Analysis

The first important observation is that for a chosen subset of people who will sit, the optimal configuration is when the empty seats are consecutive and the 2 people who have a neighboring empty seat to be the 2 people with the largest B. The configuration of the others doesn't matter.

(P, P, P, maxB1, __, __, __, __, maxB2, P, P, P ; where P=passenger, __=empty seat, maxB1/2 = the passenger with the first/second maximal B)

I'll be referring to the consecutive empty seats as "the gap".

1. **Partial solution *O(N*N*L*)**
   Let's fix the two passengers who we will put next to "the gap" and what will be the size of "the gap". Then we only need the passengers with maximum A to fill the rest of the seats. In order to do that fast, we can sort the passengers by their A, and after *O(N)* precompute we can have the sum of the first K passengers in O(1)
   The other solutions also require sorting the array of passengers in decreasing order of their A, so from now on I'll consider it done.

2. **Partial solution *O(N*N)***
   Let's fix the number of passengers who will sit, and let it be *K*. Then there are 3 possible configurations:
   A) We can choose the *K* passengers with the biggest A and place the 2 with maximal B in the 2 ends of "the gap".
   B) We can choose the *(K-1)* passengers with maximal A and from the others choose the passenger, who increases the sum as much as possible (he has a maximal A + B*(*L-K*)). Then in the 2 ends of "the gap" we will place the passenger with maximal B from the first (K-1) and the passenger with the maximal A+B*(*L-K*) from the others.
   C) We can choose the (*K-2*) passengers with maximal A and choose the 2 passengers from the others who will increase the sum as much as possible (again they must have maximal A+B*(*L-K*))

   The passengers who will be neighboring "the gap" in each of these cases, we can find by iterating the whole array of passengers.

3. **Full solution *O(N*log(N) + N*20)***
   Again, as in 2. we will be iterating the values K- the number of passengers seated. But now we want to find the passengers who will be sitting next to "the gap" faster. We can find the sum of the other (*K-2*) passengers' A in O(1), because they will be a subset of the first K (sorted in decreasing order of their A). Then if we know the sum of the A's of the first K passengers and we know exactly who will sit next to "the gap", we can find the sum of the A's who won't sit next to "the gap". To do that in O(1) we will precompute the sum of the A's of the first I passengers for every valid I.

The cases are the same as in 2. but we will be finding the people next to "the gap" faster.

A) Here we need the passengers with maximal B from the first K passengers. We can precompute that beforehand in *O(N)*.

In B) and C) we search for passengers who have maximal A+B*(*L-K*). We can notice that this looks like a line equation, so we will use Upper/Lower Envelope(also known as Convex Hull Trick)

Upper/Lower Envelope does the following:

Consider N lines, each defined by an equation $y = B[i]*x + A[i]$ (the angular coefficient is B[i], so that the variable's names are the same as in the original problem). We want to find a list of lines, so that for some X, the maximal Y is on one of those lines. One can prove that for every line, the X-es on which it has the maximal Y are on a segment [p[i], q[i]]. We also want this list to be ordered in increasing order of these segments. (For lines *I* and *J*, $I < J \Leftrightarrow q[i] \leq p[j]$)

-Building an Envelope in *O(N)* having the lines sorted by their A's:

Consider the lines in decreasing order of their A. We will only consider the lines for positive values of *X*. It is clear, that the first line will be in the Envelope and that it will be the first line in it. We will iterate over the lines and we will be adding them to the end of the Envelope and sometimes we will remove the lines in the end of the Envelope.

Let the last 2 lines on the Envelope be *L1* and *L2*, and the current line we want to add is *P*. Let *X(A,B)* is a function, which returns the value of the *x*-coordinate of the intersection of lines A and B. If *X(L1, P) < X(L1, L2)* we will remove *L2* from the Envelope. That is because by this inequality we notice that *L2* will never be the line with a maximal *Y*; lines *L1* and *P* are above it on every *X*. We should do that operation on the last 2 lines in the Envelope as long as that inequality holds. In the end we add *P* at the end of the Envelope.

Now, having this Envelope, we can answer the query "What's the maximal *Y* of some line on some specific *X*" fast enough.

Let's go back to the original task.

In B) we needed the passenger with maximal A+B*(*L-K*). Here (*L-K*) is the X-coordinate in the geometric representation. If we iterate over *K* in decreasing order and before every query on the Envelope we add the *K*-th passenger, we will have the Envelope built on the last (*N-K*) passengers. Now we can use binary search to find the line(passenger) with maximal *Y* on X=(*L-K*)

For C) the thing get a little more complicated, because we need the maximal 2 *Y*-s on a fixed *X*. In order to find those values, we will keep 2 Envelopes at the same time and we will add every passenger to a random one. So for the 2 passengers in the ends of "the gap" we will take the maximal lines in the 2 Envelopes. The chance of not finding the optimal passengers is 1:2 (Only if they are put in the same Envelope). If we execute that algorithm *P* times, the chance of not getting the optimal passengers is (½)^P. For P = 20, this chance is small enough.

About the queries on the Envelope – if we know the maximal line for some *X*, the next query will be for *X+1*. So it is not necessary to do a binary search once again, we can simply start from the last maximal line and start "going up" (go in that direction, where the Y increases). In the end we will go through every line once => *O(N).* And so executing this algorithm 20 times we get complexity *O(20*N).*

Author: Viktor Terziev