STL APPLICATIONS

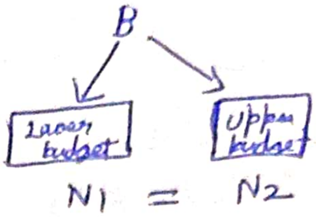
1. N items → $P_1, P_2, P_3 \ldots P_n$ ↗ price

Q queries → $B_1\ B_2, \ldots B_n$ (Budgets)

Given budget B_i, max. ^cost of amount of item we can buy ?

Exa: N → 5 4 2 1 6 3

Q → 2 5 3 10 7



2 → 1/2 → 1

5 → 1/2 → 3 (Take lower ~~oter~~ one to accommodate more items)

2/3 → 5

3 → 1/2 → 3

10 → 1 2 3 4 → 10

7 → 1 2 3 ✓ / 1 2 4

T.C : O(Q log n)
S.C : O(n)

① ~~Take~~ Try to take more items.

② If no. of items is same take the items with lower cost to save more budget.

Approach :-

1. Sort by ~~cost~~ price. ~~[2 3 5 7 10]~~

2. Create Prefix sum of price. ~~[2 5 10 17 27]~~

3. Use upper bound for a given budget ~~be~~ B, ~~find the~~ it gives me the index value.

```
int max Items = upper-bound (prefix_sum.begin (), prefix_sum.end(),
                 budget) - prefix_sum.begin ();
```

Exa :-

1) If budget = 10 : upper_bound (10) in [2, 5, 10, 17, 27] → pos^n 3 → can take 3 items

2) If budget = 16 : upper_bound (16) in ~~B~~ ⇒ pos^n 3 → can take 3 items.

2. Running of streams are there. What is the current mean of running stream return it?

Q1 → Add 1 → [1] → 1

Q2 → Add 2 → [1 2] → 1.5

Q3 → Add 3 → [1 2 3] → $\frac{\Sigma E}{n} = 2$

Q4 → At Mean → 2

Q5 → Add 2 → [1 2 2 3] → $\frac{8}{4} = 2$

Pseudocode :-

```
class Mean {
    int sum = 0;
    int cnt = 0;
    ~~int~~ void add (int num) {        1-2000
        sum += num;
        cnt++;
    }
    double mean () {
        if (cnt! = 0)
            return (double) sum/cnt;
    }
```

If removal of any number
↳ sum -= num
cnt --
& then calculate mean.

If asked for mode? Also removal of streams is possible?

Use a freq. map. and also keep track of max element which occurs most

1 → { 1 : 1 }   max = ~~1~~ 2

2 → { 1 : 1, 2 : 1 }   mode = ~~1~~ 2

2 → { 1 : 1, 2 : 2 }

Whenever element += 1 > max
↳ then update mode.

∴ mode = e
max → f[e].

* If remove 2, query is given. How to store overall mode to 1 back?

Do reverse mapping,

map < int, int >   (freq ← int, int → num)

Whatever int we have treated earlier as mode → last value present in stream we can try to keep.

3. LRU Cache

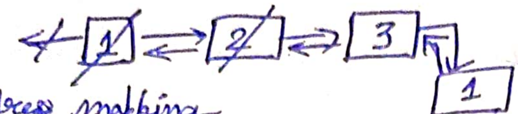
Queries: Put ~~→~~ 1 : 2

Put → 2 : 3

Put → 3 : 1

get → 1 → 2

Put → 4 : 5



New LL

⇒ ←3 ⇄ 1 ⇄ 4→

1 is the least ~~recentely~~ recently used so remove from front & add it to last

Address mapping

{ 1 : (add(1))

~~2 : (add(2))~~

3 : (add(3))

4 : (add(4)) }

Hashmap

{ 1 : 2

~~2 : 3~~

3 : 1

4 : 5 }

① Hash Map → store the mapping of key → node

② Doubly LL is used to maintain the order of :

- Most recently used node should be at the front.
- Least recently used node should be at the end.

40 → 1.

(30 → F)

Operations :-

1) get (key)

- If key exists :
  → Move the node to the front of the list.
  → Return value
- Else return -1.

2) put (key, value)

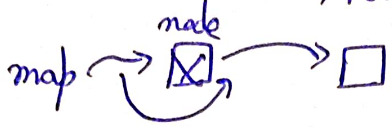
- If key exists
  → Update value and move node to front.
- Else
  → Create new node and add to front.
  → Add key → node mapping to hash map.
  → If capacity exceeded :
    - Remove the tail node (least recently used)
    - Delete from hash map.

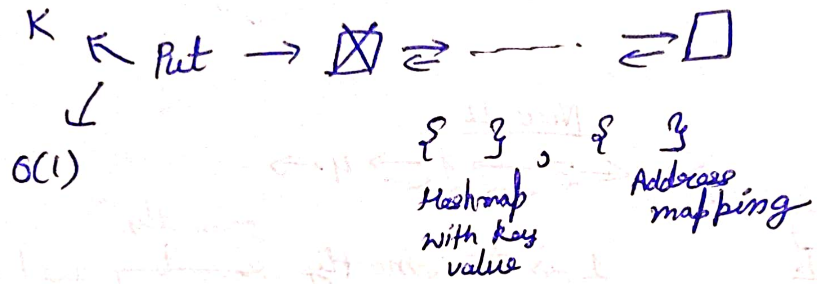
O(1) get → map lookup

~~Put~~ → LL → removal

↳ Pt~~rf~~ removal / add^n (K unit operations)

O(1)




K
Put
{ } , { }
Hashmap with key value
Address mapping
O(1)