Name: Yogendra Pokhrel

617585

Q1A. Start with an empty stack of integers. You will attempt to do a sequence of pushes and pops so that the sequence of pops will be a specified permutation of 1, 2, 3 ,4 , 5, 6. You will be able to do exactly 6 push operations and 6 pop operations. The first push pushes 1 onto the stack; the next pushes 2; and so forth. The sixth push pushes 6 onto the stack. For this exercise, we will let S denote a push operation and X a pop operation. Example: The sequence SSSSSSXXXXXX outputs 654321.

A. Describe a sequence of pushes and pops that would produce output 325641 (or explain why it is not possible)

Stack Simulation Rules:

- Elements are pushed in increasing order: 1 2 3 4 5 6
- At any point, you can only pop the top of the stack.
- You must perform 6 pushes and 6 pops exactly.
- The output sequence must be the result of the pops.

a. Output sequence: 325641

We'll simulate this with step tracking.

Target output: 3 2 5 6 4 1

Let's simulate step-by-step:

Step	Stack	Action	Next to output	Comment
1	1	S	3	Push 1
2	1 2	S	3	Push 2
3	1 2 3	S	3	Push 3
4	1 2	X	2	Pop 3
5	1	X	5	Pop 2
6	1 4	S	5	Push 4
7	1 4 5	S	5	Push 5
8	1 4	X	6	Pop 5
9	1 4 6	S	6	Push 6
10	1 4	X	4	Pop 6
11	1	X	1	Pop 4
12		X	-	Pop 1

Output matches: 3 2 5 6 4 1

Valid sequence

Answer:

SSSXXSSXSXX produces 325641

B. Output sequence: 154623

Target output: 1 5 4 6 2 3

Let's simulate step-by-step:

Step	Stack	Action	Next to output	Comment
1	1	S	1	Push 1
2		X	5	Pop 1
3	2	S	5	Push 2
4	2 3	S	5	Push 3
5	2 3 4	S	5	Push 4
6	2 3 4 5	S	5	Push 5
7	2 3 4	X	4	Pop 5
8	2 3	X	6	Pop 4
9	2 3 6	S	6	Push 6
10	2 3	X	2	Pop 6
11	2	X	3	Pop 3 ❌ but expected 2

Problem: Stack top is 3, but next expected is 2. That violates the stack's LIFO rule.

You cannot pop 2 after 3 unless 3 is popped first, which would break the required sequence.

Answer

Not possible, the sequence 154623 cannot be produced by any valid sequence of stack operations.

Q1 B. Suppose we store n keys in a hash table of size m = n^2 using a hash function h randomly chosen from a Universal class H of hash functions. Assume that X is a random variable that counts the number of collisions. Show that the Expected number of Collisions is < 1/2.

We are given:

- n keys
- A hash table of size m= n^2
- A universal hash function hh, where for any two different keys $x_i \neq x_j$, the chance of collision is at most 1/m
- Let X be the number of collisions (i.e., how many pairs of keys get the same hash value)

Step-by-step:

1. There are (n 2)=n(n−1)/2 possible key pairs.

2. For each pair, the chance of a collision is at most 1/m, since we are using a universal hash function.

3. So, the expected number of collisions is:

$$E[X] \leq (n\ 2) \times 1/\ m$$
$$=(n(n-1)/2) \times (1/n^2)$$
$$= (n - 1) / 2n$$

4. Since (n - 1) / 2n < 1 / 2 for all n >= 1, we conclude:

$$E[X] < 1/2$$

So, when using a universal hash function and a table of size n^2, the expected number of collisions is less than 0.5.

Q2. For each integer n = 1,2,3,…, 7, determine whether there exists a red-black tree having exactly n nodes, with all of them black. Fill out the chart below to tabulate the results:

To answer this question, we need to understand how red-black trees work, especially when all nodes are black.

Key Red-Black Tree Properties:

1. Every node is either red or black
2. The root is always black
3. No two red nodes can be adjacent
4. Every path from a node to its descendant NIL nodes must contain the same number of black nodes

For this problem:

We're only considering red-black trees with all black nodes — no red nodes at all. That means:

- The tree must be a perfect binary tree (completely balanced)
- All nodes, including the root and internal nodes, are black
- The number of nodes n must fit a complete binary tree with only black nodes

That means only certain values of n are allowed. Specifically, we need to check which values of n allow forming a full binary tree with all black nodes.

Strategy:

A perfect binary tree of height hh has exactly:

$$n=2^h-1 \text{ (for } h=1,2,3,\ldots)$$

So we check:

- $h=1 \rightarrow 2^1-1 = 1$
- $h=2 \rightarrow 2^2-1 = 2$
- $h=3 \rightarrow 2^3-1 = 3$

Only 1, 3, and 7 fit this pattern.

Final Table:

Num nodes n	**Exists? (All nodes black)**
1	Yes
2	**No**
3	**Yes**
4	**No**
5	**No**
6	**No**
7	**Yes**

Q3. For each integer n = 1,2,3,…, 7, determine whether there exists a red-black tree having exactly n nodes and exactly one red node. Fill out the chart below to tabulate the results:

Red-Black Tree Constraints :

- The root must be black.
- No two red nodes can be adjacent.
- Every path from any node to its descendant NIL nodes must contain the same number of black nodes.

So, to have exactly one red node, we need to:

- Add one red node without violating any rule
- Ensure the red node is not the root and not connected to another red node

Let's go through each value from **2 to 7** and check whether it's possible:

Case-by-case:

n	Tree Possible with 1 Red Node?	Reason
2	**Yes**	Root is black, one red child. Balanced black height paths.
3	**Yes**	Root black, one black child, one red child — still balanced.
4	**Yes**	Root black, two children (one red, one black) — valid.
5	**Yes**	Root black, build left or right-heavy tree with one red node.

6	**Yes**	Can structure subtree to include one red node safely.
7	**Yes**	Several valid structures — one red leaf or intermediate node.

Final Table:

Num nodes n	Exists? (Exactly one red node)
1	No
2	Yes
3	Yes
4	Yes
5	Yes
6	Yes
7	Yes

Q4. show the red-black tree that results after each of the integer keys 21,32,64,75 and 15 are inserted, in that case into an initially empty red-black tree. Clearly show the tree that results after each insertion(indication the collision on each node) and make clear any rotations that must be performed.

Let’s walk through the insertion of keys 21, 32, 64, 75, and 15 into a Red-Black Tree, one step at a time. I’ll clearly show:

- The tree after each insertion
- Any coloring
- Any rotation and recoloring done to maintain red-black properties

Insertion Rules Recap:

1. New nodes are always inserted as red
2. If the parent is black → no fix needed
3. If the parent is red → fix with recoloring or rotation (based on uncle's color)
4. The root is always black

Step-by-Step Insertion

Insert 21

- Tree: Only one node → make it root
- 21 becomes black

[21B]

Insert 32

- 32 is red, inserted as right child of 21 (black parent) → no fix needed

```
 [21B]
     \
    [32R]
```

No violations

Insert 64

- 64 is red, inserted as right child of 32 (which is red) → violation: red-red
- Uncle is null (black) → Left Rotation on 21
- Then recolor 32 to black and 21 to red

Left Rotate on 21
Recolor 32 → black, 21 → red

```
   [32B]
   /   \
[21R]  [64R]
```

Insert 75

- 75 is red, inserted as right child of 64 (which is red) → violation: red-red
- Uncle is null → need left rotation on 64’s parent (32’s right child)

Left Rotate on 64's parent → 64
Recolor 64 → black, 32 → red (but since 32 is root, we recolor it back to black at the end)

```
    [32B]
    /    \
 [21R]   [64B]
           \
          [75R]
```

Fixed with one rotation and recolor

Insert 15

- 15 is red, inserted as left child of 21 (red) → violation: red-red
- Uncle (64) is black

Recolor: 21 and 15 stay (21 becomes black), and no rotation needed — we recolor 21 → black and 32 → red
But since 32 is the root, it must stay black.

```
    [32B]
    /    \
 [21B]   [64B]
  /          \
[15R]       [75R]
```

Balanced tree with valid red-black properties

Final Tree (after all insertions):

```
      [32B]
      /     \
  [21B]    [64B]
  /            \
[15R]         [75R]
```

No red-red violations
Black height consistent
Root is black