

LAB-PROGRAM

1. Write a program to Print Fibonacci Series using recursion.

AIM:

To write a C program to print Fibonacci series using recursion.

Algorithm:

- Start.
- Read number of terms n .
- Define a recursive function to generate Fibonacci numbers.
- Use a loop to print Fibonacci numbers.
- Stop.

Code:

```
#include <stdio.h>

int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n, i;
    printf("Enter number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci Series:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```

Output:

```
main.py  [Icons]  Share  Run  Output
1- def fibonacci(n):
2-     if n <= 1:
3-         return n
4-     else:
5-         return fibonacci(n-1) + fibonacci(n-2)
6
7 n = 7
8 print("Fibonacci Series:")
9 for i in range(n):
10     print(fibonacci(i),end=" ")

Fibonacci Series:
0 1 1 2 3 5 8
=== Code Execution Successful ===
```

2. Write a program to check the given no is Armstrong or not.

AIM:

To write a C program to check if a number is Armstrong or not.

Algorithm:

- Start.
- Read a number.
- Count its digits.
- Compute sum of digits raised to the power of number of digits.
- Compare sum to original number.
- Print result.
- Stop.

Code:

```
#include <stdio.h>
#include <math.h>
int main() {
    int num, temp, digits = 0, sum = 0, rem;
    printf("Enter a number: ");
    scanf("%d", &num);
    temp = num;
    while (temp != 0) {
        digits++;
        temp /= 10;
    }
    temp = num;
    while (temp != 0) {
        rem = temp % 10;
        sum += pow(rem, digits);
        temp /= 10;
    }
```

```

    }
    if (sum == num)
        printf("Armstrong Number\n");
    else
        printf("Not an Armstrong Number\n");
    return 0;
}

```

Output:

main.py	Output
<pre> 1 num = 153 2 order = len(str(num)) 3 sum = 0 4 temp = num 5 6 while temp > 0: 7 digit = temp % 10 8 sum += digit ** order 9 temp //= 10 10 11 if num == sum: 12 print(num, "is an Armstrong number") 13 else: 14 print(num, "is not an Armstrong number") </pre>	<pre> 153 is an Armstrong number === Code Execution Successful === </pre>

3. Write a program to find the GCD of two numbers .

AIM:

To write a C program to find GCD of two numbers using recursion.

Algorithm:

- Start.
- Read two numbers.
- Use recursion to compute GCD.
- Print result.
- Stop.

Code:

```

#include <stdio.h>
int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
int main() {
    int x, y;
    printf("Enter two numbers: ");

```

```

scanf("%d %d", &x, &y);
printf("GCD is %d\n", gcd(x, y));
return 0;
}

```

Output:

The screenshot shows a code editor with a dark theme. On the left, a file named 'main.py' contains the following Python code:

```

1 def gcd(a, b):
2     if b == 0:
3         return a
4     else:
5         return gcd(b, a % b)
6
7 x = 34
8 y = 87
9 print("GCD =", gcd(x, y))

```

On the right, the 'Output' panel shows the result of running the code:

```

GCD = 1
=== Code Execution Successful ===

```

4. Write a program to get the largest element of an array.

AIM:

To find and print the largest element of an array.

Algorithm:

- Start.
- Read number of elements.
- Read array elements.
- Initialize max to first element.
- Loop to compare each element; update max as needed.
- Print max.
- Stop.

Code:

```

#include <stdio.h>

int main() {
    int n, i, max;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter array elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    max = arr[0];
    for (i = 1; i < n; i++)

```

```

        if (arr[i] > max)
            max = arr[i];
    printf("Largest element is %d\n", max);
    return 0;
}

```

Output:

main.py	Share	Run	Output
<pre> 1 arr = [4,7,2,0,6] 2 print("Largest element =",max(arr)) </pre>	Share code		<pre> Largest element = 7 === Code Execution Successful === </pre>

5. Write a program to find the Factorial of a number .

AIM:

To write a C program to find the factorial of a given number using recursion.

Algorithm:

- Start.
- Read a number n.
- Define a recursive function to calculate factorial.
- Print the result.
- Stop.

Code:

```

#include <stdio.h>

int factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    return n * factorial(n - 1);
}

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Factorial of %d is %d\n", n, factorial(n));
    return 0;
}

```

Output:

```
main.py [ ] [ ] [ ] Share Run Output Clear
```

```
1 - def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * factorial(n-1)  
6  
7 num = 98  
8 print("Factorial of", num, "=",factorial(num))
```

```
Factorial of 98 = 94268904488832477456261857430572424738096937640789516634942387772947  
0707002322379888297615920772911982360585058860846042941264756736000000000000000000  
0000  
  
=== Code Execution Successful ===
```

6. Write a program to check a number is a prime number or not .

AIM:

To write a C program to check if a number is prime.

Algorithm:

- Start.
- Read a number n .
- Check if n is less than or equal to 1.
- For each number i from 2 to $n/2$, check if n is divisible by i .
- Print if prime or not.
- Stop.

Code:

```
#include <stdio.h>

int main() {
    int n, i, flag = 0;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n <= 1) flag = 1;
    for (i = 2; i <= n / 2; i++) {
        if (n % i == 0) {
            flag = 1;
            break;
        }
    }
    if (flag == 0)
        printf("%d is a prime number\n", n);
    else
        printf("%d is not a prime number\n", n);
    return 0;
}
```

Output:

main.py	Output
<pre>1 n = 19 2 3 if n <= 1: 4 print(n, "is not a prime number") 5 else: 6 for i in range(2, int(n/2)+1): 7 if n % i == 0: 8 print(n, "is not a prime number") 9 break 10 else: 11 print(n, "is a prime number")</pre>	<pre>19 is a prime number === Code Execution Successful ===</pre>

7. Write a program to perform Selection sort.

AIM:

To write a C program to perform selection sort on an array.

Algorithm:

- Start.
- Read array size and elements.
- For each position in the array, find the minimum element in the unsorted portion and swap.
- Repeat until sorted.
- Print sorted array.
- Stop.

Code:

```
#include <stdio.h>

int main() {
    int n, i, j, min, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter array elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min])
                min = j;
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}
```

```

printf("Sorted array:\n");
for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
return 0;
}

```

Output:

main.py	Output
<pre> 1 # Simple Selection Sort in Python 2 3 def selection_sort(arr): 4 for i in range(len(arr)): 5 min_index = i 6 for j in range(i + 1, len(arr)): 7 if arr[j] < arr[min_index]: 8 min_index = j 9 arr[i], arr[min_index] = arr[min_index], arr[i] 10 11 return arr 12 13 # Example 14 numbers = [64, 25, 12, 22, 11] 15 print("Original list:", numbers) 16 17 sorted_list = selection_sort(numbers) 18 print("Sorted list:", sorted_list) </pre>	<pre> Original list: [64, 25, 12, 22, 11] Sorted list: [11, 12, 22, 25, 64] === Code Execution Successful === </pre>

8. Write a program to perform Bubble sort.

AIM:

To write a C program to perform bubble sort on an array.

Algorithm:

- Start.
- Read array size and elements.
- For each element, compare with the next and swap if needed.
- Repeat until sorted.
- Print sorted array.
- Stop.

Code:

```

#include <stdio.h>

int main() {
    int n, i, j, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter array elements:\n");
    for (i = 0; i < n; i++)

```



```

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

```

Output:

main.py	Output
<pre> 1 # Simple Bubble Sort in Python 2 3 def bubble_sort(arr): 4 n = len(arr) 5 6 for i in range(n): 7 for j in range(0, n - i - 1): 8 if arr[j] > arr[j + 1]: 9 # Swap elements 10 arr[j], arr[j + 1] = arr[j + 1], arr[j] 11 12 return arr 13 14 15 # Example 16 numbers = [64, 34, 25, 12, 22, 11, 90] 17 print("Original list:", numbers) 18 19 sorted_list = bubble_sort(numbers) 20 print("Sorted list:", sorted_list) </pre>	<pre> Original list: [64, 34, 25, 12, 22, 11, 90] Sorted list: [11, 12, 22, 25, 34, 64, 90] === Code Execution Successful === </pre>

9. Write a program for to multiply two Matrix.

Aim:

To multiply two matrices and display the resultant matrix.

Algorithm:

1. Start
2. Read number of rows and columns for the first matrix (r1, c1)
3. Read number of rows and columns for the second matrix (r2, c2)
4. Check if column of first matrix (c1) is equal to row of second matrix (r2). If not, multiplication is not possible.
5. Read elements of the first matrix and second matrix.
6. Initialize a result matrix with dimensions r1 x c2 with all elements as 0.

7. Multiply matrices using nested loops:
For i = 0 to r1-1
For j = 0 to c2-1
For k = 0 to c1-1
result[i][j] += matrix1[i][k] * matrix2[k][j]
8. Display the resultant matrix.
9. Stop

Code:

```
#include <stdio.h>

int main() {
    int r1, c1, r2, c2, i, j, k;
    int matrix1[10][10], matrix2[10][10], result[10][10];
    printf("Enter rows and columns for first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns for second matrix: ");
    scanf("%d %d", &r2, &c2);
    if (c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 0;
    }
    printf("Enter elements of first matrix:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    printf("Enter elements of second matrix:\n");
    for (i = 0; i < r2; i++) {
        for (j = 0; j < c2; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            result[i][j] = 0;
        }
    }
}
```

```

    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            for (k = 0; k < c1; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }

    printf("Resultant matrix after multiplication:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}

```

Output:

main.py	Run	Output
<pre> 1 # Program to multiply two matrices 2 3 # Matrix A 4 A = [5 [1, 2, 3], 6 [4, 5, 6] 7] 8 9 # Matrix B 10 B = [11 [7, 8], 12 [9, 10], 13 [11, 12] 14] 15 16 # Result matrix will have rows of A and columns of B 17 result = [[0 for _ in range(len(B[0]))] for _ in range(len(A))] 18 19 # Matrix multiplication 20 for i in range(len(A)): # rows of A 21 for j in range(len(B[0])): # columns of B 22 for k in range(len(B)): # rows of B 23 result[i][j] += A[i][k] * B[k][j] 24 25 # Print result 26 print("Result of Matrix Multiplication:") 27 for row in result: 28 print(row) </pre>	Run	<pre> Result of Matrix Multiplication: [58, 64] [139, 154] === Code Execution Successful === </pre>

10. Write a program for to check whether a given String is Palindrome or not.

Aim:

To check whether a given string is a palindrome or not.

Algorithm:

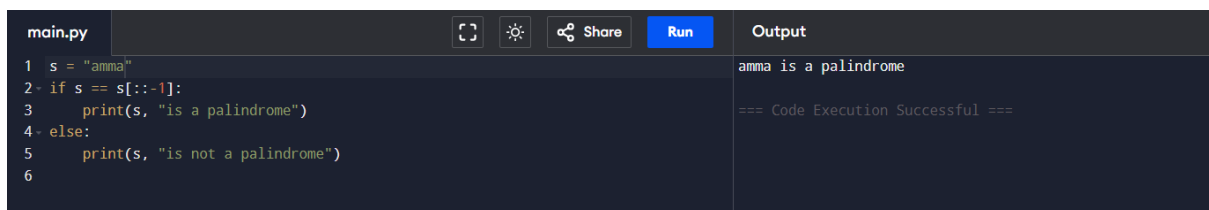
1. Start
2. Read the input string.
3. Initialize two indices, start=0 and end=length_of_string - 1.
4. Compare characters at start and end position.

5. If characters are equal, increment start and decrement end; repeat step 4 until start >= end.
6. If any character pair mismatches, the string is not a palindrome.
7. If all pairs match, string is a palindrome.
8. Print the result.
9. Stop

Code:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];
    int start = 0, end, flag = 1;
    printf("Enter a string: ");
    gets(str);
    end = strlen(str) - 1;
    while (start < end) {
        if (str[start] != str[end]) {
            flag = 0;
            break;
        }
        start++;
        end--;
    }
    if (flag)
        printf("The string is a palindrome.\n");
    else
        printf("The string is not a palindrome.\n");
    return 0;
}
```

Output:



```
main.py  [Run] [Share] [Settings] [Full Screen]
1 s = "amma"
2 if s == s[::-1]:
3     print(s, "is a palindrome")
4 else:
5     print(s, "is not a palindrome")
6

Output
amma is a palindrome
=== Code Execution Successful ===
```

10. Write a program for to copy one string to another.

Aim:

To copy one string into another string.

Algorithm:

1. Start
2. Take input string from the user.
3. Copy each character from source string to destination string until the null character is encountered.
4. Add the null character at the end of the destination string.
5. Print the copied string.
6. Stop.

Code:

```
#include <stdio.h>

int main() {
    char str1[100], str2[100];
    int i = 0;
    printf("Enter a string: ");
    fgets(str1, sizeof(str1), stdin);
    while (str1[i] != '\0') {
        str2[i] = str1[i];
        i++;
    }
    str2[i] = '\0';
    printf("Copied string: %s", str2);
    return 0;
}
```

Output:

main.py	Run	Output
<pre>1 # Program to copy one string to another 2 3 # Original string 4 str1 = "Hello Python" 5 6 # Copying string 7 str2 = str1 8 9 print("Original String:", str1) 10 print("Copied String:", str2)</pre>		<pre>Original String: Hello Python Copied String: Hello Python === Code Execution Successful ===</pre>

11. Write a Program to perform binary search.

Aim:

To perform binary search on a sorted array to find a target element.

Algorithm:

1. Start
2. Take sorted array input and target element.
3. Initialize low = 0, high = size-1.
4. Find mid = (low + high)/2.
5. If arr[mid] is the target, print index and stop.
6. If arr[mid] < target, set low = mid + 1, else high = mid - 1.
7. Repeat steps 4-6 until low > high.
8. If element not found, print not found message.
9. Stop.

Code:

```
#include <stdio.h>

int binarySearch(int arr[], int size, int target) {
    int low = 0, high = size - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 23;
    int result = binarySearch(arr, size, target);
    if (result != -1)
        printf("Element found at index %d\n", result);
    else
```

```

        printf("Element not found\n");
    return 0;
}

```

Output:

The screenshot shows a code editor with a file named 'main.py'. The code implements a binary search algorithm in Python. It defines a function 'binary_search(arr, target)' that takes an array and a target value. The function uses a while loop to find the target, returning the index if found or -1 if not. An example array [2, 4, 7, 10, 14, 20, 25] and target 10 are used. The output shows 'Element found at index 3' and '=== Code Execution Successful ==='.

```

main.py
1 # Binary Search Program in Python
2
3 def binary_search(arr, target):
4     low = 0
5     high = len(arr) - 1
6
7     while low <= high:
8         mid = (low + high) // 2
9
10        # Check if target is at mid
11        if arr[mid] == target:
12            return mid # return the index
13
14        # If target is smaller, ignore right half
15        elif arr[mid] > target:
16            high = mid - 1
17
18        # If target is larger, ignore left half
19        else:
20            low = mid + 1
21
22    return -1 # not found
23
24
25 # Example array (must be sorted)
26 numbers = [2, 4, 7, 10, 14, 20, 25]
27
28 # Target value to search
29 target = 10
30
31 result = binary_search(numbers, target)
32
33 if result != -1:
34     print("Element found at index {result}")
35 else:
36     print("Element not found")

```

Output

```

Element found at index 3
=== Code Execution Successful ===

```

12. Write a program to print the reverse of a string.

Aim:

To print the reverse of a given string.

Algorithm:

1. Start
2. Read the string.
3. Find length of the string.
4. Use a loop to print characters from the last to first.
5. Stop.

Code:

```

#include <stdio.h>
#include <string.h>

int main() {
    char str[100];
    int i, length;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    length = strlen(str);
    if(str[length-1] == '\n') {

```

```

        str[length-1] = '\0';
        length--;
    }
    printf("Reversed string: ");
    for (i = length - 1; i >= 0; i--) {
        printf("%c", str[i]);
    }
    printf("\n");
    return 0;
}

```

Output:

The screenshot shows a code editor with a file named 'main.py'. The code is a Python program that reverses the string 'PREMM' using slicing. The output panel on the right shows 'Reversed string: MMERP' and '=== Code Execution Successful ==='.

```

main.py
1 # Program to reverse a string
2
3 text = "PREMM"
4
5 # Reverse the string using slicing
6 reversed_text = text[::-1]
7
8 print("Reversed string:", reversed_text)
9
10

```

Reversed string: MMERP
 === Code Execution Successful ===

13. Write a program to find the length of a string.

AIM:

To find the length of a string without using library function.

ALGORITHM:

Read string

Initialize count = 0

Traverse until NULL character

Increment count

Display count

Code:

```

#include <stdio.h>

int main() {

    char s[100];

```



```

int len=0;

printf("Enter string: ");

gets(s);

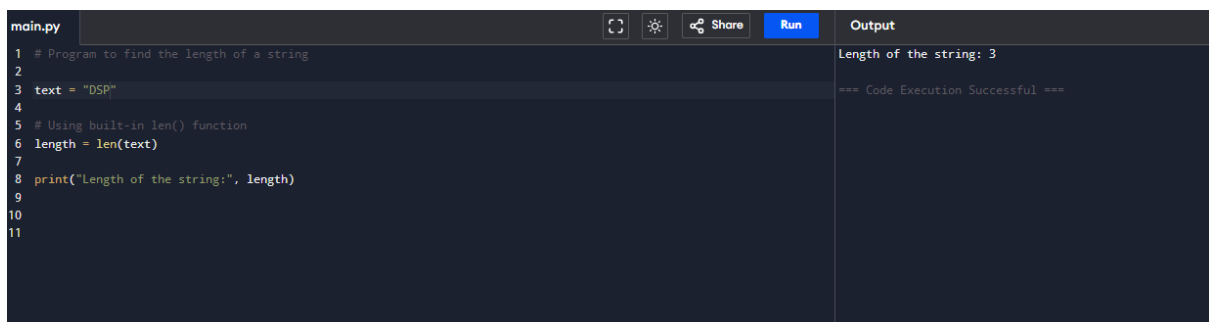
while(s[len] != '\0') len++;

printf("Length: %d", len);

}

```

Output:



```

main.py
1 # Program to find the length of a string
2
3 text = "DSP"
4
5 # Using built-in len() function
6 length = len(text)
7
8 print("Length of the string:", length)
9
10
11

```

Length of the string: 3

=== Code Execution Successful ===

14. Write a program to perform Strassen's Matrix Multiplication.

AIM:

To multiply two matrices using Strassen's technique.

ALGORITHM:

- Divide both matrices into sub-matrices
- Calculate 7 intermediate products
- Combine results to form resultant matrix

Code:

```

#include <stdio.h>

int main() {

    int A[2][2], B[2][2], C[2][2];

    int p1,p2,p3,p4,p5,p6,p7;

```

```

printf("Enter Matrix A: ");

for(int i=0;i<2;i++)

    for(int j=0;j<2;j++) scanf("%d",&A[i][j]);


printf("Enter Matrix B: ");

for(int i=0;i<2;i++)

    for(int j=0;j<2;j++) scanf("%d",&B[i][j]);


p1 = (A[0][0]+A[1][1])*(B[0][0]+B[1][1]);
p2 = (A[1][0]+A[1][1])*B[0][0];
p3 = A[0][0]*(B[0][1]-B[1][1]);
p4 = A[1][1]*(B[1][0]-B[0][0]);
p5 = (A[0][0]+A[0][1])*B[1][1];
p6 = (A[1][0]-A[0][0])*(B[0][0]+B[0][1]);
p7 = (A[0][1]-A[1][1])*(B[1][0]+B[1][1]);


C[0][0]=p1+p4-p5+p7;
C[0][1]=p3+p5;
C[1][0]=p2+p4;
C[1][1]=p1-p2+p3+p6;


printf("Result Matrix:\n");

for(int i=0;i<2;i++){

    for(int j=0;j<2;j++) printf("%d ",C[i][j]);

    printf("\n");

}

}

```

Output:

```
main.py 1 # Strassen's Matrix Multiplication for 2x2 matrices
2
3 def strassen(A, B):
4     # Extract elements for easier reference
5     a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
6     e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]
7
8     # Strassen's formula
9     p1 = a * (f - h)
10    p2 = (a + b) * h
11    p3 = (c + d) * e
12    p4 = d * (g - e)
13    p5 = (a + d) * (e + h)
14    p6 = (b - d) * (g + h)
15    p7 = (a - c) * (e + f)
16
17    # Compute final matrix values
18    c11 = p5 + p4 - p2 + p6
19    c12 = p1 + p2
20    c21 = p3 + p4
21    c22 = p1 + p5 - p3 - p7
22
23    # Return resulting matrix
24    return [[c11, c12],
25            [c21, c22]]
26
27
28 # Example matrices
29 A = [
30     [1, 2],
31     [3, 4]
32 ]
33
34 B = [
35     [5, 6],
36     [7, 8]
37 ]
```

Result of Strassen's Matrix Multiplication:
[[19, 22],
[43, 50]]
=== Code Execution Successful ===

15. Write a program to perform Merge Sort.

AIM:

To sort an array using Merge Sort algorithm.

ALGORITHM:

Divide array into halves

Sort each half recursively

Merge sorted halves

Code:

```
#include <stdio.h>

void merge(int a[], int l, int m, int r){
    int i=l,j=m+1,k=0,temp[100];

    while(i<=m && j<=r)
        temp[k++] = (a[i]<a[j])?a[i++]:a[j++];

    while(i<=m) temp[k++] = a[i++];
    while(j<=r) temp[k++] = a[j++];

    for(i=l,k=0;i<=r;i++) a[i]=temp[k++];
}
```

```

void mergesort(int a[], int l, int r){
    if(l<r){
        int m=(l+r)/2;
        mergesort(a,l,m);
        mergesort(a,m+1,r);
        merge(a,l,m,r);
    }
}

int main(){
    int a[10], n;
    scanf("%d",&n);
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    mergesort(a,0,n-1);
    for(int i=0;i<n;i++) printf("%d ",a[i]);
}

```

Output:

main.py	Output
<pre> 1 # Simple Bubble Sort in Python 2 3 def bubble_sort(arr): 4 n = len(arr) 5 6 for i in range(n): 7 for j in range(0, n - i - 1): 8 if arr[j] > arr[j + 1]: 9 # Swap elements 10 arr[j], arr[j + 1] = arr[j + 1], arr[j] 11 12 return arr 13 14 15 # Example 16 numbers = [64, 34, 25, 12, 22, 11, 90] 17 print("Original list:", numbers) 18 19 sorted_list = bubble_sort(numbers) 20 print("Sorted list:", sorted_list) </pre>	<pre> Original list: [64, 34, 25, 12, 22, 11, 90] Sorted list: [11, 12, 22, 25, 34, 64, 90] === Code Execution Successful === </pre>

16. Using Divide and Conquer strategy to find Max and Min value in the list.

AIM:

To find maximum and minimum values using Divide and Conquer.

ALGORITHM:

Divide list into two halves

Find max and min of each half

Compare and combine results

Code:

```
#include <stdio.h>

int maxMin(int a[], int l, int r, int *max, int *min){

    if(l==r){

        *max=*min=a[l];

    } else {

        int mid=(l+r)/2, max1,min1;

        maxMin(a,l,mid,max,min);

        maxMin(a,mid+1,r,&max1,&min1);

        if(max1>*max) *max=max1;

        if(min1<*min) *min=min1;

    }

}

int main(){

    int a[10],n,max,min;

    scanf("%d",&n);

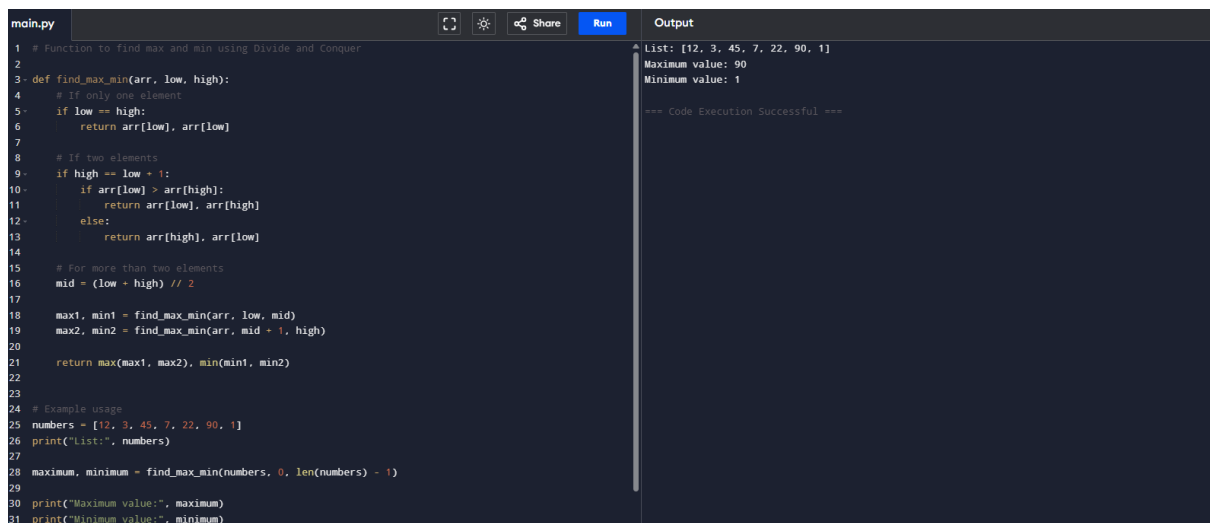
    for(int i=0;i<n;i++) scanf("%d",&a[i]);

    maxMin(a,0,n-1,&max,&min);

    printf("Max=%d Min=%d",max,min);

}
```

Output:



```
1 # Function to find max and min using Divide and Conquer
2
3 def find_max_min(arr, low, high):
4     # If only one element
5     if low == high:
6         return arr[low], arr[low]
7
8     # If two elements
9     if high == low + 1:
10        if arr[low] > arr[high]:
11            return arr[low], arr[high]
12        else:
13            return arr[high], arr[low]
14
15    # For more than two elements
16    mid = (low + high) // 2
17
18    max1, min1 = find_max_min(arr, low, mid)
19    max2, min2 = find_max_min(arr, mid + 1, high)
20
21    return max(max1, max2), min(min1, min2)
22
23
24 # Example usage
25 numbers = [12, 3, 45, 7, 22, 90, 1]
26 print("List:", numbers)
27
28 maximum, minimum = find_max_min(numbers, 0, len(numbers) - 1)
29
30 print("Maximum value:", maximum)
31 print("Minimum value:", minimum)
```

Output

List: [12, 3, 45, 7, 22, 90, 1]
Maximum value: 90
Minimum value: 1
--- Code Execution Successful ---

17. Write a program to generate all the prime numbers.

AIM:

To generate all prime numbers up to N.

ALGORITHM:

Read N

For each number from 2 to N

Check divisibility

If prime, print

Code:

```
#include <stdio.h>
```

```
int main(){
```

```
    int n,i,j,flag;
```

```
    scanf("%d",&n);
```

```
    for(i=2;i<=n;i++){
```

```
        flag=1;
```

```
        for(j=2;j<i;j++)
```

```
            if(i%j==0) flag=0;
```

```

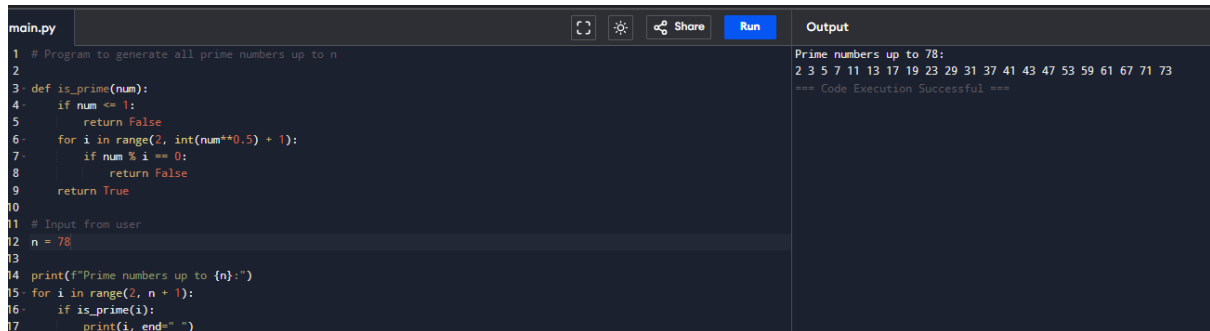
        if(flag) printf("%d ",i);

    }

}

```

Output:



The screenshot shows a code editor with a file named 'main.py'. The code is a Python program to generate all prime numbers up to a user-defined value 'n'. The code includes a function 'is_prime(num)' that checks for primality and a main loop that prints all primes up to 'n'. The output on the right shows the prime numbers up to 78: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73.

```

main.py
1 # Program to generate all prime numbers up to n
2
3 def is_prime(num):
4     if num <= 1:
5         return False
6     for i in range(2, int(num**0.5) + 1):
7         if num % i == 0:
8             return False
9     return True
10
11 # Input from user
12 n = 78
13
14 print(f"Prime numbers up to {n}:")
15 for i in range(2, n + 1):
16     if is_prime(i):
17         print(i, end=" ")

```

Output

```

Prime numbers up to 78:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
=== Code Execution Successful ===

```

18. Write a program to perform Knapsack problem using greedy techniques.

Aim:

To solve fractional knapsack problem using greedy approach.

ALGORITHM:

- Input weights and profits
- Sort by profit/weight ratio
- Select items until capacity is full

Code:

```

#include <stdio.h>

int main(){

    int n,cap,i;

    float profit=0;

    scanf("%d %d",&n,&cap);

    int w[10], p[10];

    for(i=0;i<n;i++) scanf("%d %d",&w[i],&p[i]);

```

```

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

            if(w[i]<=cap){

                profit+=p[i];

                cap-=w[i];

            }

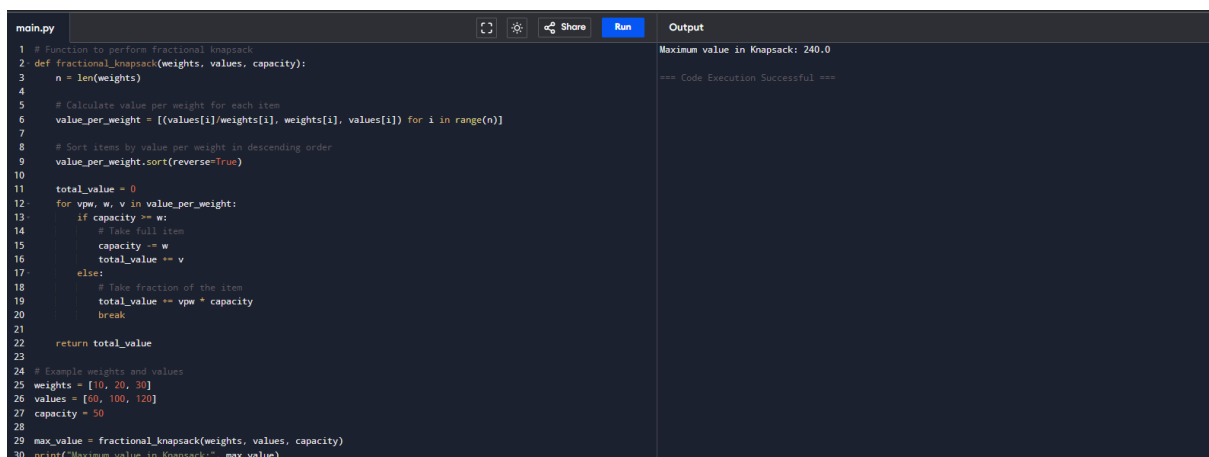
        }

        printf("Max Profit: %.2f",profit);

    }

```

Output:



```

main.py
1 # Function to perform fractional knapsack
2 def fractional_knapsack(weights, values, capacity):
3     n = len(weights)
4
5     # Calculate value per weight for each item
6     value_per_weight = [(values[i]/weights[i], weights[i], values[i]) for i in range(n)]
7
8     # Sort items by value per weight in descending order
9     value_per_weight.sort(reverse=True)
10
11     total_value = 0
12     for vpw, w, v in value_per_weight:
13         if capacity >= w:
14             # Take full item
15             capacity -= w
16             total_value += v
17         else:
18             # Take fraction of the item
19             total_value += vpw * capacity
20             break
21     return total_value
22
23
24 # Example weights and values
25 weights = [10, 20, 30]
26 values = [60, 100, 120]
27 capacity = 50
28
29 max_value = fractional_knapsack(weights, values, capacity)
30 print("Maximum value in Knapsack:", max_value)

```

Maximum value in Knapsack: 240.0

=== Code Execution Successful ===

19. Write a program to perform MST using greedy techniques.

AIM:

To find Minimum Spanning Tree of a graph using greedy technique.

ALGORITHM:

Read adjacency matrix

Select smallest edge

Check for cycle

Add edge to MST

Repeat until n-1 edges

Code:


```

#include <stdio.h>

int parent[10];

int find(int i){
    while(parent[i])
        i=parent[i];
    return i;
}

int uni(int i,int j){
    if(i!=j){
        parent[j]=i;
        return 1;
    }
    return 0;
}

int main(){
    int n,i,j,min,a,b,u,v;

    int cost[10][10];

    scanf("%d",&n);

    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&cost[i][j]);

    int ne=1,mincost=0;

    while(ne<n){

```

```

min=999;

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

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

        if(cost[i][j]<min){

            min=cost[i][j]; a=u=i; b=v=j;

        }

u=find(u); v=find(v);

if(uni(u,v)){

    printf("%d edge (%d,%d) cost=%d\n",ne++,a,b,min);

    mincost+=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("Min Cost=%d",mincost);

}

```

Output:

The screenshot shows a C++ IDE with a dark theme. The left pane displays the source code for a Minimum Spanning Tree (MST) algorithm using Kruskal's method. The code includes functions for finding the parent of a node, unioning two sets, and checking if two nodes are in the same set. The main function reads the number of vertices and edges, processes each edge, and prints the edges in the MST along with their costs. The right pane shows the output of the program, which lists the edges in the MST and the total minimum cost.

```

1 //C++ program to find MST
2 #include <iostream>
3 using namespace std;
4 class Edge
5 {
6     int u, v, w;
7     int u1, v1, w1;
8     int u2, v2, w2;
9     int u3, v3, w3;
10 };
11 //Function to find parent of a node
12 int find_parent(int parent[], int x)
13 {
14     if (parent[x] == x)
15         return x;
16     return find_parent(parent, parent[x]);
17 }
18 //Function to union two sets
19 void union_set(int parent[], int x, int y)
20 {
21     int root_x = find_parent(parent, x);
22     int root_y = find_parent(parent, y);
23     if (root_x != root_y)
24     {
25         parent[root_x] = root_y;
26         parent[root_y] = root_y;
27     }
28 }
29 //Function to check if two nodes are in the same set
30 bool is_same_set(int parent[], int x, int y)
31 {
32     int root_x = find_parent(parent, x);
33     int root_y = find_parent(parent, y);
34     return root_x == root_y;
35 }
36 //Function to find MST using Kruskal's algorithm
37 int find_MST(int V, int E, Edge edges[])
38 {
39     int result = 0;
40     int i;
41     //Sort edges by weight
42     sort(edges, edges + E);
43     //Create a disjoint set
44     int parent[V];
45     for (i = 0; i < V; i++)
46         parent[i] = i;
47     //Find MST using Kruskal's algorithm
48     for (i = 0; i < E; i++)
49     {
50         int u = edges[i].u;
51         int v = edges[i].v;
52         int w = edges[i].w;
53         if (!is_same_set(parent, u, v))
54         {
55             union_set(parent, u, v);
56             result += w;
57         }
58     }
59     return result;
60 }
61 //Driver code
62 int main()
63 {
64     int V = 4, E = 5;
65     Edge edges[E];
66     edges[0].u = 0, edges[0].v = 1, edges[0].w = 10;
67     edges[1].u = 0, edges[1].v = 2, edges[1].w = 6;
68     edges[2].u = 0, edges[2].v = 3, edges[2].w = 5;
69     edges[3].u = 1, edges[3].v = 3, edges[3].w = 4;
70     edges[4].u = 2, edges[4].v = 3, edges[4].w = 3;
71     int result = find_MST(V, E, edges);
72     cout << "Minimum Spanning Tree Cost: " << result << endl;
73     return 0;
74 }

```

Output

```

Edges in the Minimum Spanning Tree:
0 1 10
0 2 6
0 3 5
1 3 4
2 3 3
Min Cost: 28

```