Fall 23 Div II Week 2 - Solution Sketches

(taken from editorials of original problems)

Problem A

(Source: Original Problem)

Note that the list of courses is sorted, so to solve this problem efficiently, you can implement a binary search algorithm. For each query, use binary search to determine if there is a course with a matching number. Initialize two pointers, one at the beginning and the other at the end of the sorted list. In each iteration of the binary search, calculate the middle index and compare the course number at that index with the query. If they match, output "COURSE" and move on to the next query. If the course number is greater than the query, update the right pointer to search in the left half of the list, or update the left pointer to search in the right half if the course number is smaller. Repeat this process until you find a matching course number or exhaust the search space. If no match is found, output "NO COURSE FOR YOU" for that query.

Problem B

(Source: Codeforces Round 367 (Div. 2) Problem B)
Again, to solve this problem, you can use binary search, but note that the binary search method you have to implement to solve this problem is a bit different from the previous one. In this problem you are not looking for an element in an array, but rather what is the largest index whose value is less than or equal to the target value.

First, sort the prices in ascending order, then, for each of the q days, perform a binary search on the sorted price list. Initialize two pointers, left and right, to the beginning and end of the list, respectively. In each iteration, calculate the middle index as mid and compare the price at mid with mi. If the price is less than or equal to mid, update the left pointer to mid to search the right half; if it's greater, update the right pointer to mid - 1 to search the left half. Keep track of the count of shops where Vasiliy can afford the drink. After the binary search, the left pointer will indicate the number of shops where Vasiliy can buy a bottle of "Beecola" on the i-th day. Output this count for each of the q days.

Here is an example solution written in Python:

```python
n = int(input())
x = list(map(int, input().split()))
x.sort()
q = int(input())

for _ in range(q):
  m = int(input())
  lo, hi = 0, n
  while lo < hi:
    mid = (lo + hi) // 2
    if mid == lo:
      mid += 1
    if x[mid - 1] <= m:
      lo = mid
    else:
      hi = mid - 1
  print(lo)
```

Problem C

(Source: Codeforces Round 403 (Div. 2, based on Technocup 2017 Finals) Problem B)

To solve this problem, we employ a binary search strategy to determine the minimum time required for all friends to convene at a common point along the road. The core concept is to pinpoint the smallest time interval (t) in which it is feasible for all friends to converge.

Suppose we want to determine, for a given y, whether the friends can meet within time y. We calculate the range of positions each friend can reach in that time frame, expressed as [xi - y * vi, xi + y * vi]. Subsequently, we scrutinize these position ranges to determine if there is a common point of intersection. To do this, we calculate the maximum left boundary (L) and the minimum right boundary (R) among all friends' ranges. If L is less than or equal to R, it implies a non-empty intersection exists, signifying that all friends can gather within t seconds.

Now we can turn this into a binary search method as follows (suppose we are keeping two variables low and high to keep track of the interval of possible values of the answer t):

- If the friends can meet within time y, we narrow down the search by updating high to y, since we know that they can meet within any time greater than y.
- Otherwise, we update low to y.

This iterative process continues until low and high are very close, with their absolute difference smaller than 10^-6, at which point t represents the minimum time required for all friends to meet. One trick to simplify this implementation is to not try to keep doing binary search updates until the difference is really small, but actually to do a set number (like 70) of binary search steps. This avoids getting stuck in infinite loops when precision issues cause the difference to never be achieved.

Problem D

(Source: 2017-2018 ACM-ICPC Nordic Collegiate Programming Contest (NCPC 2017))
Again the trick here is to binary search over the answer (i.e. the maximum speed of the slowest kayak). To check if we can assign participants to kayaks such that the slowest kayak has speed at least v, greedily assigning people to kayaks. Doing this is a little tricky, so here is a short description on how to do it: