALPHABETS)

The screenshot shows a code editor interface with a dark theme. On the left, the code file 'main.c' is displayed with line numbers from 1 to 23. The code is a C program that prints a hollow full pyramid of alphabets. On the right, the 'Output' panel shows the execution results. The user input 'Enter the number of rows: 9' is followed by the printed pyramid:

```
A  
B B  
C C  
D D  
E E  
F F  
G G  
H H  
I I I I I I I I I  
==== Code Execution Successful ===
```

● LAB-1

1. Find sum of all array elements using recursion.

The screenshot shows a code editor window with a dark theme. The file is named '1.c'. The code defines a recursive function 'sumOfArray' and a main function. The terminal tab is selected, showing the execution of the program and its output.

```
C 1.c      X
C 1.c > ⚏ main()
1   #include <stdio.h>
2   int sumOfArray(int arr[], int n) {
3       if (n == 0)
4           return 0;
5       return arr[n - 1] + sumOfArray(arr, n - 1);
6   }
7
8   int main() {
9       int n;
10      printf("Enter the number of elements in the array: ");
11      scanf("%d", &n);
12
13      int arr[n];
14      printf("Enter %d elements:\n", n);
15      for (int i = 0; i < n; i++) {
16          scanf("%d", &arr[i]);
17      }
18      int result = sumOfArray(arr, n);
19      printf("The sum of all array elements is: %d\n", result);
20
21      return 0;
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
}
```

The sum of all array elements is: 15
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\Use
}

Enter the number of elements in the array: 6
Enter 6 elements:
2
3
4
1
2
3
The sum of all array elements is: 15
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>

2. Create an array ‘a1’ with ‘n’ elements. Insert an element in ith position of ‘a1’ and also delete an element from jth position of ‘a1’.

```
C 2.c x
C 2.c > ...
1 #include <stdio.h>
2 int main() {
3     int n, i, j, element;
4
5     printf("Enter the number of elements in the array: ");
6     scanf("%d", &n);
7
8     int arr[100];
9
10    printf("Enter %d elements:\n", n);
11    for (int k = 0; k < n; k++) {
12        scanf("%d", &arr[k]);
13    }
14
15    printf("Enter the position (0-based index) to insert an element: ");
16    scanf("%d", &i);
17    printf("Enter the element to insert: ");
18    scanf("%d", &element);
19
20    for (int k = n; k > i; k--) {
21        arr[k] = arr[k - 1];
22    }
23    arr[i] = element;
24    n++;
25
26    printf("Array after insertion:\n");
27    for (int k = 0; k < n; k++) {
28        printf("%d ", arr[k]);
29    }
30    printf("\n");
31
32    printf("Enter the position (0-based index) to delete an element: ");
33    scanf("%d", &j);
34
35    for (int k = j; k < n - 1; k++) {
36        arr[k] = arr[k + 1];
37    }
38    n--;
39
40    printf("Array after deletion:\n");
41    for (int k = 0; k < n; k++) {
42        printf("%d ", arr[k]);
```

```
35
40     printf("Array after deletion:\n");
41     for (int k = 0; k < n; k++) {
42         printf("%d ", arr[k]);
43     }
44     printf("\n");
45
46     return 0;
47 }
48
```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\U
}
Enter the number of elements in the array: 5
Enter 5 elements:
1
2
3
4
5
Enter the position (0-based index) to insert an element: 3
Enter the element to insert: 9
Array after insertion:
1 2 3 9 4 5
Enter the position (0-based index) to delete an element: 2
Array after deletion:
1 2 9 4 5
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

3. Convert uppercase string to lowercase using for loop.

The screenshot shows a code editor window with a dark theme. At the top, it says "C 3.c" and has a close button "X". Below the editor area, there are tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The code in the editor is:

```
C 3.c > main()
1 #include <stdio.h>
2
3 int main() {
4     char str[100];
5
6     printf("Enter an uppercase string: ");
7     scanf("%s", str);
8
9     for (int i = 0; str[i] != '\0'; i++) {
10         if (str[i] >= 'A' && str[i] <= 'Z') {
11             str[i] = str[i] + ('a' - 'A');
12         }
13     }
14     printf("Lowercase string: %s\n", str);
15
16     return 0;
17 }
```

Below the editor, the terminal output is displayed:

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd
}
Enter an uppercase string: TILAKYADAV
Lowercase string: tilakyadav
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> 
```

4. Find the sum of rows and columns of matrix of given order (row x column).

C 4.c

C 4.c > ...

```
1 #include <stdio.h>
2 int main() {
3     int rows, columns;
4     printf("Enter the number of rows: ");
5     scanf("%d", &rows);
6     printf("Enter the number of columns: ");
7     scanf("%d", &columns);
8
9     int matrix[10][10];
10    printf("Enter the elements of the matrix:\n");
11    for (int i = 0; i < rows; i++) {
12        for (int j = 0; j < columns; j++) {
13            scanf("%d", &matrix[i][j]);
14        }
15    }
16    printf("Sum of each row:\n");
17    for (int i = 0; i < rows; i++) {
18        int rowSum = 0;
19        for (int j = 0; j < columns; j++) {
20            rowSum += matrix[i][j];
21        }
22        printf("Row %d: %d\n", i + 1, rowSum);
23    }
24
25    printf("Sum of each column:\n");
26    for (int j = 0; j < columns; j++) {
27        int colSum = 0;
28        for (int i = 0; i < rows; i++) {
29            colSum += matrix[i][j];
30        }
31        printf("Column %d: %d\n", j + 1, colSum);
32    }
33
34    return 0;
35 }
```

```

PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\"
}
Enter the number of rows: 3
Enter the number of columns: 2
Enter the elements of the matrix:
2
3
5
7
8
3
Sum of each row:
Row 1: 5
Row 2: 12
Row 3: 11
Sum of each column:
Column 1: 15
Column 2: 13
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>

```

5. Find the product of two matrices using pointers.

```

C 5.c      x
C 5.c > main()
1 #include <stdio.h>
2
3 void multiplyMatrices(int *mat1, int *mat2, int *result, int r1, int c1, int c2) {
4     for (int i = 0; i < r1; i++) {
5         for (int j = 0; j < c2; j++) {
6             *(result + i * c2 + j) = 0;
7             for (int k = 0; k < c1; k++) {
8                 *(result + i * c2 + j) += *(mat1 + i * c1 + k) * *(mat2 + k * c2 + j);
9             }
10        }
11    }
12 }
13 int main() {
14     int r1, c1, r2, c2;
15     printf("Enter rows and columns of first matrix: ");
16     scanf("%d %d", &r1, &c1);
17     printf("Enter rows and columns of second matrix: ");
18     scanf("%d %d", &r2, &c2);
19
20     if (c1 != r2) {
21         printf("Matrix multiplication not possible. Columns of first matrix must match rows of second matrix.\n");
22         return 1;
23     }
24
25     int mat1[r1][c1], mat2[r2][c2], result[r1][c2];
26     printf("Enter elements of first matrix (%dx%d):\n", r1, c1);
27     for (int i = 0; i < r1; i++) {
28         for (int j = 0; j < c1; j++) {
29             scanf("%d", &mat1[i][j]);
30         }
31     }
32     printf("Enter elements of second matrix (%dx%d):\n", r2, c2);
33     for (int i = 0; i < r2; i++) {
34         for (int j = 0; j < c2; j++) {
35             scanf("%d", &mat2[i][j]);
36         }
37     }
38     multiplyMatrices(&mat1[0][0], &mat2[0][0], &result[0][0], r1, c1, c2);
39     printf("Product of the matrices:\n");
40     for (int i = 0; i < r1; i++) {
41         for (int j = 0; j < c2; j++) {
42             printf("%d ", result[i][j]);
43         }
44     }
45 }

```

```
    multiplyMatrices(&mat1[0][0], &mat2[0][0], &result[0][0], r1, c1, c2);
    printf("Product of the matrices:\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
}

return 0;
}
```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c"
}
Enter rows and columns of first matrix: 2
3
Enter rows and columns of second matrix: 3
2
Enter elements of first matrix (2x3):
2
4
6
4
7
8
Enter elements of second matrix (3x2):
6
4
12
9
6
5
Product of the matrices:
96 74
156 119
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

6. Store 'n' numbers (integers or real) in an array. Conduct a linear search for a given number and report success or failure in the form of a suitable message.

The screenshot shows a code editor window with a dark theme. The file is named '6.c'. The code implements a linear search algorithm. It first asks for the number of elements and then reads them into an array. It then asks for a search key and performs a linear search through the array to find it. If found, it prints the position; if not found, it prints a message indicating the key was not found.

```
C 6.c      X
C 6.c > ⚏ main()
1   #include <stdio.h>
2
3   int main() {
4       int n, key, found = 0;
5       printf("Enter the number of elements: ");
6       scanf("%d", &n);
7
8       int arr[n];
9       printf("Enter %d numbers:\n", n);
10      for (int i = 0; i < n; i++) {
11          scanf("%d", &arr[i]);
12      }
13      printf("Enter the number to search: ");
14      scanf("%d", &key);
15      for (int i = 0; i < n; i++) {
16          if (arr[i] == key) {
17              found = 1;
18              printf("Number %d found at position %d.\n", key, i + 1);
19              break;
20          }
21      }
22      if (!found) {
23          printf("Number %d not found.\n", key);
24      }
25      return 0;
26 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\Users\tila
}
Enter the number of elements: 4
Enter 4 numbers:
12
32
11
45
Enter the number to search: 45
Number 45 found at position 4.
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> □
```

7. Write a program to reverse an array.

```
C 7.c      X
C 7.c > ⚡ main()
1 #include <stdio.h>
2
3 int main() {
4     int n;
5     printf("Enter the number of elements: ");
6     scanf("%d", &n);
7
8     int arr[n];
9     printf("Enter the elements:\n");
10    for (int i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13    printf("Reversed array:\n");
14    for (int i = n - 1; i >= 0; i--) {
15        printf("%d ", arr[i]);
16    }
17    printf("\n");
18
19    return 0;
20 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd
}
Enter the number of elements: 4
Enter the elements:
1
13
2
45
Reversed array:
45 2 13 1
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

8. Find the largest three distinct elements in an array: Input:

arr[] = {10, 4, 3, 50, 23, 90} Output: 90, 50, 23

The screenshot shows a terminal window with the following content:

```
C 8.c x
C 8.c > main()
1 #include <stdio.h>
2
3 void findLargestThree(int arr[], int n) {
4     int first, second, third;
5
6     if (n < 3) {
7         printf("Array should have at least 3 elements.\n");
8         return;
9     }
10    first = second = third = -1;
11    for (int i = 0; i < n; i++) {
12        if (arr[i] > first) {
13            third = second;
14            second = first;
15            first = arr[i];
16        } else if (arr[i] > second && arr[i] < first) {
17            third = second;
18            second = arr[i];
19        } else if (arr[i] > third && arr[i] < second) {
20            third = arr[i];
21        }
22    }
23    printf("The largest three distinct elements are: %d, %d, %d\n", first, second, third);
24 }
25
26 int main() {
27     int arr[] = {10, 4, 3, 50, 23, 90};
28     int n = sizeof(arr) / sizeof(arr[0]);
29
30     findLargestThree(arr, n);
31     return 0;
32 }
```

Below the code, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS.

```
}
```

The largest three distinct elements are: 90, 50, 23

PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>

9. Move all zeroes to end of array

```
C 9.c      X
C 9.c > ...
1 #include <stdio.h>
2 void moveZeroesToEnd(int arr[], int n) {
3     int count = 0;
4     for (int i = 0; i < n; i++) {
5         if (arr[i] != 0) {
6             arr[count++] = arr[i];
7         }
8     }
9     while (count < n) {
10         arr[count++] = 0;
11     }
12 }
13 int main() {
14     int arr[] = {10, 0, 5, 0, 8, 0, 12};
15     int n = sizeof(arr) / sizeof(arr[0]);
16
17     moveZeroesToEnd(arr, n);
18     printf("Array after moving zeroes to the end:\n");
19     for (int i = 0; i < n; i++) {
20         printf("%d ", arr[i]);
21     }
22     printf("\n");
23
24     return 0;
25 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\users\tilak\onedrive\desktop\documents\c codes\problems"
}
Array after moving zeroes to the end:
10 5 8 12 0 0 0
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

10. Rearrange an array in maximum minimum form using Two Pointer Technique. Input: arr[] = {1, 2, 3, 4, 5, 6, 7} Output: arr[] = {7, 1, 6, 2, 5, 3, 4}.

```
C 10.c      X
C 10.c > ...
1 #include <stdio.h>
2 void rearrangeArray(int arr[], int n) {
3     int result[n];
4     int left = 0, right = n - 1, index = 0;
5
6     while (left <= right) {
7         if (index % 2 == 0) {
8             result[index] = arr[right];
9             right--;
10        } else {
11            result[index] = arr[left];
12            left++;
13        }
14        index++;
15    }
16    for (int i = 0; i < n; i++) {
17        arr[i] = result[i];
18    }
19 }
20
21 int main() {
22     int arr[] = {1, 2, 3, 4, 5, 6, 7};
23     int n = sizeof(arr) / sizeof(arr[0]);
24     for (int i = 0; i < n - 1; i++) {
25         for (int j = 0; j < n - i - 1; j++) {
26             if (arr[j] > arr[j + 1]) {
27                 int temp = arr[j];
28                 arr[j] = arr[j + 1];
29                 arr[j + 1] = temp;
30             }
31         }
32     }
33
34     rearrangeArray(arr, n);
35     printf("Rearranged array in maximum-minimum form:\n");
36     for (int i = 0; i < n; i++) {
37         printf("%d ", arr[i]);
38     }
39     printf("\n");
40
41     return 0;
42 }
```

```
7 1 6 2 5 3 4
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

11. Print all Distinct (Unique) Elements in given Array:

Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45} Output: 12, 10, 9, 2

```
C 11.c ×
C 11.c > ...
1 #include <stdio.h>
2
3 void printDistinctElements(int arr[], int n) {
4     int count;
5     for (int i = 0; i < n; i++) {
6         count = 0;
7         for (int j = 0; j < n; j++) {
8             if (arr[i] == arr[j]) {
9                 count++;
10            }
11        }
12        if (count == 1) {
13            printf("%d ", arr[i]);
14        }
15    }
16 }
17
18 int main() {
19     int arr[] = {12, 10, 9, 45, 2, 10, 10, 45};
20     int n = sizeof(arr) / sizeof(arr[0]);
21     printf("Distinct elements in the array are:\n");
22     printDistinctElements(arr, n);
23
24     return 0;
25 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "11"
Distinct elements in the array are:
12 9 2
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

12. Write a program to count the total number of nonzero elements in a two-dimensional array.

The screenshot shows a code editor window with the following details:

- Title Bar:** C 12.c
- Code Content:**

```
C 12.c > ⚙ countNonZeroElements(int arr[3][3], int rows, int cols)
1 #include <stdio.h>
2
3 int countNonZeroElements(int arr[3][3], int rows, int cols) {
4     int count = 0;
5     for (int i = 0; i < rows; i++) {
6         for (int j = 0; j < cols; j++) {
7             if (arr[i][j] != 0) {
8                 count++;
9             }
10        }
11    }
12    return count;
13 }
14 int main() {
15     int arr[3][3] = {
16         {1, 0, 3},
17         {0, 2, 0},
18         {4, 0, 5}
19     };
20     int rows = 3, cols = 3;
21
22     int result = countNonZeroElements(arr, rows, cols);
23
24     printf("Total number of non-zero elements: %d\n", result);
25 }
```
- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS
- Terminal Output:**

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems"
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> 12
Total number of non-zero elements: 5
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

13. Write a program using pointers to interchange the second biggest and the second smallest number in the array.

```
c 13.c  x
C 13.c > ⌂ findAndSwap(int [], int)
1 #include <stdio.h>
2
3 void swap(int *a, int *b) {
4     int temp = *a;
5     *a = *b;
6     *b = temp;
7 }
8
9 void findAndSwap(int arr[], int n) {
10    if (n < 4) {
11        printf("Array should have at least 4 elements to swap second smallest and second largest.\n");
12        return;
13    }
14    int smallest = -1, secondSmallest = -1;
15    int largest = -1, secondLargest = -1;
16
17    for (int i = 0; i < n; i++) {
18        if (smallest == -1 || arr[i] < arr[smallest]) {
19            secondSmallest = smallest;
20            smallest = i;
21        } else if (secondSmallest == -1 || arr[i] < arr[secondSmallest]) {
22            secondSmallest = i;
23        }
24
25        if (largest == -1 || arr[i] > arr[largest]) {
26            secondLargest = largest;
27            largest = i;
28        } else if (secondLargest == -1 || arr[i] > arr[secondLargest]) {
29            secondLargest = i;
30        }
31    }
32    swap(&arr[secondSmallest], &arr[secondLargest]);
33 }
34
35 int main() {
36     int arr[] = {10, 20, 4, 45, 99, 3, 25};
37     int n = sizeof(arr) / sizeof(arr[0]);
38
39     printf("Original array:\n");
40     for (int i = 0; i < n; i++) {
41         printf("%d ", arr[i]);
42     }
}
```

```
39     printf("Original array:\n");
40     for (int i = 0; i < n; i++) {
41         printf("%d ", arr[i]);
42     }
43     printf("\n");
44     findAndSwap(arr, n);
45     printf("Array after swapping second smallest and second largest:\n");
46     for (int i = 0; i < n; i++) {
47         printf("%d ", arr[i]);
48     }
49     printf("\n");
50
51     return 0;
52 }
53
```

```
\13 }
Original array:
10 20 4 45 99 3 25
Array after swapping second smallest and second largest:
10 20 4 45 99 3 25
Array after swapping second smallest and second largest:
10 20 45 4 99 3 25
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

● LAB-2

1. FIND NUMBER OF ROTATIONS IN A CIRCULARLY SORTED ARRAY

The screenshot shows a code editor window with a tab labeled "Main.c". The code implements a solution to find the second largest and second smallest elements in a circularly sorted array. It includes functions for swapping, finding the second largest and second smallest, printing the array, and searching for an element. The code uses standard C libraries like stdio.h and stdlib.h.

```
1 #include <stdio.h>
2 void swap(int *a, int *b) {
3     int temp = *a;
4     *a = *b;
5     *b = temp;
6 }
7 void findSecondLargestAndSecondSmallest(int arr[], int n) {
8     if (n < 4) {
9         printf("Array must have at least four elements.\n");
10        return;
11    }
12    int *secondLargest = NULL, *secondSmallest = NULL;
13    int largest = arr[0], smallest = arr[0];
14    for (int i = 1; i < n; i++) {
15        if (arr[i] > largest) {
16            secondLargest = &largest;
17            largest = arr[i];
18        } else if (arr[i] < smallest) {
19            secondSmallest = &smallest;
20            smallest = arr[i];
21        }
22    }
23    for (int i = 0; i < n; i++) {
24        if (arr[i] < largest && (secondLargest == NULL || arr[i] > *secondLargest)) {
25            secondLargest = &arr[i];
26        }
27        if (arr[i] > smallest && (secondSmallest == NULL || arr[i] < *secondSmallest)) {
28            secondSmallest = &arr[i];
29        }
30    }
31    swap(secondLargest, secondSmallest);
32 }
33 void printArray(int arr[], int n) {
34     for (int i = 0; i < n; i++) {
35         printf("%d ", arr[i]);
36     }
37     printf("\n");
38 }
```

2. SEARCH AN ELEMENT IN A CIRCULARLY SORTED ARRAY

The screenshot shows a code editor window with a tab labeled "Main.c". The code implements a search algorithm for a circularly sorted array. It defines a function `searchInCircularArray` that performs a modified binary search. The main function initializes an array and prints its elements. The search function handles cases where the target element is found in the middle or at the boundaries of the current search range.

```
1 #include <stdio.h>
2 int searchInCircularArray(int arr[], int n, int target) {
3     int left = 0;
4     int right = n - 1;
5     while (left <= right) {
6         int mid = left + (right - left) / 2;
7         if (arr[mid] == target) {
8             return mid;
8         }
9         if (arr[left] <= arr[mid]) {
10             if (target >= arr[left] && target <= arr[mid]) {
11                 right = mid - 1;
12             } else {
13                 left = mid + 1;
14             }
15         } else {
16             if (target > arr[mid] && target <= arr[right]) {
17                 left = mid + 1;
18             } else {
19                 right = mid - 1;
20             }
21         }
22     }
23     return -1;
24 }
25
26
27
28
29 int main() {
30     int arr[] = {8, 9, 10, 1, 2, 3, 4, 5, 6, 7};
31     int n = sizeof(arr) / sizeof(arr[0]);
32
33     printf("Array: ");
34     for (int i = 0; i < n; i++) {
35         printf("%d ", arr[i]);
36     }
37     printf("\n");
38     int test_values[] = {3, 8, 10, 6, 11};
```

3. Find the first or last occurrence of a given number in a sorted array

The screenshot shows a C IDE interface with the following details:

- Left Panel (Code Editor):** The file is named "Main.c". It contains two functions: `findFirstOccur` and `findLastOccur`. Both functions use binary search on a sorted array to find the first or last occurrence of a target value `x`.
- Top Bar:** Shows the file name "42z5zqdx3" and various tool buttons.
- Right Panel (Output):**
 - STDIN:** Shows the input "3" which is used as the target value `x`.
 - Output:** Displays the results of the program execution:
"First occurrence of 3 is at index 3"
"Last occurrence of 3 is at index 5"

4. Count occurrences of a given number in a sorted array

The screenshot shows a C IDE interface with the following details:

- Left Panel (Code Editor):** The file is named "Main.c". It contains two functions: `findFirstOccur` and `findLastOccur`. These functions are identical to the ones in the previous screenshot, but they are now part of a larger program that also includes a counting function.
- Top Bar:** Shows the file name "42z5zqdx3" and various tool buttons.
- Right Panel (Output):**
 - STDIN:** Shows the input "3" which is used as the target value `x`.
 - Output:** Displays the result of the program execution: "The number of occurrences of 3 in the array is: 3"

5. Find the smallest missing element from a sorted array

The screenshot shows the OneCompiler IDE interface. The left pane displays the C code for 'Main.c'. The right pane shows the execution results: 'STDIN' is empty, 'Input for the program (Optional)' contains '2', and 'Output' shows the result 'The missing smallest element is: 2'.

```
1 #include <stdio.h>
2 int MissingSmallestElement(int arr[], int low, int high)
3 {
4     if (low > high)
5     {
6         return low; // Return Low as it represents the smallest missing element
7     }
8     int mid = low + (high - low) / 2;
9     if (arr[mid] == mid)
10    {
11        return MissingSmallestElement(arr, mid + 1, high);
12    }
13    else
14    {
15        return MissingSmallestElement(arr, low, mid - 1);
16    }
17 }
18
19 int main()
20 {
21     int arr[] = {0, 1, 3, 4, 5, 6, 7, 9};
22     int n = sizeof(arr) / sizeof(arr[0]);
23     int missingElement = MissingSmallestElement(arr, 0, n - 1);
24     printf("The missing smallest element is: %d\n", missingElement);
25     return 0;
26 }
```

6. Find floor and ceil of a number in a sorted integer array

The screenshot shows the OneCompiler IDE interface. The left pane displays the C code for 'Main.c'. The right pane shows the execution results: 'STDIN' is empty, 'Input for the program (Optional)' contains '4', and 'Output' shows 'Floor of 4 is 3' and 'Ceil of 4 is 3'.

```
1 #include <stdio.h>
2 int findFloor(int arr[], int n, int x)
3 {
4     int low = 0, high = n - 1;
5     while (low <= high)
6     {
7         int mid = low + (high - low) / 2;
8         if (arr[mid] == x)
9             return mid;
10        if (arr[mid] < x)
11            low = mid + 1;
12        else
13            high = mid - 1;
14    }
15    return high;
16 }
17 int findCeil(int arr[], int n, int x) {
18     int low = 0, high = n - 1;
19
20     while (low <= high)
21     {
22         int mid = low + (high - low) / 2;
23         if (arr[mid] == x)
24             return mid;
25         if (arr[mid] < x)
26             high = mid - 1;
27         else
28             low = mid + 1;
29    }
30    return low;
31 }
32 int main()
33 {
34     int arr[] = {1, 2, 3, 4, 5, 6, 7};
35     int n = sizeof(arr) / sizeof(arr[0]);
36     int x = 4;
37     int floor = findFloor(arr, n, x);
38     int ceil = findCeil(arr, n, x);
39     printf("Floor of %d is %d\n", x, floor);
40 }
```

7. Search in a nearly sorted array in logarithmic time

The screenshot shows a code editor interface with a light gray background. On the left is the code editor pane containing a C program named 'main.c'. The code implements a search algorithm for a nearly sorted array. It includes a main loop that prints the array, reads a search key from standard input, and calls a search function. The search function uses a modified binary search on a nearly sorted array where elements are at most 2 positions off from their sorted position. The output pane on the right shows the execution results: it prints the nearly sorted array [2, 1, 3, 5, 4, 7, 6, 8, 9], prompts for a search element, finds 'Element 5 found at index 3', and concludes with 'Code Execution Successful'.

```
main.c
12     return mid + 1;
13     if (key < arr[mid])
14         high = mid - 2;
15     else
16         low = mid + 2;
17     }
18     return -1;
19 }
20 int main() {
21     int arr[] = {2, 1, 3, 5, 4, 7, 6, 8, 9};
22     int n = sizeof(arr) / sizeof(arr[0]);
23     int key;
24     printf("Nearly Sorted Array: ");
25     for(int i = 0; i < n; i++) {
26         printf("%d ", arr[i]);
27     }
28     printf("\nEnter element to search: ");
29     scanf("%d", &key);
30     int result = searchNearlySorted(arr, n, key
31         );
32     if (result != -1)
33         printf("Element %d found at index %d\n",
34             key, result);
35     else
36         printf("Element %d not found in the
```

Nearly Sorted Array: 2 1 3 5 4 7 6 8 9
Enter element to search: 5
Element 5 found at index 3
==== Code Execution Successful ===

8. Find the number of 1's in a sorted binary array

The screenshot shows a code editor interface with a light gray background. On the left is the code editor pane containing a C program named 'Main.c'. The program defines a function 'countOnes' that performs a binary search on a sorted binary array to find the count of 1's. It initializes 'low' to 0 and 'high' to n-1. It then enters a loop where it calculates 'mid' as the average of 'low' and 'high', compares the middle element with 1, and updates 'low' or 'high' based on the comparison. The output pane on the right shows the execution results: it prints 'Input for the program (Optional)' followed by 'Number of 1s: 4'.

```
Main.c
1 #include <stdio.h>
2 int countOnes(int arr[], int n)
3 {
4     int low = 0, high = n - 1;
5     while (low <= high) {
6         int mid = low + (high - low) / 2;
7         if (arr[mid] == 1)
8             high = mid - 1;
9         else
10            low = mid + 1;
11    }
12    return n - low;
13 }
14 int main() {
15     int arr[] = {0, 0, 1, 1, 1, 1};
16     int n = sizeof(arr) / sizeof(arr[0]);
17     int count = countOnes(arr, n);
18     printf("Number of 1s: %d\n", count);
19     return 0;
20 }
```

STDIN
Input for the program (Optional)
Output:
Number of 1s: 4

9. Find the peak element in an array

The screenshot shows an online compiler interface with the following details:

- File:** Main.c
- Code:**

```
1 #include <stdio.h>
2 int findPeakElement(int arr[], int n) {
3     // If array has only one element, it's the peak
4     if (n == 1)
5         return 0;
6     if (arr[0] >= arr[1])
7         return 0;
8     if (arr[n-1] >= arr[n-2])
9         return n-1;
10    int low = 1;
11    int high = n - 2;
12    while (low <= high) {
13        int mid = low + (high - low) / 2;
14        if (arr[mid] >= arr[mid-1] && arr[mid] >= arr[mid+1])
15            return mid;
16        if (arr[mid-1] > arr[mid])
17            high = mid - 1;
18        else
19            low = mid + 1;
20    }
21    return -1;
22 }
23
24 int main() {
25     int arr[] = {1, 3, 20, 4, 1, 0};
26     int n = sizeof(arr) / sizeof(arr[0]);
27     printf("Array: ");
28     for(int i = 0; i < n; i++) {
29         printf("%d ", arr[i]);
30     }
31     int result = findPeakElement(arr, n);
32     if (result != -1)
33         printf("\nPeak element is %d at index %d\n", arr[result], result);
34     else
35         printf("\nNo peak element found\n");
36     return 0;
37 }
```
- Output:**

STDIN
Input for the program (Optional)

Output:
Array: 1 3 20 4 1 0
Peak element is 20 at index 2

10. Find the missing term in a sequence in logarithmic time

The screenshot shows an online compiler interface with the following details:

- File:** Main.c
- Code:**

```
1 #include <stdio.h>
2
3 int findMissingTerm(int arr[], int n) {
4     int low = 0;
5     int high = n - 1;
6     int d = (arr[n-1] - arr[0]) / n; // Common difference of the A.P.
7
8     while (low <= high) {
9         int mid = low + (high - low) / 2;
10        int expected = arr[0] + mid * d; // Expected value in the sequence
11
12        if (arr[mid] == expected) {
13            low = mid + 1; // Missing term must be after mid
14        } else {
15            if (mid > 0 && arr[mid - 1] == arr[0] + (mid - 1) * d)
16                return expected; // Found the missing term
17            high = mid - 1; // Missing term must be before mid
18        }
19    }
20    return -1;
21 }
22
23
24 int main() {
25     int arr[] = {2, 4, 8, 10, 12, 14}; // Example array with a missing term
26     int n = sizeof(arr) / sizeof(arr[0]);
27
28     printf("Sequence: ");
29     for (int i = 0; i < n; i++) {
30         printf("%d ", arr[i]);
31     }
32
33     int result = findMissingTerm(arr, n);
34
35     if (result != -1)
36         printf("\nMissing term in the sequence is: %d", result);
37 }
```
- Output:**

Input for the program (Optional)

Output:
Sequence: 2 4 8 10 12 14
Missing term in the sequence is: 6

11. Find floor and ceil of a number in a sorted array(recursive solution)

```

main.c | Run | Output | Clear
1 #include <stdio.h>
2 int findFloor(int arr[], int low, int high, int x) {
3     if (low > high)
4         return -1;
5     if (x == arr[high])
6         return arr[high];
7     if (x < arr[low])
8         return -1;
9     int mid = low + (high - low) / 2;
10    if (arr[mid] == x)
11        return arr[mid];
12    if (mid > 0 && arr[mid-1] <= x && x < arr[mid])
13        return arr[mid-1];
14    if (x < arr[mid])
15        return findFloor(arr, low, mid-1, x);
16    return findFloor(arr, mid+1, high, x);
17 }
18 int findCeil(int arr[], int low, int high, int x) {
19     if (low > high)
20         return -1;
21     if (x <= arr[low])
22         return arr[low];

```

Sorted Array: 1 2 4 6 8 10 12 14
Enter number to find floor and ceil: 5.5
Floor of 5 is 4
Ceil doesn't exist
==== Code Execution Successful ====

12. Find frequency of each element in a sorted array containing duplicates

```

Main.c | Run | 42z637pxy
1 #include <stdio.h>
2 int findFirst(int arr[], int low, int high, int x) {
3     if (low > high)
4         return -1;
5     int mid = low + (high - low) / 2;
6     if ((mid == 0 || x > arr[mid-1]) && arr[mid] == x)
7         return mid;
8     else if (x > arr[mid])
9         return findFirst(arr, mid+1, high, x);
10    else
11        return findFirst(arr, low, mid-1, x);
12 }
13 int findLast(int arr[], int low, int high, int x) {
14     if (low > high)
15         return -1;
16     int mid = low + (high - low) / 2;
17     if ((mid == high || x < arr[mid+1]) && arr[mid] == x)
18         return mid;
19     else if (x < arr[mid])
20         return findLast(arr, low, mid-1, x);
21     else
22         return findLast(arr, mid+1, high, x);
23 }
24 void findFrequency(int arr[], int n) {
25     int i = 0;
26     while (i < n) {
27         int first = findFirst(arr, 0, n-1, arr[i]);
28         int last = findLast(arr, 0, n-1, arr[i]);
29         if (first != -1) {
30             printf("Element %d occurs %d times\n",
31                   arr[i], last - first + 1);
32         } else {
33             i++;
34         }
35     }
36 }
37

```

STDIN
Input for the program (Optional)

Output:
Sorted Array: 1 1 2 2 2 3 4 4 4 5
Frequency Count:
Element 1 occurs 2 times
Element 2 occurs 4 times
Element 3 occurs 1 times
Element 4 occurs 3 times
Element 5 occurs 1 times

13. Find square root of a number using binary search

The screenshot shows a code editor with a C file named `main.c`. The code implements the binary search algorithm to find the square root of a number. It includes error handling for non-positive numbers and uses a precision of 10 decimal places. The output window shows the user entering the number 4 and specifying a precision of 2 decimal places. The program outputs the result as approximately 1.9960937500.

```

main.c | Run | Output | Clear
1 #include <stdio.h>
2 double findSquareRoot(int x, int precision) {
3     if (x == 0 || x == 1)
4         return x;
5     double low = 1;
6     double high = x;
7     double epsilon = 1.0;
8     for(int i = 0; i < precision; i++) {
9         epsilon /= 10;
10    }
11    double result = 0;
12    while (high - low > epsilon) {
13        double mid = (low + high) / 2;
14        double sqr = mid * mid;
15        if (sqr == x) {
16            return mid;
17        }
18        else if (sqr < x) {
19            low = mid;
20            result = mid;
21        }
22        else {
23            high = mid;
24        }
25    }
26
27    return result;

```

Output:

```

Enter number to find square root: 4
Enter precision (number of decimal places): 2
Square root of 4 is approximately 1.9960937500
==== Code Execution Successful ====

```

14. Division of two numbers using binary search algorithm

The screenshot shows a code editor with a C file named `main.c`. The code performs integer division using a binary search approach. It handles cases where either or both numbers are negative. The user enters a dividend of 128 and a divisor of 2, resulting in a quotient of 64. The output window also displays a success message.

```

main.c | Run | Output | Clear
1 #include <stdio.h>
2 #include <stdlib.h>
3 double divide(int dividend, int divisor, int
precision) {
4     if (divisor == 0) {
5         printf("Error: Division by zero\n");
6         exit(1);
7     }
8     int sign = ((dividend < 0) ^ (divisor < 0))
9     ? -1 : 1;
10    dividend = abs(dividend);
11    divisor = abs(divisor);
12    int quotient = 0;
13    int low = 0;
14    int high = dividend;
15    while (low <= high) {
16        int mid = low + (high - low) / 2;
17        if (mid * divisor == dividend) {
18            quotient = mid;
19            break;
20        }
21        else if (mid * divisor < dividend) {
22            quotient = mid;
23            low = mid + 1;
24        }
25        else {
26            high = mid - 1;
27        }
28    }
29    if (sign == -1)
30        quotient = -quotient;
31
32    return quotient;
33}

```

Output:

```

Enter dividend: 128
Enter divisor: 2
Enter precision (decimal places): 2
128 / 2 = 64.0000000000
==== Code Execution Successful ====

```

15. Find the odd occurring element in an array in logarithmic time

The screenshot shows a code editor interface with a tab labeled "main.c". The code implements a binary search algorithm to find the first occurrence of an element that appears an odd number of times in an array. The code includes a main function that initializes an array with values {1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 7} and calls the helper function `findOddOccurrence`.

```
1 #include <stdio.h>
2 int findOddOccurrence(int arr[], int n) {
3     int low = 0;
4     int high = n - 1;
5     while (low <= high) {
6         if (low == high)
7             return arr[low];
8
9         int mid = low + (high - low) / 2;
10        if (mid % 2 == 0) {
11            if (arr[mid] == arr[mid + 1])
12                low = mid + 2;
13            else
14                high = mid;
15        }
16        else {
17            if (arr[mid] == arr[mid - 1])
18                low = mid + 1;
19            else
20                high = mid - 1;
21        }
22    }
23    return -1;
24 }
25 int main() {
26     int arr[] = {1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6,
27 , 7, 7};
```

The output window displays the array and the result of the search.

Array: 1 1 2 2 3 3 4 4 5 5 6 7 7
Element occurring odd number of times: 6
==== Code Execution Successful ====

16. Find pairs with difference 'k' in an array | constant space solution

The screenshot shows a code editor interface with a tab labeled "main.c". The code uses a two-pointer approach to find pairs with a specific difference k in an array. It first sorts the array and then iterates through it, using two pointers i and j to find pairs where $|arr[j] - arr[i]| = k$. The code includes a main function that initializes an array with values {1, 5, 2, 2, 2, 5, 5, 4} and sets the size and difference k .

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int compare(const void* a, const void* b) {
4     return *(int*)a - *(int*)b;
5 }
6 void findPairsWithDiffK(int arr[], int n, int k)
7 {
8     qsort(arr, n, sizeof(int), compare);
9     int i = 0;
10    int j = 1;
11    while (i < n && j < n) {
12        if (i != j && arr[j] - arr[i] == k) {
13            printf("Pair found: (%d, %d)\n",
14                  arr[i], arr[j]);
15            i++;
16            j++;
17        }
18        else if (arr[j] - arr[i] < k)
19            j++;
20        else
21            i++;
22    }
23    int main() {
24        int arr[] = {1, 5, 2, 2, 2, 5, 5, 4};
25        int n = sizeof(arr) / sizeof(arr[0]);
26        int k;
```

The output window displays the array, the difference k , the pairs found, and a success message.

Array: 1 5 2 2 2 5 5 4
Enter difference k: 3

Pairs with difference 3:
Pair found: (1, 4)
Pair found: (2, 5)
Pair found: (2, 5)
Pair found: (2, 5)

==== Code Execution Successful ====

17. Find 'k' closest elements to a given value in an array

The screenshot shows a code editor interface with a tab labeled "main.c". The code is a C program that finds the k closest elements to a given value x in an array. It includes includes for stdio.h, stdlib.h, and math.h. The main logic is implemented in two functions: findCrossOver and printKClosest. The findCrossOver function uses a binary search-like approach to find the index of the element closest to x. The printKClosest function then prints the k closest elements around this index. The output panel shows the array [12, 16, 22, 30, 35, 39, 42, 45, 48, 50, 53, 55, 56], the target value x = 40, the number of closest elements k = 5, and the resulting output: "5 closest elements to 40 are: 39 42 35 45 48". A message at the bottom indicates "Code Execution Successful".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 int findCrossOver(int arr[], int n, int x) {
5     int low = 0;
6     int high = n - 1;
7     if (x >= arr[high])
8         return high;
9     if (x < arr[low])
10        return low;
11    while (low <= high) {
12        int mid = low + (high - low) / 2;
13        if (arr[mid] <= x && arr[mid + 1] > x)
14            return mid;
15        else if (arr[mid] < x)
16            low = mid + 1;
17        else
18            high = mid - 1;
19    }
20    return -1;
21 }
22 void printKClosest(int arr[], int n, int k, int x) {
23     int left = findCrossOver(arr, n, x);
24     int right = left + 1;
25     if (arr[left] == x)
26         left--;
```

Output

Array: 12 16 22 30 35 39 42 45 48 50 53 55 56
Enter value x: 40
Enter number of closest elements (k): 5
5 closest elements to 40 are: 39 42 35 45 48
---- Code Execution Successful ----

● LAB-3

1. Implement the activity selection problem to get a clear understanding of greedy approach.

The screenshot shows a code editor interface with a C file named "Main.c". The code implements a recursive algorithm to solve the Activity Selection Problem. The code includes a struct for activities, a recursive function "RECURSIVE_ACTIVITY_SELECTOR", and a main function that initializes an array of activities and prints the maximum number of non-overlapping activities. The code is annotated with line numbers from 1 to 29. The code editor has tabs for "NEW", "C", and "RUN". The output window shows the result of running the program with input "2", which outputs "Maximum number of activities: 2".

```
1 #include <stdio.h>
2 struct Activity
3 {
4     int start, finish;
5 };
6 int RECURSIVE_ACTIVITY_SELECTOR(struct Activity activities[], int s, int f, int n)
7 {
8     int m = s + 1;
9     while (m <= n && activities[m].start < activities[s].finish)
10    {
11        m++;
12    }
13    if (m <= n)
14    {
15        return 1 + RECURSIVE_ACTIVITY_SELECTOR(activities, m, f, n);
16    }
17    else
18    {
19        return 0;
20    }
21 }
22
23 int main()
24 {
25     struct Activity activities[] = {{1, 4}, {3, 5}, {0, 6}, {5, 7}, {3, 8}, {5, 9}, {8, 11}};
26     int n = sizeof(activities) / sizeof(activities[0]);
27     int maxActivities = RECURSIVE_ACTIVITY_SELECTOR(activities, 0, n - 1, n);
28     printf("Maximum number of activities: %d\n", maxActivities);
29 }
```

Output:
Maximum number of activities: 2

2. Get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on time requirements for matrix multiplication.

```
main.c
```

  

Run

Clear

```
1 #include <stdio.h>
2 #include <limits.h>
3 ~ int matrixChainOrder(int p[], int n) {
4     int m[n][n];
5     int s[n][n];
6     for (int i = 1; i < n; i++)
7         m[i][i] = 0;
8 ~     for (int L = 2; L < n; L++) {
9 ~         for (int i = 1; i < n - L + 1; i++) {
10            int j = i + L - 1;
11            m[i][j] = INT_MAX;
12 ~         for (int k = i; k <= j - 1; k++) {
13             int q = m[i][k] + m[k + 1][j] +
14                 p[i - 1] * p[k] * p[j];
15             if (q < m[i][j]) {
16                 m[i][j] = q;
17                 s[i][j] = k;
18             }
19         }
20     }
21     return m[1][n - 1];
22 }
23 ~ void printOptimalParenthesis(int s[][10], int i,
24     int j) {
25     if (i == j)
26         printf("A%d", i);
27     else
28         printOptimalParenthesis(s, i, s[i][j], j);
29 }
```

Output

```
Minimum number of multiplications is 18
Optimal Parenthesization is: ((((((((A*B*C*D)*E)*F)*G)*H)*I)*J)

===== Code Exited With Errors =====
```

3. Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4 #include <time.h>
5
6 #define V 6 // Number of vertices
7 #define INF INT_MAX
8
9 // Structure to represent an edge
10 struct Edge {
11     int src, dest, weight;
12 };
13
14 // Function to initialize the distance array
15 void initializeDistances(int dist[], int src) {
16     for (int i = 0; i < V; i++) {
17         dist[i] = INF;
18         dist[src] = 0;
19     }
20 }
21 // Function to find the vertex with minimum distance
22 int minDistance(int dist[], int visited[]) {
23     int min = INF, min_index;
24     for (int v = 0; v < V; v++) {
25         if (!visited[v] && dist[v] <= min) {
26             min = dist[v];
27             min_index = v;
28         }
29     }
30     return min_index;
31 }
32 // Dijkstra's Algorithm
33 void dijkstra(int graph[V][V], int src, int dist[]) {
34     int visited[V];
35 }
```

STDIN
Input for the program (Optional)

Output:

Dijkstra's Algorithm Results:
 Vertex Distance from Source
 0 0
 1 4
 2 12
 3 19
 4 26
 5 16
 Time taken: 0.000002 seconds

Bellman-Ford Algorithm Results:
 Vertex Distance from Source
 0 0
 1 4
 2 12
 3 19
 4 26
 5 16
 Time taken: 0.000001 seconds

4. Through 0/1 Knapsack problem, analyze the greedy and dynamic programming approach for the same dataset.

```

1 #include <stdio.h>
2 typedef struct {
3     int weight;
4     int value;
5     float ratio;
6 } Item;
7 int compare(const void *a, const void *b) {
8     Item *item1 = (Item *)a;
9     Item *item2 = (Item *)b;
10    return (item2->ratio > item1->ratio) - (item2->ratio < item1->ratio);
11 }
12 float greedyKnapsack(Item items[], int n, int capacity) {
13     qsort(items, n, sizeof(Item), compare);
14     float totalValue = 0.0;
15     for (int i = 0; i < n; i++) {
16         if (items[i].weight <= capacity) {
17             capacity -= items[i].weight;
18             totalValue += items[i].value;
19         } else {
20             totalValue += items[i].value * ((float)capacity / items[i].weight);
21             break;
22         }
23     }
24     return totalValue;
25 }
26 int main() {
27     Item items[] = {{10, 60}, {20, 100}, {30, 120}};
28     int n = sizeof(items) / sizeof(items[0]);
29     int capacity = 50;
30
31     float maxValue = greedyKnapsack(items, n, capacity);
32     printf("Maximum value in Greedy Knapsack: %.2f\n", maxValue);
33     return 0;
34 }
```

STDIN
Input for the program (Optional)

Output:

Maximum value in Greedy Knapsack: 240.00

```

1 #include <stdio.h>
2 int knapsackDP(int capacity, int weights[], int values[], int n) {
3     int dp[n + 1][capacity + 1];
4     for (int i = 0; i <= n; i++) {
5         for (int w = 0; w <= capacity; w++) {
6             if (i == 0 || w == 0)
7                 dp[i][w] = 0;
8             else if (weights[i - 1] <= w)
9                 dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]]);
10                values[i - 1] + dp[i - 1][w - weights[i - 1]];
11            else
12                dp[i][w] = dp[i - 1][w];
13        }
14    }
15    return dp[n][capacity];
16 }
17 int main() {
18     int values[] = {60, 100, 120};
19     int weights[] = {10, 20, 30};
20     int capacity = 50;
21     int n = sizeof(values) / sizeof(values[0]);
22     int maxValue = knapsackDP(capacity, weights, values, n);
23     printf("Maximum value in Dynamic Programming Knapsack: %d\n", maxValue);
24     return 0;
25 }

```

STDIN

Input for the program (Optional)

Output:

Maximum value in Dynamic Programming Knapsack: 220

5. Implement the sum of subset and N Queen problem.

```

1 #include <stdio.h>
2 int isSubsetSum(int set[], int n, int sum) {
3     if (sum == 0)
4         return 1;
5     if (n == 0)
6         return 0;
7     if (set[n - 1] > sum)
8         return isSubsetSum(set, n - 1, sum);
10    return isSubsetSum(set, n - 1, sum) || isSubsetSum(set, n - 1, sum - set[n - 1]);
11 }
12 int main() {
13     int set[] = {3, 34, 4, 12, 5, 2};
14     int sum = 9;
15     int n = sizeof(set) / sizeof(set[0]);
16
17     if (isSubsetSum(set, n, sum))
18         printf("Found a subset with given sum\n");
19     else
20         printf("No subset with given sum\n");
21
22     return 0;
23 }
24

```

STDIN

Input for the program (Optional)

Output:

Found a subset with given sum

The screenshot shows a code editor with the file 'main.c' containing C code for solving the N-Queens problem. The code includes input validation, a check for trivial cases, and a recursive function to find a solution. It prints the board configuration if a solution exists. The output window shows the user entering '5' and the program outputting a 5x5 board where a solution is found. The board is represented by dots ('.') and the queen ('Q'). A message at the bottom indicates successful execution.

```

main.c
1 //include <stdio.h>
2 //include <stdlib.h>
3 //include <time.h>
4 //include <stdbool.h>
5
6 // Structure for Branch and Bound
7 typedef struct {
8     float bound;
9     float profit;
10    float weight;
11    int level;
12 } Node;
13
14 // Global variables to store best solution for
15 // backtracking
16 int* bestSolution;
17 int maxProfit = 0;
18
19 // Function to get maximum of two integers
20 int max(int a, int b) {
21     return (a > b) ? a : b;
22 }
23
24 int main() {
25     int N;
26     printf("Enter the size of the chessboard (N): ");
27     scanf("%d", &N);
28     if(N <= 0) {
29         printf("Invalid board size!\n");
30         return 1;
31     }
32     if(N == 2 || N == 3) {
33         printf("Solution does not exist for N = %d\n", N);
34         return 1;
35     }
36     int **board = initializeBoard(N);
37     if(solveNQueens(board, 0, N)) {
38         printf("\nSolution exists for N = %d", N);
39         printBoard(board, N);
40     } else {
41         printf("\nSolution does not exist for N = %d\n", N);
42     }
43     freeBoard(board, N);
44     return 0;
45 }

```

Output

Enter the size of the chessboard (N): 5

Solution exists for N = 5

Solution Board:

Q . . .
. . . Q .
. Q . .
. . . . Q
. . Q . .

==== Code Execution Successful =====

6. Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic programming approach.

The screenshot shows a code editor with the file 'main.c' for a 0/1 Knapsack problem. The code defines a structure for nodes, initializes global variables for best solution and maximum profit, and implements a recursive function to find the best solution using Branch and Bound. The user enters the number of items (34) and knapsack capacity (25). The program generates test data, compares different approaches (Dynamic Programming, Backtracking, Branch and Bound), and prints the results.

```

main.c
1 //include <stdio.h>
2 //include <stdlib.h>
3 //include <time.h>
4 //include <stdbool.h>
5
6 // Structure for Branch and Bound
7 typedef struct {
8     float bound;
9     float profit;
10    float weight;
11    int level;
12 } Node;
13
14 // Global variables to store best solution for
15 // backtracking
16 int* bestSolution;
17 int maxProfit = 0;
18
19 // Function to get maximum of two integers
20 int max(int a, int b) {
21     return (a > b) ? a : b;
22 }
23
24 int main() {
25     int n, W;
26     printf("Enter number of items: ");
27     scanf("%d", &n);
28     printf("Enter knapsack capacity: ");
29     scanf("%d", &W);
30
31     Generated Test Data:
32
33     Weights: 34 37 38 18 5 13 38 4 35 1 24 11 35 18 22 10 49 12
34     32 29 30 45 38 30 1 8 32 16 20 40 45 45 47 30
35     Values: 35 6 34 92 100 5 97 60 4 83 1 90 48 72 12 55 97 86
36     29 89 91 69 48 24 30 27 20 67 83 91 49 93 8 31
37
38     Comparing different approaches:
39
40     Time taken by Dynamic Programming: 0.000016 seconds
41     Time taken by Backtracking: 0.000039 seconds
42     Time taken by Branch and Bound: 0.000008 seconds
43
44     Results:
45     Dynamic Programming: 363
46     Backtracking: 363
47     Branch and Bound: 305
48 }

```

