Assignment 4

Dhruv Kulkarni
U21CS036

April 21, 2023

1 Skyline problem

The following is the solution to the skyline problem using a divide and conquer approach.

```
def skyline(buildings):
    n = len(buildings)
    if n == 1:
        x1, x2, h = buildings[0]
        return [(x1, h), (x2, 0)]

    mid = n // 2
    left = skyline(buildings[:mid])
    right = skyline(buildings[mid:])
    return merge_skylines(left, right)

def merge_skylines(left, right):
    i, j = 0, 0
    h1, h2 = 0, 0
    merged = []

    while i < len(left) and j < len(right):
        x1, h1 = left[i]
        x2, h2 = right[j]

        if x1 < x2:
            h = max(h1, h2)
            merged.append((x1, h))
            i += 1
        elif x1 > x2:
            h = max(h1, h2)
            merged.append((x2, h))
            j += 1
        else:
            h = max(h1, h2)
            merged.append((x1, h))
            i += 1
            j += 1
```

```
    merged.extend(left[i:])
    merged.extend(right[j:])

    return merged


buildings = [(33, 41, 5), (4, 9, 21), (30, 36, 9), (14, 18, 11),
             (2, 12, 14), (34, 43, 19), (23, 25, 8), (14, 21, 16),
             (32, 37, 12), (7, 16, 7), (24, 27, 10)]
result = skyline(buildings)
print(result)
```

Intuition:Any divide and conquer approach involves these things: Divide the problem into subproblems, solve them and combine them into one single problem to get a final solution. Here the skyline() function divides it and the merge function combines them.Time complexity is (nlogn).

2 Skyline problem-iterative

The following is the solution to the skyline problem using an iterative approach.

```
def get_skyline(buildings):

    skyline = []
    active_buildings = []
    processed_buildings = []
    points = [(x, h, 'start') for x, _, h in buildings] + [(x, h, 'end'
    points.sort()


    for x, h, event_type in points:
        # If the current point is the start of a building, add it to the
        if event_type == 'start':
            active_buildings.append((h, x))
         active buildings list
        else:
            active_buildings.remove((h, x))


        max_height = max([h for h, _ in active_buildings] + [0])


        if not skyline or max_height != skyline[-1][1]:
            skyline.append((x, max_height))

    return skyline
```

3 Matrix multiplication using incremental approach

The following is the solution to the matrix multiplication problem using an incremental approach.

```
def matrix_multiply(A, B):
    n = len(A)
    m = len(B[0])
    C = [[0] * m for _ in range(n)]

    for i in range(n):
        for j in range(m):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]

    return C



A = [[1, 2,3]]
B = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
result = matrix_multiply(A, B)
print(result)
```

Intuition:Here we simply multiply the two given matrices using Matrix multiplication method, by using three for loops and the common mathematical incremental approach.Time complexity is (n3).

4 Matrix multiplication using DnC-Strassen's algorithm

The following is the solution to the matrix multiplication using a divide and conquer approach, and this algorithm is known as Strassen's matrix multiplication algorithm.

```
import numpy as np

def strassen_multiply(A, B):
    n = A.shape[0]


    if n == 1:
        return A * B


    A11 = A[:n//2, :n//2]
    A12 = A[:n//2, n//2:]
    A21 = A[n//2:, :n//2]
    A22 = A[n//2:, n//2:]
    B11 = B[:n//2, :n//2]
    B12 = B[:n//2, n//2:]
    B21 = B[n//2:, :n//2]
    B22 = B[n//2:, n//2:]


    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)

    C11 = P1 + P4 - P5 + P7
    C12 = P3 + P5
    C21 = P2 + P4
```

```
    C22 = P1 - P2 + P3 + P6

    C = np.zeros((n, n))
    C[:n//2, :n//2] = C11
    C[:n//2, n//2:] = C12
    C[n//2:, :n//2] = C21
    C[n//2:, n//2:] = C22

    return C
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = strassen_multiply(A, B)
print(C)
```

Intuition:Any divide and conquer approach involves these things: Divide the problem into subproblems, solve them and combine them into one single problem to get a final solution. Here the strassens() function divides the matrices into 4 quadrants in order to solve them.