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

```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 7
print("Fibonacci Series:")
for i in range(n):
    print(fibonacci(i),end=" ")
```

Output

```
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:

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:

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

AIM:

To find and print the largest element of an array.

Algorithm:

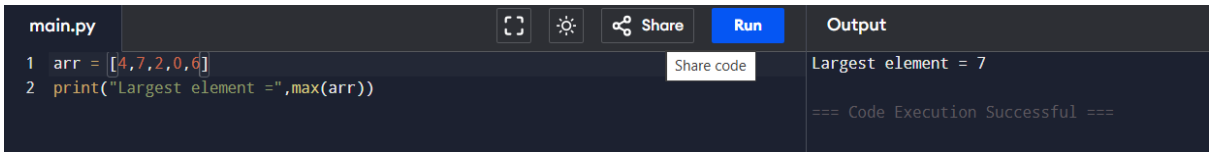
- Start.
- Read number of elements.
- Read array elements.
- Initialize max to first element.
- Loop to compare each element; update max as needed.
- Print max.
- Stop.

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:



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.

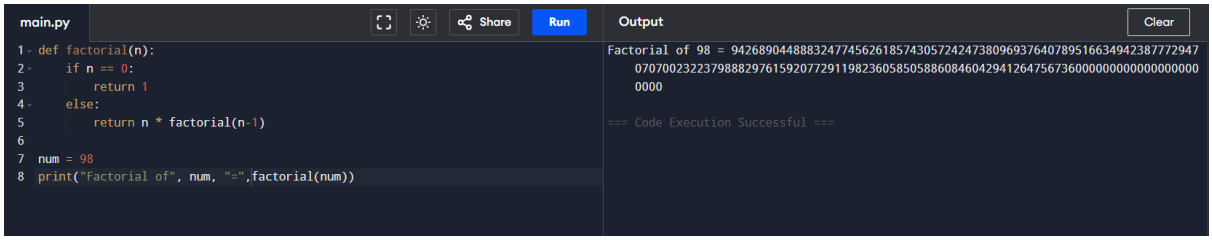
Algorithm:

- 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:



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 Share Run Output

```python
n = 19

if n <= 1:
    print(n, "is not a prime number")
else:
    for i in range(2, int(n/2)+1):
        if n % i == 0:
            print(n, "is not a prime number")
            break
    else:
        print(n, "is a prime number")
```

```
19 is a prime number

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

7. Write a program to perform Selection sort.

AIM:

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

Algorithm:

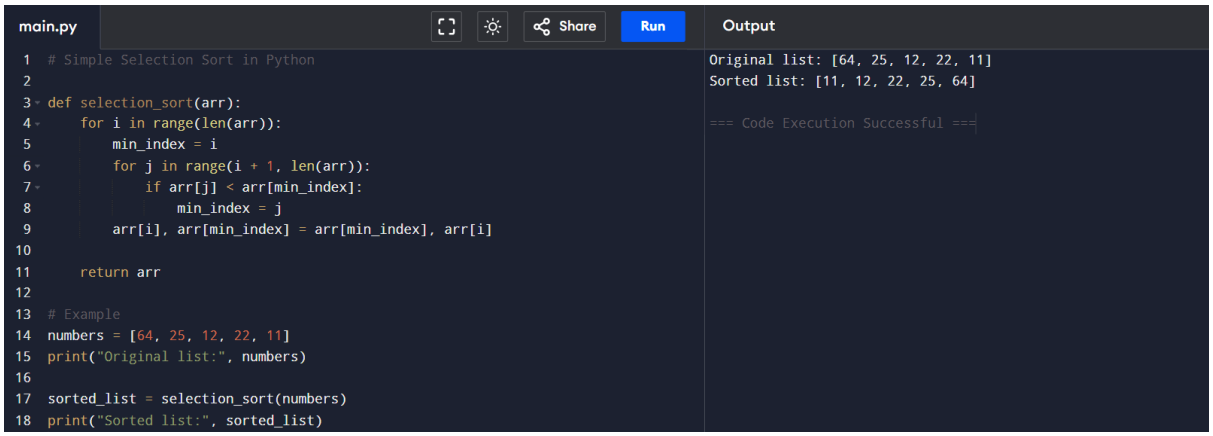
- Start.
- Read array size and elements.
- For each position in the array, find the minimum element in the unsorted portion and swap.
- Repeat until sorted.
- Print sorted array.
- Stop.

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:



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 | Share | Run | Output

```
# Simple Bubble Sort in Python

def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    return arr


# Example
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)

sorted_list = bubble_sort(numbers)
print("Sorted list:", sorted_list)
```

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list: [11, 12, 22, 25, 34, 64, 90]

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

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.

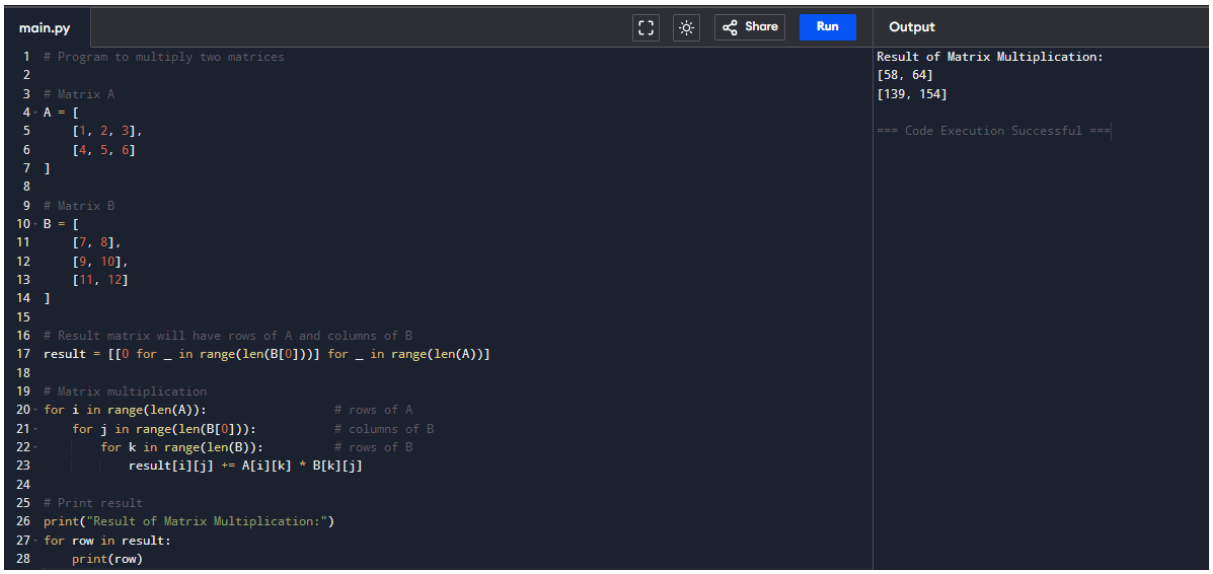
7. Multiply matrices using nested loops:
   For i = 0 to r1-1
   For j = 0 to c2-1
   For k = 0 to c1-1
   result[i][j] += matrix1[i][k] * matrix2[k][j]
8. Display the resultant matrix.
9. Stop

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:



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:

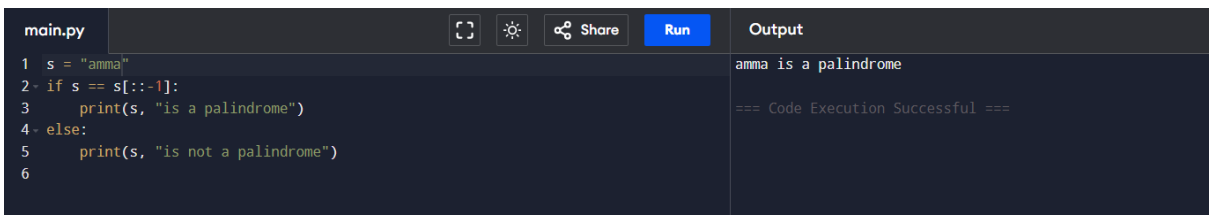
1. Start
2. Read the input string.
3. Initialize two indices, start=0 and end=length_of_string - 1.
4. Compare characters at start and end position.

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:



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

Aim:

To copy one string into another string.

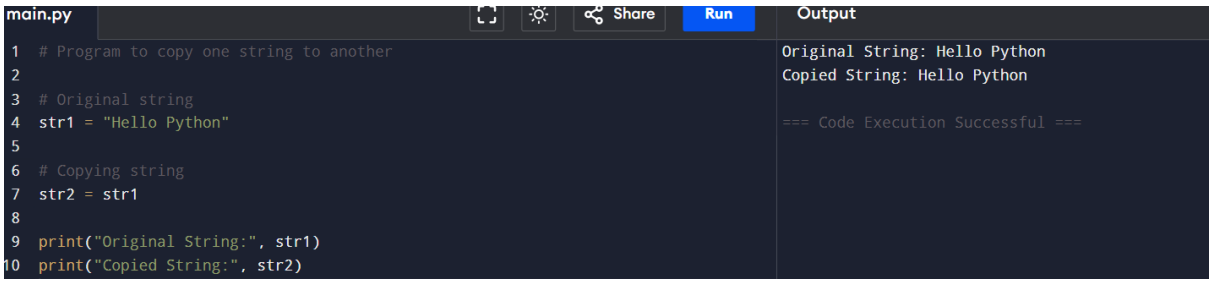
Algorithm:

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:



11. Write a Program to perform binary search.

Aim:

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

Algorithm:

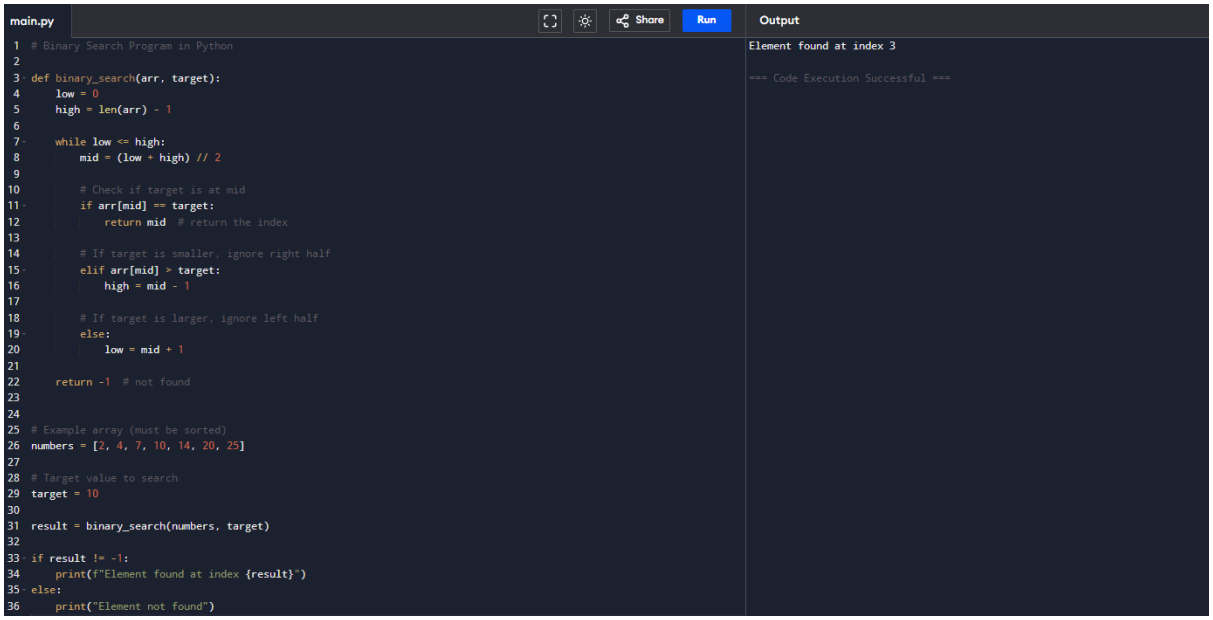
1. Start
2. Take sorted array input and target element.
3. Initialize low = 0, high = size-1.
4. Find mid = (low + high)/2.
5. If arr[mid] is the target, print index and stop.
6. If arr[mid] < target, set low = mid + 1, else high = mid -1.
7. Repeat steps 4-6 until low > high.
8. If element not found,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



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

Aim:

To print the reverse of a given string.

Algorithm:

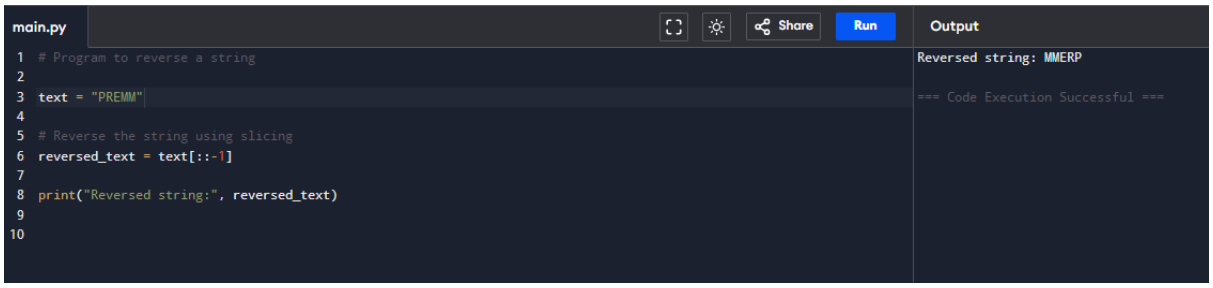
1. Start
2. Read the string.
3. Find length of the string.
4. Use a loop to print characters from the last to first.
5. Stop.

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:



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

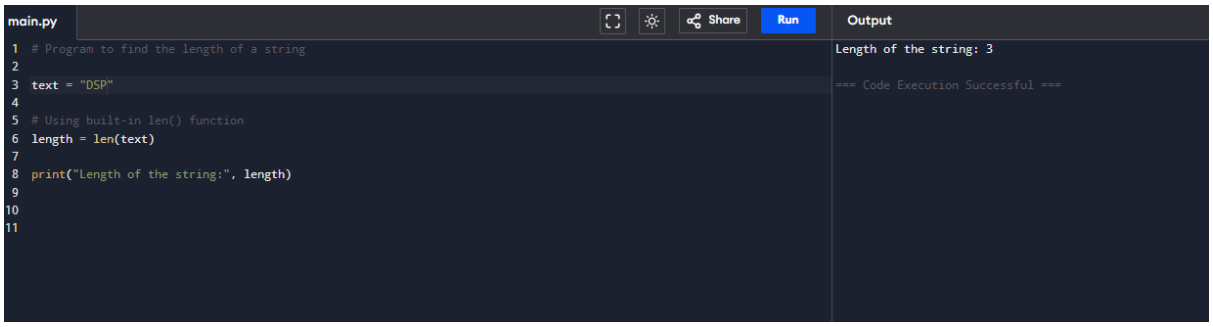
Display count

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:



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

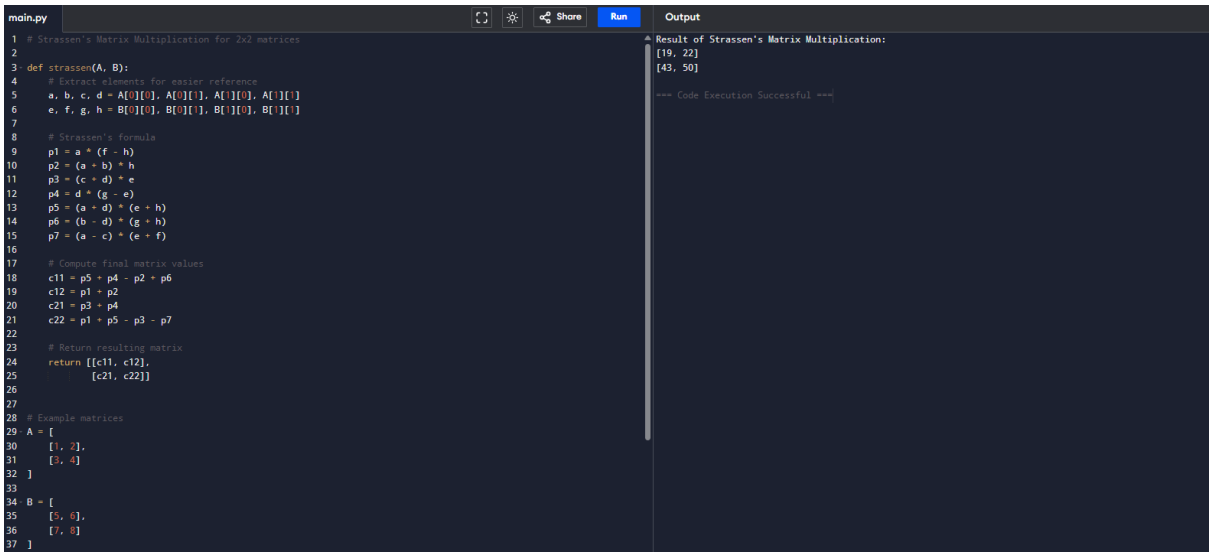
Combine results to form resultant matrix

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:



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 | Share | Run | Output

```
# Simple Bubble Sort in Python

def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    return arr


# Example
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)

sorted_list = bubble_sort(numbers)
print("Sorted list:", sorted_list)
```

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list: [11, 12, 22, 25, 34, 64, 90]

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

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

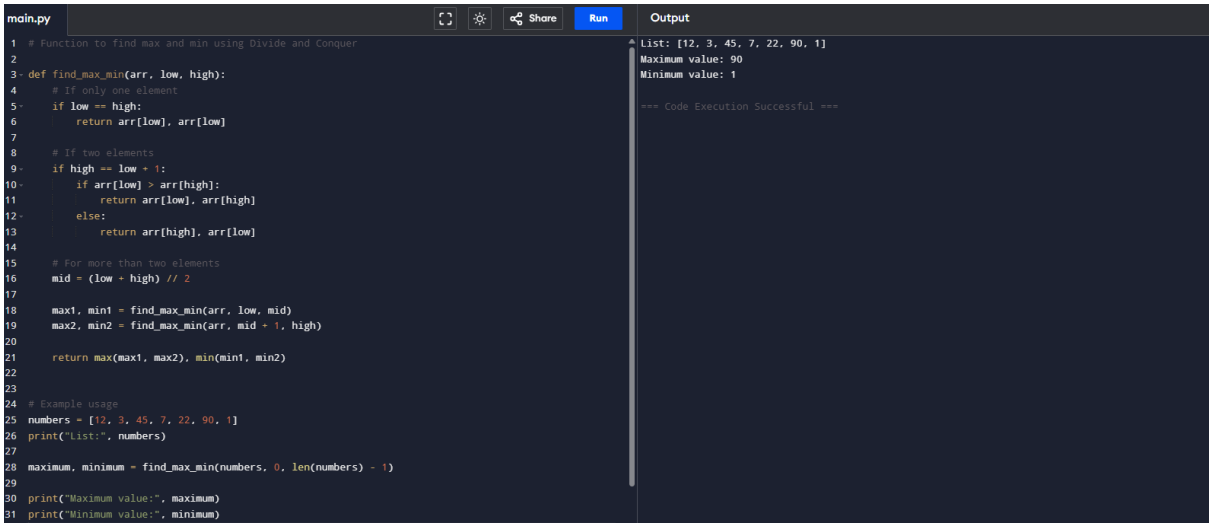
Find max and min of each half

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:



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

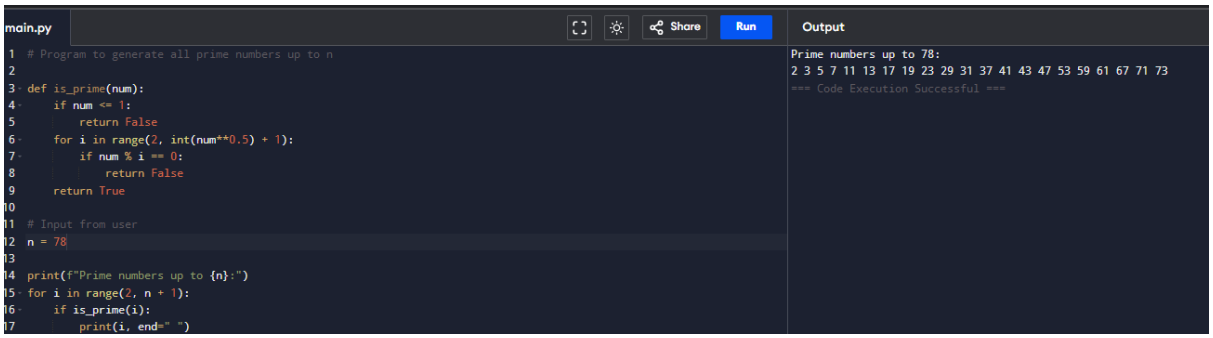
If prime, print

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:



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

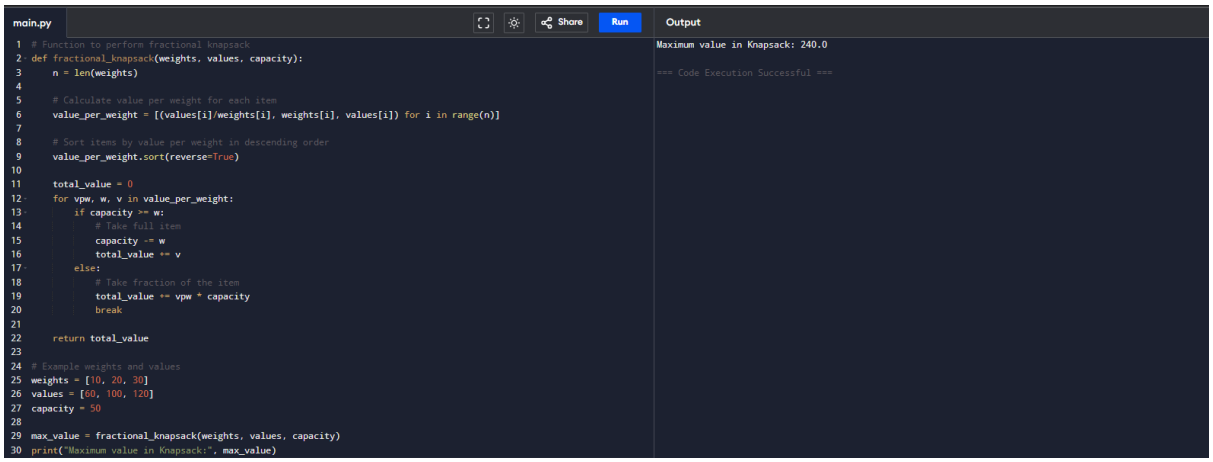
Select items until capacity is full

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:



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: