60 mins total divided as follows:

* 5 min intro (brief discussion of the resume)
* 10 min of 3 algorithmic questions (no coding)
  + 2 time complexity questions
  + 1 question where I was given an algorithmic problem and a solution to said problem and was asked if the solution would do as intended.
* 45 mins of as many coding problems as you can solve
  + 1 easy-medium hashmap question
  + 1 medium DFS question
  + 1 medium hashmap + sliding window question building upon the first question

My biggest tip would be to share your thought process with the interviewer and to be wary of the time limit.

393 UTF-8 Validation 39.1% Medium  
539 Minimum Time Difference 53.9% Medium  
811 Subdomain Visit Count 73.9% Medium  
1319 Number of Operations to Make Network Connected 57.2% Medium  
723 Candy Crush 74.9% Medium  
273 Integer to English Words 29.5% Hard  
694 Number of Distinct Islands 59.5% Medium  
695 Max Area of Island 69.0% Medium  
72 Edit Distance 50.2% Hard  
733 Flood Fill 57.7% Easy  
718 Maximum Length of Repeated Subarray 51.3% Medium  
721 Accounts Merge 55.5% Medium  
56 Merge Intervals 44.5% Medium  
200 Number of Islands 53.2% Medium  
23 Merge k Sorted Lists 46.6% Hard  
236 Lowest Common Ancestor of a Binary Tree 54.5% Medium

393 UTF-8 Validation  
539 Minimum Time Difference  
56 Merge Intervals  
273 Integer to English Words  
146 LRU Cache  
200 Number of Islands  
1319 Number of Operations to Make Network Connected  
811 Subdomain Visit Count  
733 Flood Fill  
723 Candy Crush  
220 Contains Duplicate III  
694 Number of Distinct Islands  
695 Max Area of Island  
72 Edit Distance  
721 Accounts Merge  
718 Maximum Length of Repeated Subarray  
236 Lowest Common Ancestor of a Binary Tree  
23 Merge k Sorted Lists  
219 Contains Duplicate II  
136 Single Number  
217 Contains Duplicate  
303 Range Sum Query - Immutable  
325 Maximum Size Subarray Sum Equals k  
1232 Check If It Is a Straight Line

**Trapping rain water**

**Provide an algorithm to show there exists/does not exist a triple I, j, k, such that arr[i] + arr[j] + arr[k] == 0 (in an array).**

A couple of corrections to your question: The answer to this question does not have the properties of NP Completeness as long as K is a given constant.

For example, using the naiive way to compute the result, the complexity of the solution is polynomial in the size of the input since Choice(N, K) = O(N^K). If and only if K can be a function of N does the worst case complexity increase to O(N^N). In all other cases, this problem can be solved in polynomial time.

---

Okay. Now to the meat of your question: why is this question valuable?

The question is valuable because of 3 things:

1. **Coding Skills**  
   Right off the bat, it allows the interviewer to assess if you have been able to detect the naiive solution of generating Choice(N, K) tuples and summing them up. They then have the option of asking you to implement it and it should be a short function: spot check of coding skills especially the ability to generalize your algorithm from the case of 3 numbers to K numbers. This allows the interviewers to ask you the computational complexity of the solution and it provides insight into your coding abilities and CS foundations.
2. **Data Structure skills, Implementation detail awareness**  
   Once your coding skills are out of the way, then it's important to judge your choice of data structures and algorithms for optimization: there are many different data structures that are suitable for application to this problem - Hash Tables, B-Trees, Sorted Arrays and corresponding algorithms - Binary Search, Element Insertion, Hashing Algorithms etc. There are also considerations of implementation such as cases of *overflow* and *underflow* that come about due to maintaining sums and differences. Subsequent questions will probably flow around the ability of the candidate to choose reasonable data structures, justify the choice and provide adequate implementation details so that the computational complexity of the solution is better than the naiive solution.
3. **Ability to optimize data dependently**  
   The richness of this question also allows for a different type of optimization: data dependent pruning. In point 2, a good candidate will probably be able to generate an efficient general solution. Further optimization will involve taking advantage of the data distribution present in the input - for example, not considering any combination of numbers whose sum is greater than the largest valid combination of negative values or individual numbers smaller than the smallest possible negative value etc which helps prune down the search space (though in a manner that is input dependent). This is basically ad-hoc analysis and it allows the interviewer to judge the smarts of the person.

Every candidate will go through these steps at a different pace and given the limited time nature of interviews, the degree to which the candidate was able to progress on these 3 issues gives insight into the abilities of the candidate across a number of dimensions and that's why this is a valuable question for tech interviews not just at Palantir but at many other tech companies. The fact that the ultimate solution has a very high polynomial complexity is irrelevant as the objective of the interview is to determine a candidate's analytical and professional skills and not the practicality of implementation of a particular solution to this problem.

I interviewed for new grad, completed all 3 trivia questions + 2.5 questions, describing approach for last question but ran out of time for complete coding solution.

Scheduled a redo that got cancelled by recruiter informing me that I passed.

I urge everyone to schedule a redo, because if you pass, the recruiter cancels it, if you fail the first one, you have a second try.

Lately, I did my phone screen interview with a Karat engineer.

Interview is over call over Karat website and coding is on the Karat Studio where there is an option to select your programming language, auto-complete feature and nice colored editor.  
Also, you run your code **as many times as** you want and actually show the correctness of your code by printing to console.

Interviewer initially introduces itself and talks about the process you are going through. Overall, Karat Interview structure is:

1. Your background (1-2 mins)
2. Simple questions about complexity analysis and data structures. (10 mins)
3. Technical algorithm question/s. The number of questions you get depend on your performance. I got two questions, last one being an extension of the first. (45 mins)

Questions were medium LC level graph questions.  
Completeness is **more** important than optimality.

There is an option to reschedule your Karat interview within 24 hours after your interview **without** a punishment if things go wrong.

Response time of Palantir is very short. (resulted within a day)

**PERSONAL EXPERIENCE WITH KARAT:**

* **In command line, go to a random byte location in a file, and parse forwards and backwards to check for “\n” character, extracts the contents between the newline characters. Does this method always work? (for first line, no newline character in the front, for last line, no newline character in the back)**
* **Given a function that takes int input num, and divides num by power of ten to extract the digit, then mods the num to 10 to extract the remainder, and sets the value of the remainder back to variable num, etc. in a while loop, this function checks if the digit extracted is 7. What is the time complexity of this function? Can the time complexity be given as a function of num itself? (ans: function has a runtime of O(log10(num)).**

**I guess another method would be to convert num to a string, and then convert each digit character of string into a int digit, and check if digit == 7 from there.**

* **Given two sorted array inputs sorted1 and sorted2, two functions are implemented to check if there are elements in sorted1 that are not in sorted2.**

1. **Store the elements in sorted2 into a hashmap, then for each element in sorted1, check if element in sorted2.**
2. **Store elements in sorted2 into a binary search tree. For each element in sorted1, check if element in sorted2 with a binary search.**

**Q: What are the runtimes for each algorithm? Which runtime is better? Under which cases does the overall worse algorithm perform faster than the overall better algorithm?**

**Code 1: Given a family pedigree (not necessarily a complete pedigree, just a fragment, may have multiple “roots” with zero parents, but pedigree is connected), return the individuals (expressed as int values) which have zero parents, and return the individuals which have only one parent.**

**Code 2: Given a family pedigree, return the most recent common ancestor of two distinct individuals (family pedigree may have multiple “roots” with zero parents, but pedigree is connected)**

**Code 3: Given a family pedigree, and two individuals, determine if one individual is descended from another.**

1. OA 2. Outsourced Coding interview 3. Phone screen with HR manager. I failed in the 3rd one. I needed to research the company more to appeal to my passion for the company.

Technical interview includes abstract problem solving: Highway and toll stations, break it down to how to optimize the max toll each station can collect