**Second Round:**  
Two questions were asked.  
1) Given two arrays were digits of one array represent a number,maxmise the number by replacing it with elements of second array.  
eg:  
arr={3,1,4,5,6}  
rep={1,9,5,2,3}

after replacement  
arr={9,4,4,5,6}  
one digit of rep can be used to replace only once.

2)convert the leaves of a binary tree to a linklist using its right pointer.(in place)

I had solved both and was selecetd to the next round along with 29 others

**Round three**  
Find all substrings of a given string.  
space can also be part of the substring

I had topped this round giving the most efficient solution.

**Round four: (Face to face)**  
All contestant were asked the same question.

1)http://www.geeksforgeeks.org/search-in-row-wise-and-column-wise-sorted-matrix/

i gave the answer but was asked to justify my approach and was also asked to code in another approach.There were several questions asked to know the various approaches it can be done in.

2)find sum pair in a sorted array.  
Again, i was asked questions regarding my approach

**Round 5**  
Three of us were taken to a room. we were asked to code a DFA which checks the correctness of grammar of a sentence in C. The mentor was very helpful and friendly.

10 SDET Interview Tips: 1. Think about how you would test mundane items, like calculators or staplers. Be sure to classify your tests (i.e., border cases, common functionality, error checking, etc.) 2. Practice your C/C#/C++ (web programming is more C#, systems more C/C++) understanding: if you have to ask basic language questions during the interview, do so confidently. 3. Review basic programming questions, such as how to turn a recursive solution into a stack-based one, how to reverse a singly linked list, how to traverse a tree (pre-, in-, and post-order), etc. 4. Make sure you understand the trade-offs between data structures, focusing on space (memory) and time (computation) as your major resources being gained or lost. 5. Know how to compute the running time (in Big-O notation) of any functions you see or write. 6. When writing code, don’t forget to include basic error-checking! You won’t be reminded by the sneakier interviewers. 7. Know what a buffer overflow is and how/why it is bad to have in your code. Have similar basic understanding of other common vulnerabilities (e.g. for web devs, know what XSS attacks are). Corollary: common programming mistakes, such as dangling pointers, malformed “while” loops, or double memory de-allocations, should also be clear concepts 8. Don’t try to optimize your code before it is finished. Get a working solution, and THEN go back and improve it. Make sure the interviewer is following your process. – I personally feel you shouldn’t optimize period on interview questions unless explicitly asked. Optimization rarely results in a clear algorithm that you’ll be able to explain within the timeframe of an interview. Obviously, don’t solve problems with incredibly naïve/stupid solutions either. 9. Keep the interviewer in the loop when you write, but don’t talk too much! You can run the risk of looking like you don’t know what you’re doing if you stall or ask too many questions. 10. That said, it’s *good* to ask clarification questions about the programming assignment you’ve been given. To any parsing question you might ask, “ANSI or Unicode?”. It is common to get a question on parsing, so review your string functions!