00:00:12 Hi, my name is Ashwatha. I'm a Software Engineer at Google. I am a Hands-On software engineer. Involving coding, mainly Java and C++. I also mentor other engineers and I am involved in design and architecture decisions as well. Hi, I'm Swati. I'm a software engineer at Google. In today's video, we will talk and demonstrate of how a Google engineer approaches an engineering problem. So, I'll be giving a problem to Ashwatha and he'll talk us through the process of solving it and towards the end, we will also look at the positive aspects

00:00:46 that he demonstrated through the interview process. So Ashwatha, the question that I'm going to give you today is this. I have a collection of numbers, and you have to take this collection and find pairs which sum up to the number that I give you. - OK. - For example the collection of numbers is 1, 2, 3, 9 and you have to find the target 10. - OK. - The pair of numbers which adds up to 10. - OK. - And you could also have a target like 8. This does not really exist with it. I have shared this problem as well in the Google Doc.

00:01:15 - OK. - And then you can go find it. OK. Let me ask some clarifying questions and we'll make some notes as we go. So, in the second example that you gave me where the target was 8 we don't have a pair of numbers in this array that will add to 8. On the other hand, the first example is possible so both of those can happen in this case. OK. So how would these numbers be given to me? For example, will this fit into memory? Is this an array? How does this work? Yes, it would be memory, and you can consider an array.

00:01:45 OK. OK. How do we handle duplicate elements here? For example, if the sum is 8 and there were to be a 4 in the array. Can I just add 4 to itself and make 8? You wouldn't have a number repeating itself as the same index. However, there could be repeated numbers in the array. OK, so there might be two 4's in the array but I wouldn't be able to reuse a 4 that already existed earlier. OK. Can I assume that these are all integers? For example, can there be floating point numbers here? Can there be negative numbers?

00:02:18 There could be negative numbers and integers only. No floating points numbers. OK. That sounds good. Thank you. So, when I think about this, the simplest possible solution here is just brute force. We can use two indexes in the array. Let's call them I and J. The I index would iterate through the whole array and the J index would start at I+1 and then iterate through the rest of the array. That way we avoid any duplicates as we discussed and for each pair of elements, we would add them up and we will see if they add up to the target number that you gave me.

00:02:50 If they do, then we have found a pair. On the other hand, if they don't and we iterate though the whole array for both indexes we will know that there is no possible solution for this. It's not very efficient but it should work. What's the complexity of the solution? This would be quadratic. - Could we do better? - OK. Let's think about it. Better than quadratic. The elements are sorted, so we should be able to use that. As we iterate through the array for example, the first element is 1 here and let's say we are looking for a sum of 8.

00:03:20 We can think of the compliment of 1, which is 7. So we can search for 7 in this array. Since it's already sorted, we can do binary search. We go to the middle of the array if the element is bigger than 7, we go left. Otherwise, we go right. So over flogged in complexity for the search itself and we'll repeat it for every element in the array. So, that would be of N log N and for the whole solution. OK. This is still not the optimal solution. You could just tell going linear. - Could you think of something better? - OK.

00:03:48 Let's think about that. I'm trying to see if we can bound this problem in some way. For example, in this example array the last two elements would give me the biggest possible sum since the array is sorted. So that would be 3+9 or 12 the first two elements would give me the smallest sum because again, those are the two smallest elements in the array and everything in between is my range of values. So if I started with two indexes one pointing to the beginning of the array and pointing to the end of the array.

00:04:19 Let's say we add them, so in this case 1+9 is 10. That's bigger than the sum that we are looking for so we would move the right index towards the left to try and see if we can make the sum smaller. If that's not the case and if my initial sum is bigger than the target we would move our left index towards the right to see if we can make our sum bigger. And we would stop whenever we find a matching pair or whenever the two indexes essentially cross over each other at which point, we know that there is no solution.

00:04:47 That could be a linear solution, I think. OK. Would you like to write down the problem? Sure, we can try to code this up. - What's your language? - Is C++ OK? - OK. Sure. - Good. Sounds good. So, I realized we haven't discussed what to return in this case. Should we just return a Boolean indicating if there is a solution or not? Or do we want to return the matching pair itself? Do you see a problem in returning the matching pair? Not really, it should be fairly easy. If you find a pair that matches, we just return the pair.

00:05:19 If we don't, we would have to return a Boolean. So we can think of creating some data structure that has a Boolean field indicating whether we found a matching pair or not and then, a matching way. It's not very elegant, but it's doable. OK. Returning a Boolean would be fine. OK. So let's do it. I'll start coding. OK. Let's call this as... PairWithSum and the input is a vector, OK? For the array. - Yes, vector is OK. - OK. I'm going to call that Data in here and then, there is an integer, which is the sum that you're looking for.

00:06:00 OK. So, I'm going to have two indexes. Let's call one of them low and that starts off as 0 and there is another one. Let's call that High, which starts off at the right end of the array. OK, so what I'm going to do is as long as Low is less than High and that in fact, that also takes care of the situation where we start with an empty vector because that would mean is Low would be 0 and High would be -1 so we would never really enter this loop so we don't have to worry about that edge case here.

00:06:35 OK. So, while Low is less than High we add up the elements at those two indexes so we basically check if data of low +data of high is equal to the sum that you gave me and if it is, we can just return true right here. If not, we keep going. So we do need to adjust the Low and High here. If the sum of these two elements is bigger than the target that you gave me we will have to decrement high. Otherwise, we'll have to increment Low. So, that's something that we'll have to do to make this easier.

00:07:11 I'm just going to declare a separate variable that stores that sum so that I don't have to recalculate every time. OK. So, if that sum is less than the target that you gave me the sum is less than the target then, we need to try and make it bigger. So, I'm going to increment my Low. And by the time the Y loop finishes if we still haven't found a matching pair we return false distribute. Excellent. - Now, let me throw a wrench into the mix. - OK. Let's assume that the collection is not sorted. - OK. - How would you go about the next?

00:07:58 OK, so if the collection is not sorted, of course this will not work but maybe we can just sort it. That's of N login. We can sort it initially and then just do the same thing. That will work. It wouldn't be linear anymore, but it will still be of N login. Are we looking for anything faster? Yeah, we could do it faster than N log N time. OK. Let's think about that. So, let's go back to the initial idea that we had of iterating through the array picking an element and then looking for its compliment.

00:08:30 If it was sorted, we could have used binary search but that's not possible here since the array is no longer sorted we can do a linear search but that will make the solution quadratic again. Let's see if we can do something better than that. OK. So, we'll stick with the compliments idea. So, I'm going to write down an example array here. Let's still use 1,2,3,9 and we'll add one more. We'll call it 22. OK. Let's say we are looking for a sum of 10 here. I start iterating through the array. The initial element is 1.

00:09:01 Now I'm looking for a 7. If there is a way, I know that I have seen a 7 before then, I'll be able to do it. So, in other words, I think what we are looking for is a data structure where we can easily look up elements or their compliments that we have seen before and then go from there. And I think that would again be linear. So, we should be able to do it linearly. A hash table might be a good data structure for this. OK. What would be the key of the hash table? That's an interesting question. I guess we don't really need a hash table.

00:09:33 We just want to keep track of what values we have seen here. Not necessarily map keys to values. So, we might get away just with a hash set. So, in C++, I think that's an unordered set as we are scanning through the values for each value with, we see if we have seen compliment before from our set. If we haven't, we compute its complement and we add it to the set and then we can keep going. It's a little tricky because, as we discussed earlier we don't want to repeat elements from the same index

00:10:02 so we should take care that we don't add values to the set before we have already checked if its compliment was in there. But that should be workable and it would still be a linear solution for this. OK. How about the collection has million numbers? So, if we have millions of numbers, would they still fir in the memory? Probably not at this point. OK. So that would make it harder. So if we have a very large set of values, maybe we can still consider ranges of values if the range of these values is bounded in some sense

00:10:30 we can take separate sets of values as to break up the whole range into subranges and we can use different computers to compute this kind of hash set of compliments for different ranges of values. So, when we scan through the full set of values which still need to know which computer to check in to find the compliment. So it's tricky, but it's absolutely doable. OK. Do you want to code it at this point? Sure, so in C++ for a hash set we'll be using an unordered set of integers in this case, it should be called compliments.

00:11:05 I'm going to call it Comp because I don't want to type this every time. OK. So, what we'll do is we'll iterate through the values in the input vector. In each case, we will check if we have already seen its compliment so we'll say... The syntax might be slightly off, but the basic idea should be here. So, we're inserting compliments. So, we'll just check if in my set of compliments we already have this value. And if we do, we can just return true. If not, we will compute its compliment and essentially add it to our set.

00:11:56 And I think it's just comp.add. That should be it, if in case, we don't find any matches we will return a false idea. So that should work. The code itself is fairly simple but let me trace it on an example to make it sure that it works. So let's use this example array: 1,2,3,4,9 and 22 and let's say our target is 10. So, initially, when we start this code, when we start the for Loop that I'm highlighting here obviously the unordered set is empty to start with. We look at the element 1, we try to find it in our set.

00:12:37 We don't find it, so we add its compliment to the set. So, it will have 9 in it at that point and then we look at the element 2. We see if that's in the compliment set. It's not, so we add 2 compliment as well, which is 8. We'll do the same thing with 3's compliment and 4's compliment as well. Right? And then we get to the element 9. We check it in our set. It's already here. At that point we have found a matching pair so we are done. At this point, we'll return true. If we are looking for a different target, like 8.

00:13:13 We would never find its compliment in our set so we'll essentially iterate through the whole array and will return false. So yes, I think this will work. - OK. Great job. Thank you. - Thank you. Alright, so just to recap the interview that we just saw there were a couple of things that you should be aware of while interviewing. One is to make sure that you understand the problem right and ask clarifying questions in order to do so. Ask for the question to be repeated if it helps and also to be written down verbatim so that the question is clear.

00:13:47 One of the question that Ashwatha asked was about the floating-point integers and that does affect the problem. So, yes, it's a good one to be asked. The other thing that you could do is to always think out loud the solution. This helps the interviewer understand your thought process and also course correct when required. It's also good to ask questions which demonstrate your expertise and also helps to come closer to the solution faster. It's always good to brainstorm with your interviewer. After all, two mindset a problem is always better.

00:14:18 One thing he did really well was to come up with a simpler solution and articulate it. The first solution is not going to be the best and it's not the best for anyone. The interviewer will challenge you to do better, be more efficient and the idea is to get to the right solution and then coded. Another great thing that he did really well and I would encourage is to test the solution. You can test it with some examples that the interviewer has already given or come up with your own examples and test out the solution.

00:14:49 Another good idea is to always test with the edge cases. Like, in this case, we took the example of empty collections and tested it as an edge case. That gives you the confidence that your solution works for all different types of examples. Just like we saw in this interview, do the trade-off analysis of your choices proactively. The interview is time-bound, so manage your time well as you navigate the solution. Write clean and structured code, do not over complicate it. Leverage the latest features of the technical language

00:15:19 for your advantage and stick to one language only throughout the code. Best is to do all of these proactively.