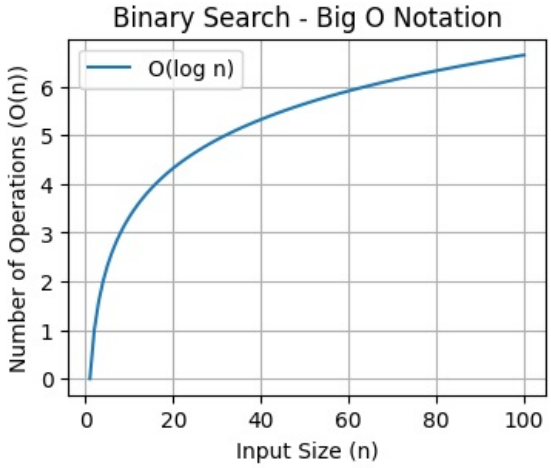
Big O Notations

O(log n)

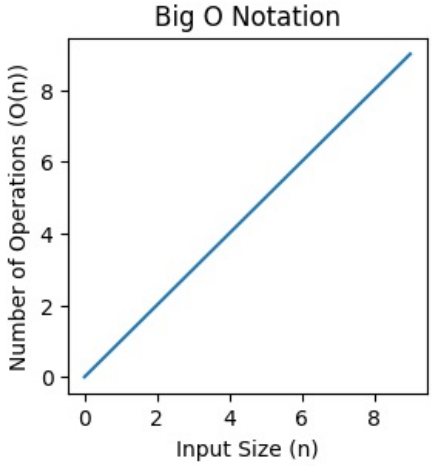
In [7]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(4, 3))
# Generate input sizes (n)
input_sizes = np.linspace(1, 100, 100)

# Calculate the number of operations (O(n)) for each input size
num_operations = [np.log2(n) for n in input_sizes]

# Plot the Big O notation
plt.plot(input_sizes, num_operations, label='O(log n)')
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n))')
plt.title('Binary Search - Big O Notation')
plt.legend()
plt.grid(True)
plt.show()
```



O(n)

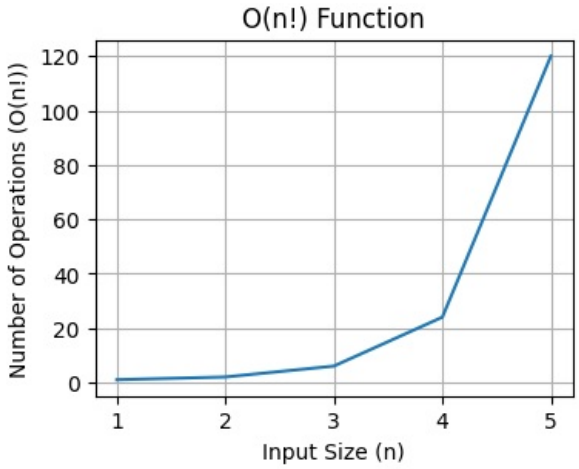
In [36]:
```
x = (list(range(0, 10)))
y = (list(range(0, 10)))

plt.figure(figsize=(3, 3))
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n))')
plt.title('Big O Notation')
```

Out[36]:
```
Text(0.5, 1.0, 'Big O Notation')
```



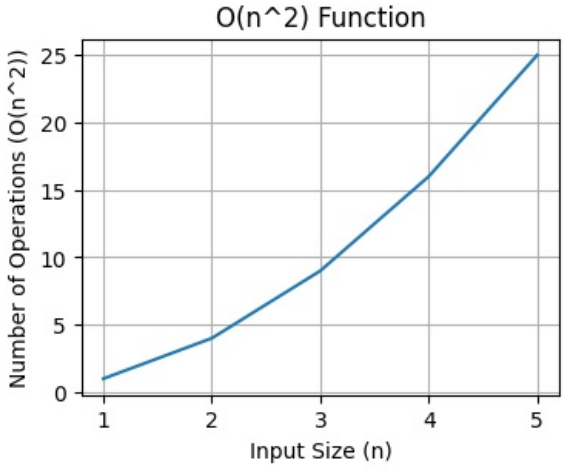
O(n!)

In [30]:
```
import math

x = [1, 2, 3, 4, 5]
y = [math.factorial(n) for n in x]

plt.figure(figsize=(4, 3))
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n!))')
plt.title('O(n!) Function')
plt.grid(True)
plt.show()
```



O(n2)

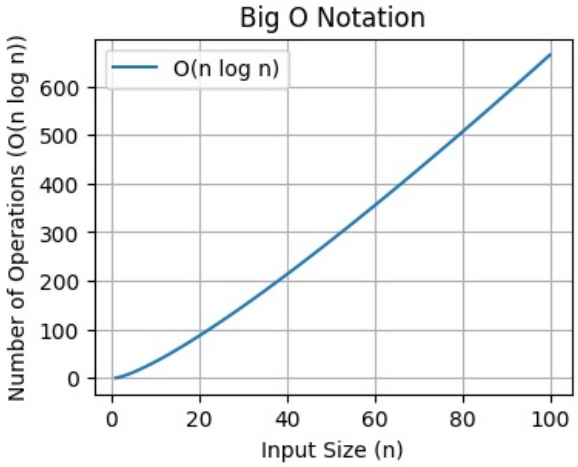
In [31]:
```
x = [1, 2, 3, 4, 5]
y = [n ** 2 for n in x]

plt.figure(figsize=(4, 3))
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n^2))')
plt.title('O(n^2) Function')
plt.grid(True)
plt.show()
```



O(n log n)

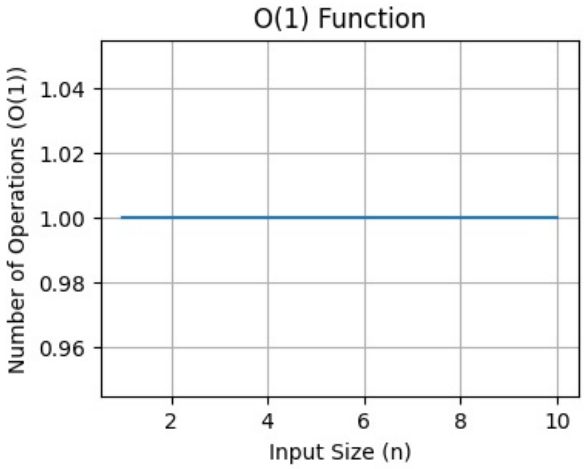
In [34]:
```
plt.figure(figsize=(4, 3))
input_sizes = np.linspace(1, 100, 100)

num_operations = [n * np.log2(n) for n in input_sizes]


plt.plot(input_sizes, num_operations, label='O(n log n)')
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n log n))')
plt.title('Big O Notation')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

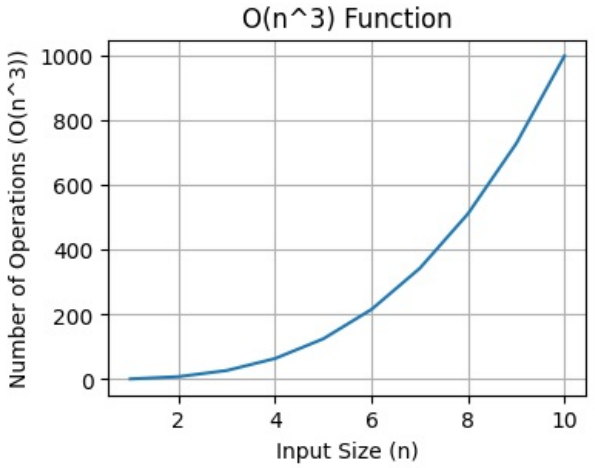


O(1)

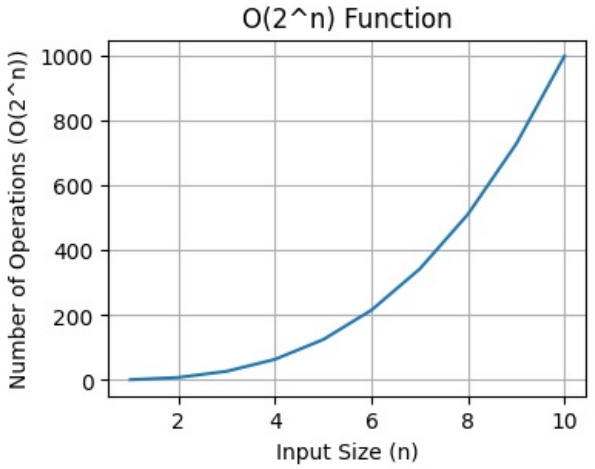
In [37]:
```
x = np.linspace(1, 10, 10)
y = [1] * len(x)

plt.figure(figsize=(4, 3))
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(1))')
plt.title('O(1) Function')
plt.grid(True)
plt.show()
```



In [41]:
```
def cubic_function(n):
    operation_counts = []
    for i in n:
        count = i ** 3
        operation_counts.append(count)
    return operation_counts

# Generate input sizes
plt.figure(figsize=(4, 3))
x = np.arange(1, 11)

# Calculate the number of operations
y = cubic_function(x)
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(n^3))')
plt.title('O(n^3) Function')
plt.grid(True)
plt.show()
```



In [43]:
```
def exponential(n):
    operation_counts = []
    for i in n:
        count = 2 ** n
        operation_counts.append(count)
    return operation_counts

# Generate input sizes
plt.figure(figsize=(4, 3))
x = np.arange(1, 11)

# Calculate the number of operations
y = cubic_function(x)
plt.plot(x, y)
plt.xlabel('Input Size (n)')
plt.ylabel('Number of Operations (O(2^n))')
plt.title('O(2^n) Function')
plt.grid(True)
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js