שאלה 1:

Given a stack with a limit of k elements, where each k operations that is a copy operation that take O(I) where I is the number of elements in the stack.

Operation running time:

$$push = O(1)$$

$$pop = O(1)$$

$$copy_i = O(i) \leq O(k)$$

Here we wrote the running time for each operation.

Let's use amortize analysis to solve this.

Firstly, we need to understand that the copy running time is bounded by

$$O(k)$$

Because we are given a stack that can't exceed k elements.

Copy operations

$$\left\lfloor \frac{m}{k} \right\rfloor$$

Running time analysis

$$E = \frac{\left[\sum_{i=0}^{i=\left\lfloor \frac{m}{k} \right\rfloor} O(k) + \sum_{i=0}^{i=m} O(1)\right]}{m} \leq \frac{\left[\sum_{i=0}^{i=\frac{m}{k}} O(k) + \sum_{i=0}^{i=m} O(1)\right]}{m}$$

$$\frac{\left[\sum_{i=0}^{i=\frac{m}{k}} O(k) + \sum_{i=0}^{i=m} O(1)\right]}{m} = \frac{\left[\frac{m}{k} \cdot O(k) + O(m)\right]}{m}$$

$$\frac{\left[\frac{m}{k} \cdot O(k) + O(m)\right]}{m} = O(1) + O(1) = O(1)$$

שאלה 2:

$$\ell(S) = 4 \cdot S$$

$$c_i = \Delta \ell(S) = \ell(\overset{new}{D_n}) - \ell(\overset{old}{D_0})$$

$O(1)$ - Insert

$O(|S|)$ - R.B.A

$$a = c \cdot S + \Delta S = cS + \ell(D_n) - \ell(D_n)$$

$$a = cS + 4 \cdot \frac{3}{4} \cdot S - 4S = cS - \frac{1}{4} 4S$$

$c \geq 4 \geq 4$ נקבל

$$a = cS - \frac{1}{4} 4S \leq cS - \frac{1}{4} 4 \cdot c \cdot S = 0$$

קבלנו שעבור נקודה תסומה על ידי $O(1)$

שאלה 3:

כאשר יש לנו עץ 2-3 הגובה יכול לנוע בין

$$\log_3(n) \leq h \leq \log_2(n)$$

ב:

להבנתי מדובר על פעולה של search, אפשר לבצע פעולה זו בצורה יחסית מהירה

$$O(\log_2(n))$$

על ידי האלגוריתם הבא

```python
# Description: Search in 2-3 tree
def member(t, x):
    if t is None:
        return False
    if t.is_leaf():
        return x in t.keys
    if x < t.keys[0]:
        return member(t.children[0], x)
    if len(t.keys) == 3 and x >= t.keys[1]:
        return member(t.children[2], x)
    if len(t.keys) == 3 and x >= t.keys[0]:
        return member(t.children[1], x)
    return member(t.children[1], x)
```

ג:

```python
def insert(t, v):
    loc = search(t, v)
    loc.push(v)
    if loc.children.len > MAX:
        # merge either the left or right child
        if loc.left.children.len < MAX:
            merge(loc, loc.left)

        elif loc.right.children.len < MAX:
            merge(loc, loc.right)

        else:
            # split the node
            split(loc)
            insert(t, v)
```

שאלה 4:

We will describe the following algorithm:

First, we will assert W.L.O.G that h1 > h2 and T1 >T2 the algorithm will be very similar for T2<T1.

We will move h2 steps on T1, so we get loc = T1.moveleft(h2) == O(h2)

We will append T2 root to loc == O(1)

If loc.count == 4: O(1)

CreateNewParent(loc) == O(h1)

Else if loc.count == 3: O(1)

FixErrorFrom(loc) == O(h1)

DONE

```
Function CreateNewParent(loc): O(H)
    P = Take the loc[2] and split the items
    P.parent.push(P)
    If P.silblings.len > MAX:
        FixErrorFrom(P)
    Return P


Function FixErrorFrom(loc): O(H)
   P = Take the middle child and push it to the parent
   If P.siblings > MAX:
        FixErrorFrom(Loc)
```