CODE:

from sortedcontainers import SortedList

class BloomFilter:

def \_init\_(self, size, hash\_functions):

self.size = size

self.hash\_functions = hash\_functions

self.bit\_array = [False] \* size

def \_hash(self, item, seed):

# Simple hash function using Python's built-in hash()

return hash(str(item) + str(seed)) % self.size

def add(self, item):

for i in range(self.hash\_functions):

index = self.\_hash(item, i)

self.bit\_array[index] = True

def contains(self, item):

for i in range(self.hash\_functions):

index = self.\_hash(item, i)

if not self.bit\_array[index]:

return False

return True

class BPlusTreeWithBloomFilter:

def \_init\_(self, bf\_capacity, tree\_order, bloom\_size=1000, bloom\_hash\_functions=3):

self.bloom\_filter = BloomFilter(bloom\_size, bloom\_hash\_functions)

self.bplus\_tree = SortedList()

self.tree\_order = tree\_order

self.bf\_capacity = bf\_capacity

def insert(self, key):

if self.bloom\_filter.contains(key):

return

self.bloom\_filter.add(key)

self.bplus\_tree.add(key)

if len(self.bplus\_tree) > self.tree\_order:

self.split\_tree()

def search(self, key):

if self.bloom\_filter.contains(key):

# Key may exist in the tree

return key in self.bplus\_tree

return False

def delete(self, key):

if not self.bloom\_filter.contains(key):

return

self.bloom\_filter.add(key)

self.bplus\_tree.remove(key)

def split\_tree(self):

middle\_index = len(self.bplus\_tree) // 2

middle\_key = self.bplus\_tree[middle\_index]

left\_subtree = self.bplus\_tree[:middle\_index]

right\_subtree = self.bplus\_tree[middle\_index:]

self.bplus\_tree = SortedList(left\_subtree)

new\_tree = BPlusTreeWithBloomFilter(self.bf\_capacity, self.tree\_order)

for key in right\_subtree:

new\_tree.insert(key)

self.bplus\_tree.update(new\_tree.bplus\_tree) # Replace extend with update

def print\_tree(self):

print(self.bplus\_tree)

# Example usage

hybrid\_ds = BPlusTreeWithBloomFilter(bf\_capacity=1000, tree\_order=4)

hybrid\_ds.insert(10)

hybrid\_ds.insert(20)

hybrid\_ds.insert(30)

hybrid\_ds.insert(40)

hybrid\_ds.insert(50)

hybrid\_ds.print\_tree() # [10, 20, 30, 40, 50]

print(hybrid\_ds.search(30)) # True

print(hybrid\_ds.search(60)) # False

hybrid\_ds.delete(40)

hybrid\_ds.print\_tree() # [10, 20, 30, 50]