LP4 Analysis

Memory Inefficient but speed efficient

HashMap<<IDS>,<Product Objects>> → Main Storage
HashMap<<Unique Description Long Int>,<HashSet Of Ids>>

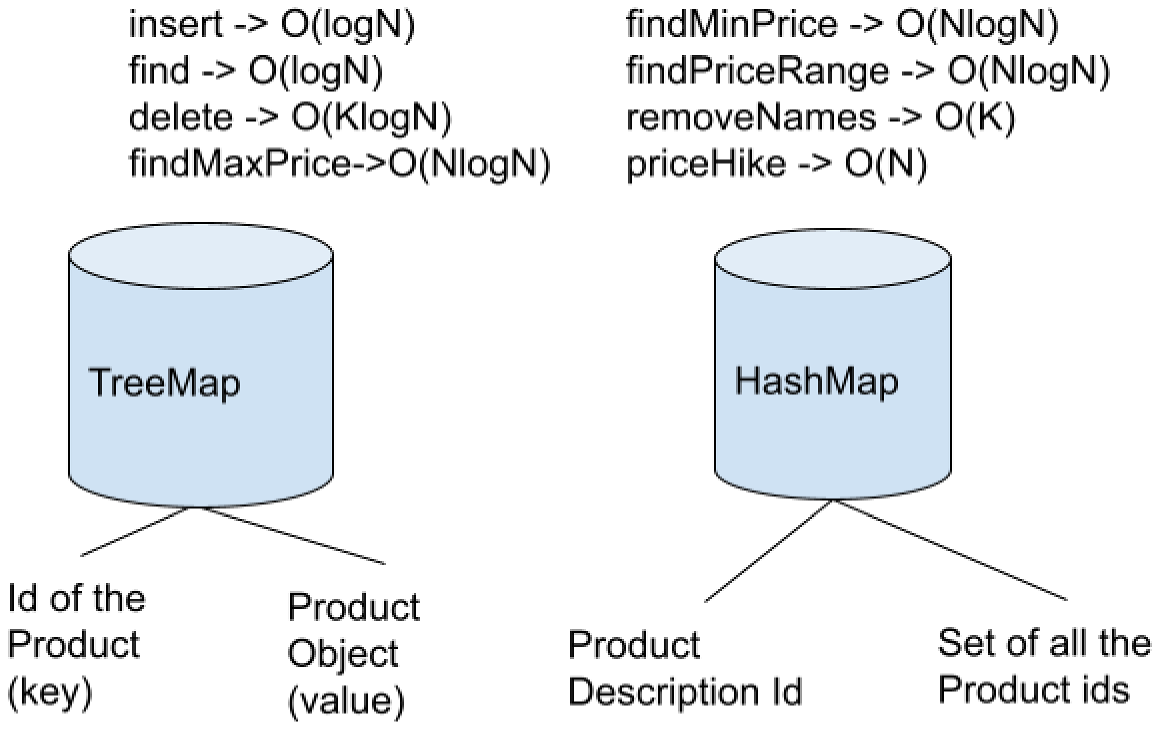
Memory Efficient but Speed Inefficient:

TreeMap<<IDS>,<Product Objects>> → Main Storage
TreeMap<<Unique Description Long Int>,<HashSet Of Ids>>

In this Project we are implementing TreeMap for the storage of items associated with Ids.
And HashMap and HashSet for storing description values associated with set of Product Ids.

Final Consideration:

TreeMap<<IDS>,<Product Objects>> → Main Product Storage
HashMap<<Unique Description Long Int>,<HashSet Of Ids>> -> Chosen HashSet over TreeSet for better performance assuming that size of IDs of a particular description is small.



Functions with Their Time Complexity:
--

1) insert(id,price,list):

Id:int/long
price:double/float
Description: HashSet of list of LongInt

HashMap : TC O(1) but memory Inefficient.
TreeMap: TC O(logN) ,stores in natural Order, Memory Efficient

Time Complexity -

1) Inserting items in TreeMap - O(logN+logK) where K is the size of description list. So, in the WC time complexity will be O(logN) or O(logK), depending on which one is greater. K>N, if there is a description List for a product which might be greater than the number of products in the whole Storage.
2) Inserting descriptions in HashMap - O(1) amortized

2) find(id):

Id:Long
Fetches the Id if it already exists from the TreeMap
Time Complexity:

1. Fetching an Id from the TreeMap : O(LogN)
   Fetching the Price from the associated Product Object to Id: O(1)

3) delete(id):

Id: Long
Fetching the the Id from the TreeMap: O(logN)
Summing up the values of Description : O(K)
Removing the Id associated with each description value in HashMap O(logK)

Time Complexity:

Total TC: O(NlogN) and O(K) if K>N K>N, if there is a description List for a product which might be greater than the number   of products in the whole Storage.

4) priceHike(l,h,r):

Traverse all the ids in HashMap or TreeMap and compare with l and h and increase the price by r%.

TC: O(N) in both HashMap / TreeMap case.

Time Complexity:

1) subMap operation on the items TreeMap to get ids in the range l to h - O(1)
2) Iterating and calculating the price hike of h-l elements in the map - O(h-l)*O(1) = O(h-l)
   So, worst case TC will be when h-l = N, i.e. O(N)

5) findMinPrice(n):

n = Long, where n = description of a product

Fetching the description value from HashMap in O(logN) worst case

Traversing through all the Ids mapped to this description value and fetching each Product associated with Ids. O(KlogN), where K = Number of Ids this description value is associated with.

Comparing with MinPrice see so Far: O(1)

Time Complexity:

The Total TC for this function would be O(logN)+O(KlogN)+O(1) , which would be O(NlogN), if K >=N.

6) findMaxPrice(n):

n = Long, where n = description value associated with some products

Fetching the description value from HashMap in O(logN) worst case

Traversing through all the Ids mapped to this description value and fetching each Product associated with Ids. O(KlogN), where K = Number of Ids this description value is associated with.

Comparing with MaxPrice see so Far: O(1)

Time Complexity:

The Total TC for this function would be O(logN)+O(KlogN)+O(1) , which would be O(NlogN), if K >=N.

7) findPriceRange(n, l, h):

n = Long, where n = description value associated with products

Fetching the IDs associated with the description involves get(n) operation which is O(1).

Iterating over prices with range l to h in the worst case is O(N) where N is the number of IDs associated with description n.

Time Complexity:

The Total TC for this function would be O(N) , where N is the number of IDs associated with the description n.

8) removeNames(id,list):

Fetch ID : O(1) HashMap and traverse the List and remove element if it is present in the description set.

Time Complexity:

O(1)+O(N) in for N elements in description list.

So worst case time complexity would be O(N).