```
[Running] python -u
"e:\MyFiles\Clg-Files\Laboratory\DAA-lab\1)fibonaci.
py"
Using Recursion:
0 1 1 2 3 5 8
Without using recursion:
0 1 1 2 3 5 8
[Done] exited with code=0 in 2.447 seconds
```

```python
def recurFib(n):
    if n<= 1:
        return n
    else:
        return recurFib(n-1)+recurFib(n-2)
n = int(input('Enter a number: '))
print('Using Recursion:')
if n == 0: print(n)
else:
    for i in range(n):
        print(recurFib(i), end=' ')
    print()

def fib(n):
    f,s = 0,1
    while (n>0):
        print(f, end=' ')
        f,s = s, f+s
        n = n-1

print('Without using recursion:')
fib(n)
```

```python
import numpy as np

def strassen_multiply(A, B):
    n = len(A)

    # Base case for recursion
    if n <= 2:
        return np.dot(A, B)

    # Splitting matrices into quadrants
    mid = n // 2
    A11 = A[:mid, :mid]
    A12 = A[:mid, mid:]
    A21 = A[mid:, :mid]
    A22 = A[mid:, mid:]

    B11 = B[:mid, :mid]
    B12 = B[:mid, mid:]
    B21 = B[mid:, :mid]
    B22 = B[mid:, mid:]

    # Recursive steps
    P1 = strassen_multiply(A11 + A22, B11 + B22)
    P2 = strassen_multiply(A21 + A22, B11)
    P3 = strassen_multiply(A11, B12 - B22)
    P4 = strassen_multiply(A22, B21 - B11)
    P5 = strassen_multiply(A11 + A12, B22)
    P6 = strassen_multiply(A21 - A11, B11 + B12)
    P7 = strassen_multiply(A12 - A22, B21 + B22)

    # Calculating quadrants of the result matrix
    C11 = P1 + P4 - P5 + P7
    C12 = P3 + P5
    C21 = P2 + P4
    C22 = P1 + P3 - P2 + P6

```

```
[Running] python -u
"e:\MyFiles\Clg-Files\Laboratory\DAA-lab\2)
matrix-mul.py"
[[ 90. 100. 110. 120.]
 [202. 228. 254. 280.]
 [314. 356. 398. 440.]
 [426. 484. 542. 600.]]

[Done] exited with code=0 in 0.75 seconds
```

```
    # Combining quadrants into the result matrix
    C = np.zeros((n, n))
    C[:mid, :mid] = C11
    C[:mid, mid:] = C12
    C[mid:, :mid] = C21
    C[mid:, mid:] = C22
    return C

A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])
B = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])
C = strassen_multiply(A, B)
print(C)
```

```
[Running] python -u
"e:\MyFiles\Clg-Files\Laboratory\DAA-lab\3)
topological-sort.py"
[1, 3, 5, 2, 4, 6]

[Done] exited with code=0 in 0.803 seconds
```

```python
from collections import defaultdict
def topological_sort(graph):
    visited = set()
    result = []
    def dfs(node):
        visited.add(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs(neighbor)
        result.append(node)
    for node in graph:
        if node not in visited:
            dfs(node)
    result.reverse()
    return result

graph = defaultdict(list)
graph[1] = [2, 3]
graph[2] = [4]
graph[3] = [4, 5]
graph[4] = [6]
graph[5] = [6]
graph[6] = []

sorted_vertices = topological_sort(graph)
print(sorted_vertices)
```

```
[Running] python -u "e:\MyFiles\Clg-
Files\Laboratory\DAA-lab\4)Heap-sort.py"
Sorted array is: [5, 6, 7, 11, 12, 13]

[Done] exited with code=0 in 0.587 seconds
```

```python
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and arr[i] < arr[left]:
        largest = left
    if right < n and arr[largest] < arr[right]:
        largest = right
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

arr = [12, 11, 13, 5, 6, 7]
heap_sort(arr)
print("Sorted array is:", arr)
```