Shopping maximum number of items

N - items with prices → $A_1, A_2, \ldots, A_N$ $\quad N \le 10^5$;
& $\quad M \le 10^5$
M - Budgets → $B_1, B_2, \ldots, B_M$

We need to find the maximum number of items we can buy for each budget.

Sol

↓ Sort this → $A_1, A_2, \ldots A_N$

Bruteforce way → Linear scan i.e. For every B_i we do $B_i - \sum_{k=1}^{j} A_k > 0$ (just before this >0 is met) ; j is number of items we can buy w/ B_i

TC → $O(N \times M)$

Optimized Soln → Prefix sum array

Create a prefix sum array of $A_1, A_2, \ldots, A_N$
So,

$$P = \{p_i\} \rightarrow p_i = \sum_{j=1}^{i} A_j$$

Eg: $A = \{2, 3, 5, 10\}$
$P = \{2, 5, 10, 20\}$
& $B = 7$
So, no. of items we can buy is = index of the upper bound in P for val = B = 7
↳ index of (5) + 1

→ Basically we want to find no. of elements ≤ 7

Finding TC → $A \xrightarrow{\text{Sort}} A' \xrightarrow[\text{sum}]{\text{Prefix}} P$; for M B_i's upper bound check

↳ $O(N \log N + N + M \log N)$
$O((M+N) \log N)$

upper_bound (a.begin(), a.end(), x) − a.begin() = No. of element ≤ x in a

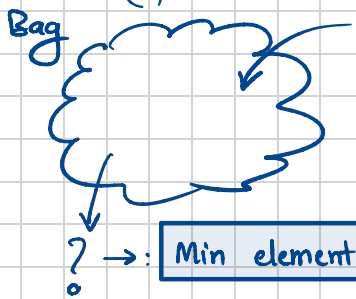
lower_bound (a.begin(), a.end(), x) − a.begin() = No. of elements < x in a

TIP

```
signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    //int _t;cin>>_t;while(_t--)
    solve();
```

Adding above lines optimizes i/p & o/p which can solve your TLE errors.

Designing Some Data Str



b.cpp

```
#include<bits/stdc++.h>
using namespace std;

struct bag{
    int minVal;
    bag(){
        minVal = INT_MAX;
    }
    void insert(int x){
        minval = min(minVal,x);
    }
    int getMin(){
        return minVal;
    }
};

void solve(){
```

Implementation

for implementing maxVal

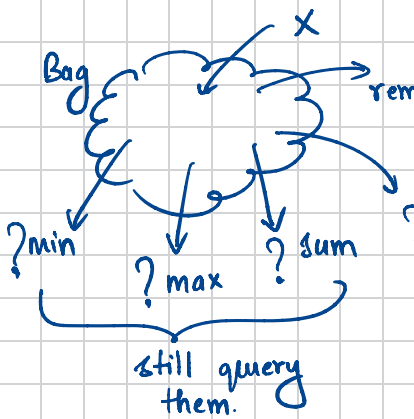
↓ initialize with INT_MIN

&

↓ inside insert() add maxVal = max(maxVal, x)

Takeaway: So for inserting we can create a normal struct.

↳ Can also implement sum as well



? No of distinct elements in bag

⇨ MAP → Pairs in sorted order

How it's created?

↳ map<int, int>

{(2,2), (5,7), (7,3)}

(value, occurence)

Working

Insertion → O(logn)

x inserted

⇒ mp[x]++;

Finding min element → O(1)

*(mp.begin()).first

OR

mp.begin() -> first

Finding max element → O(1)

(mp.end() - 1) -> first

OR

mp.rbegin() -> first

You should check whether map is empty or not

For sum → O(1)

↳ We can have a seperate variable as we did earlier to keep track of sum.

For removing → O(logn)

↳ Let say we want to remove x

↓

Check if x present or not

↓ If yes

mp[x]--; sum -= x;

If mp[x] == 0;

mp.erase(x);

For number of distinct elements in the bag → O(1)

→ mp.size() → gives no. of pairs

Code for this is present as: designing_data_str1.cpp

Let us try to get one more operation

For given X → find the freq of the smallest number greater than X (equal to) in the map → O(log n)

Find iterator using lower bound

```
auto it = mp.lower_bound(x);
if (it == mp.end()) return 0; // If element not present.
return it -> second
```

Cond Problem → Exchange argument

Two players

So we have N-cards.

$\begin{pmatrix} A_1 \\ B_1 \end{pmatrix}, \begin{pmatrix} A_2 \\ B_2 \end{pmatrix}, \ldots \begin{pmatrix} A_N \\ B_N \end{pmatrix}$

card 1 card 2

P-1 → takes a card → score1 += A_i

P-2 → takes a card → score2 += B_i

Basically if you are P-1 → your strategy is = max (score1 − score2)

Example: To see what's happening.

You are P-1, two cards are → $\begin{pmatrix} 2 \\ 4 \end{pmatrix}$ ① $\begin{pmatrix} 3 \\ 1 \end{pmatrix}$ ②

which card will you choose

	①	②
P-1	2	3
P-2	1	4

→ ① → max (score1 − score2) = 1

→ ② → —"— = −1

I spent a lot of time on this but there's a better way to approach it

$\binom{a}{b}$ ① $\binom{c}{d}$ ②

if P-1 chooses ① → (a-d)
if P-1 chooses ② → (c-b)

Say we know it's better for P-1 to take ①

⇒ a-d > c-b

⇒ (a+b) > (c+d)

↓

sum of numbers A_i & B_i on cards

So finally strategy would be ↴

> Sort cards in descending order based on the sum of A_i & B_i

↳ This is popularly known as **EXCHANGE ARGUMENTS**

Desining data str 2

sum of top k-elements

basically sum of the top max elements present.

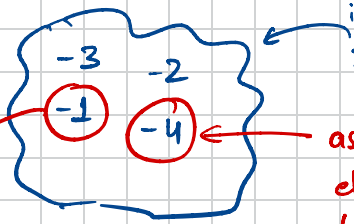
⇨ Priority Queue → they implement max heaps

One Way : Bruteforce

↳ iterate over top k elements to get sum → O(k)

Optimized way

↳ So in priority queue we only maintain k-elements & maintain a sumVal variable



inserted one by one but -ve
3, 2, 1, 4

as -4 is inserted there are k+1 elements

Let k=3 here

remove the element from pq using pop()

so

removes max element → ∵ -1 is max there basically +1 was min

After removing we have to update sumVal respectively & return accordingly.