Palindromes :

1. Check if a given string s is palindrome

Note that the check s.equals(newStringBuilder(s).reverse()) is inefficient for large strings.

Better soln : Take 2 pointers star and end, keep incrementing and decrementing if chars match.

Follow up : Can you create a boolean 2d palindrome table to answer in constant time if any subsrting(i,j) is palindrome. Start by palindromes of length 1 to n, since check for larger palindrome will use checks for smaller length palindromes. Recursion is not a good idea.

1. Given a string S, count all palindromic substrings and print them.Note that if we apply the above check 1 for every substring of the given string, we will end up with a horribly inefficient algo because we will repeatedly solving the the same sub-problems.Solve it using DP.

Manacher’s algo does in O(n) time.

<http://stackoverflow.com/questions/19801081/find-all-substrings-that-are-palindromes>

1. Given a string S, print all palindromic subsequences effectively.
2. Longest palindromic prefix/suffix
3. Find the minimum number of insertions needed at the end of the string to make it a palindrome(Ans : length of string- length of longest palindromic suffix)
4. Find the minimum number of insertions needed at the beginning of a string to make it a palindrome.(Ans : length of string - length of longest palindromic prefix)
5. Find minimum number of insertions to convert a string into a palindrome.(Ans : insertion may be at start or at end. Find the best one. Use DP-find the longest common substring between string and its reverse and subtract from string length.)
6. Palindrome Partitioning : Given a string S, partition it such that every substring of the partition is a palindrome. Return the minimum cuts needed for a palindrome partitioning of S. Return all possible palindrome partitioning of S.
7. Given a string, find the lexicographically smallest palindrome.

10.Read palindromic tree.