Palindrome Partition - QOTD 22 Jan 23

Leetcode Link : Click

$Approach - 1$ (backtrack & recursion)

Time : $O(n^n)$

> where n is length of input string

Space : $O(n^n)$

approach :-

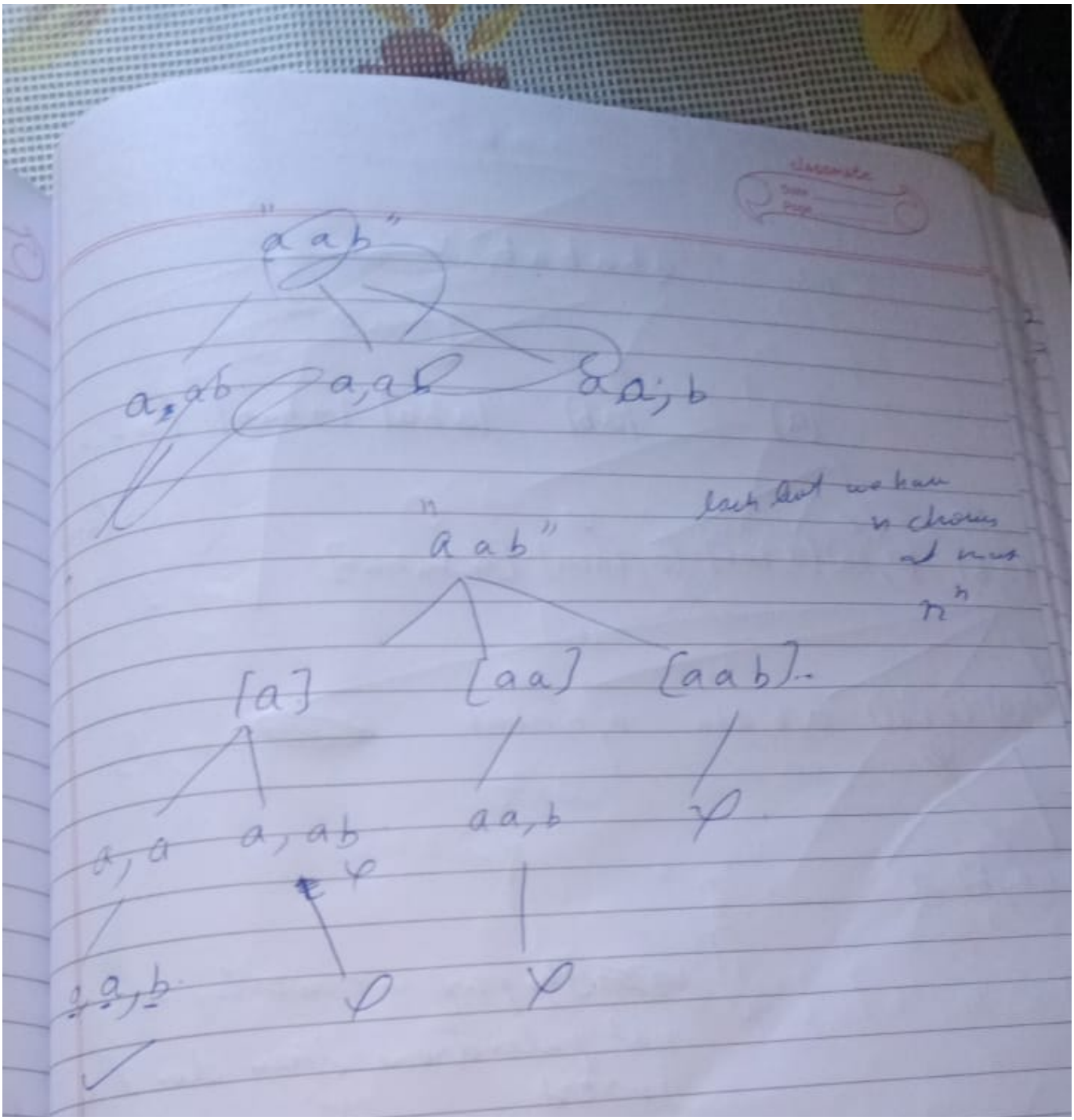
```
/*✔️ ⭐Appraoch - 1 (recursive & backtracking)

        explanation :-

                -> // Fun.3 - to check if the string we are inserting into 'ans' vector
is palindrome or not

                 -> // Fun.2 : this function will fill out result 2d vector with all
the possible /partitioning combinations that can be palindrome

                    step1 : base case- if the string is completely consumed, then push
the ans into result
                    step 2 : run a loop for complete length of string (including i =
length )
                        step 2.1 : if the sub part doesnt exsist then break the loop
                        step 2.2 : fetch the part of string and then check if it is
palindrome or not
                        step 2.3 : if part is palindrome then push it into the ans ,
and recursively call for index = index + i, while returning pop the last element of
the ans


                -> // MAIN Function
                        step 1 : create a 'ans' vector<string> and a 'result' 2d
string vector, index = 0
                        step 2 : call function.2 palindromePart(s, result, ans, index)

        ☑️T : O(n^n)
          S - O(n^n) [worst case]


        ☑️solved at leetcode :

*/
```

code :-

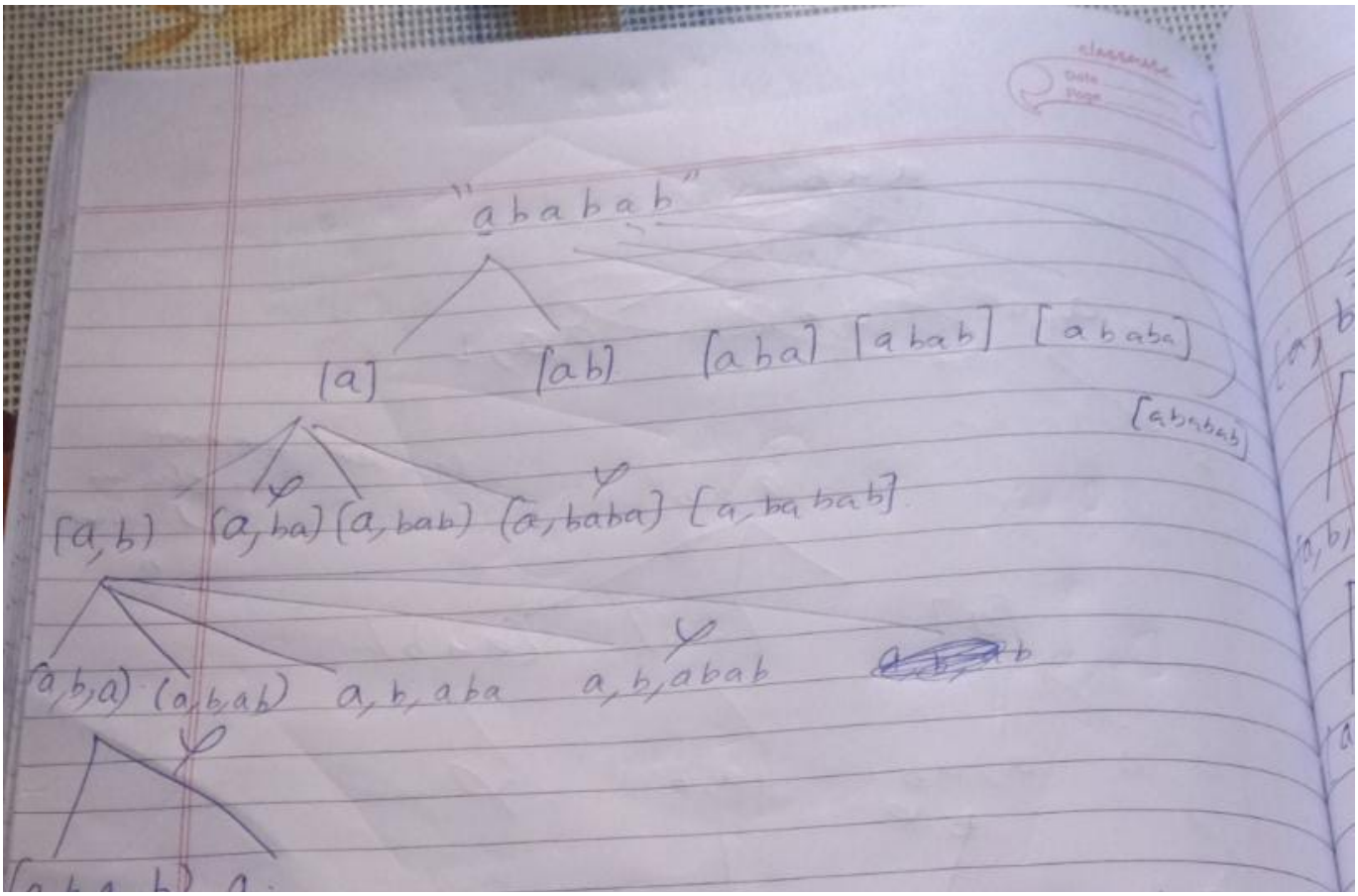
```
private:
    // Fun.3 - to check if the string we are inserting into 'ans' vector is palindrome
or not
    bool isPalindrome(string s){

        int i = 0, j = s.length() -1;
        while(i < j){
            if(s[i] != s[j]) return false;
            i++;
            j--;
        }
        return true;
    }

    // Fun.2 : this function will fill out result 2d vector with all the possible
partitioning combinations that can be palindrome
    void palindromePart(string &s, vector<vector<string>> &result, vector<string>
&ans, int index){
        // step1 : base case- if the string is completely consumed, then push the ans
into result
        if(index == s.length()){
            result.push_back(ans);
            return;
        }

        // step 2 : run a loop for complete length of string (including i =  length )
        for(int i = 1; i <= s.length(); i++){

            // step 2.1 : if the sub part doesnt exsist then break the loop
            if(i + index > s.length()) break;

            // step 2.2 : fetch the part of string and then check if it is palindrome
or not
            string part = s.substr(index,i);

            // step 2.3 : if part is palindrome then push it into the ans , and
recursively call for index = index + i, while returning pop the last element of the
ans
            if(isPalindrome(part)){
                ans.push_back(part);
                palindromePart(s, result, ans, index + i);
                ans.pop_back();

            }
        }
    }
public: // MAIN Function
    vector<vector<string>> partition(string s) {

        // step 1 : create a 'ans' vector<string> and a 'result' 2d string vector,
```

```
index = 0
        int index = 0;
        vector<string> ans;
        vector<vector<string>> result;

        // step 2 : call function.2 palindromePart(s, result, ans, index)
        palindromePart(s, result, ans, index);

        return result;
    }
```

dry run :



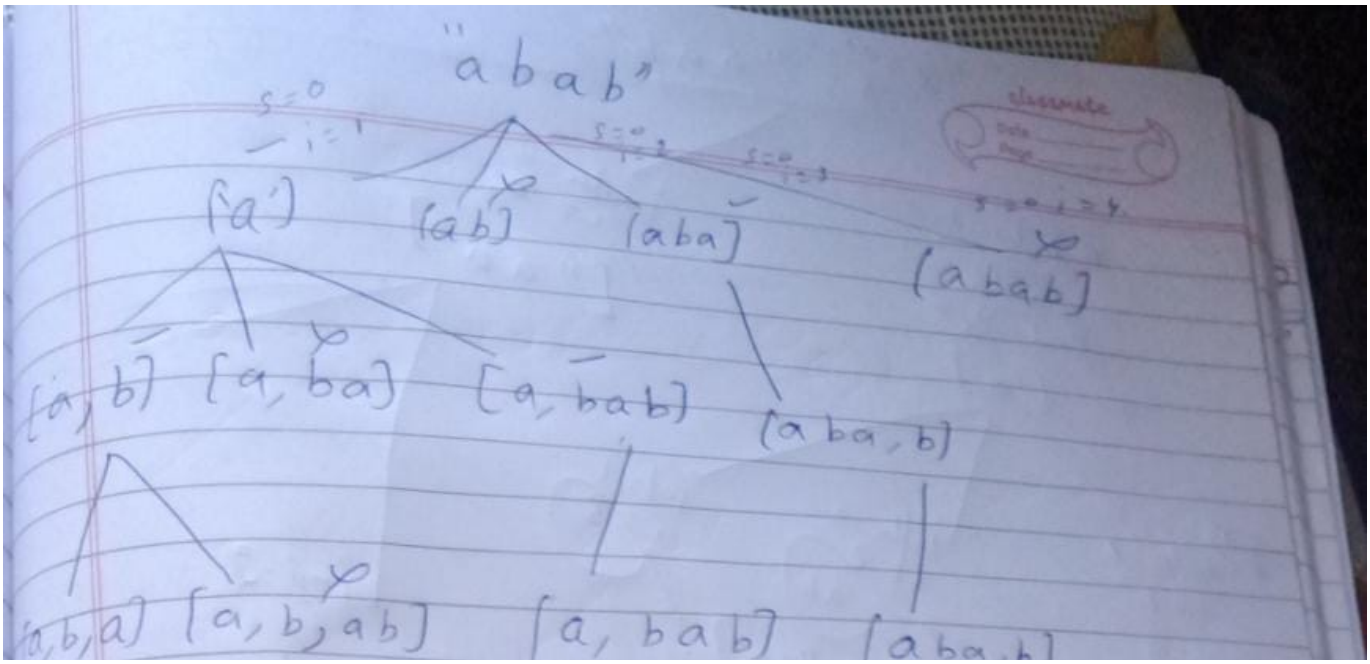
['a', 'a', b'] ['aa', b']

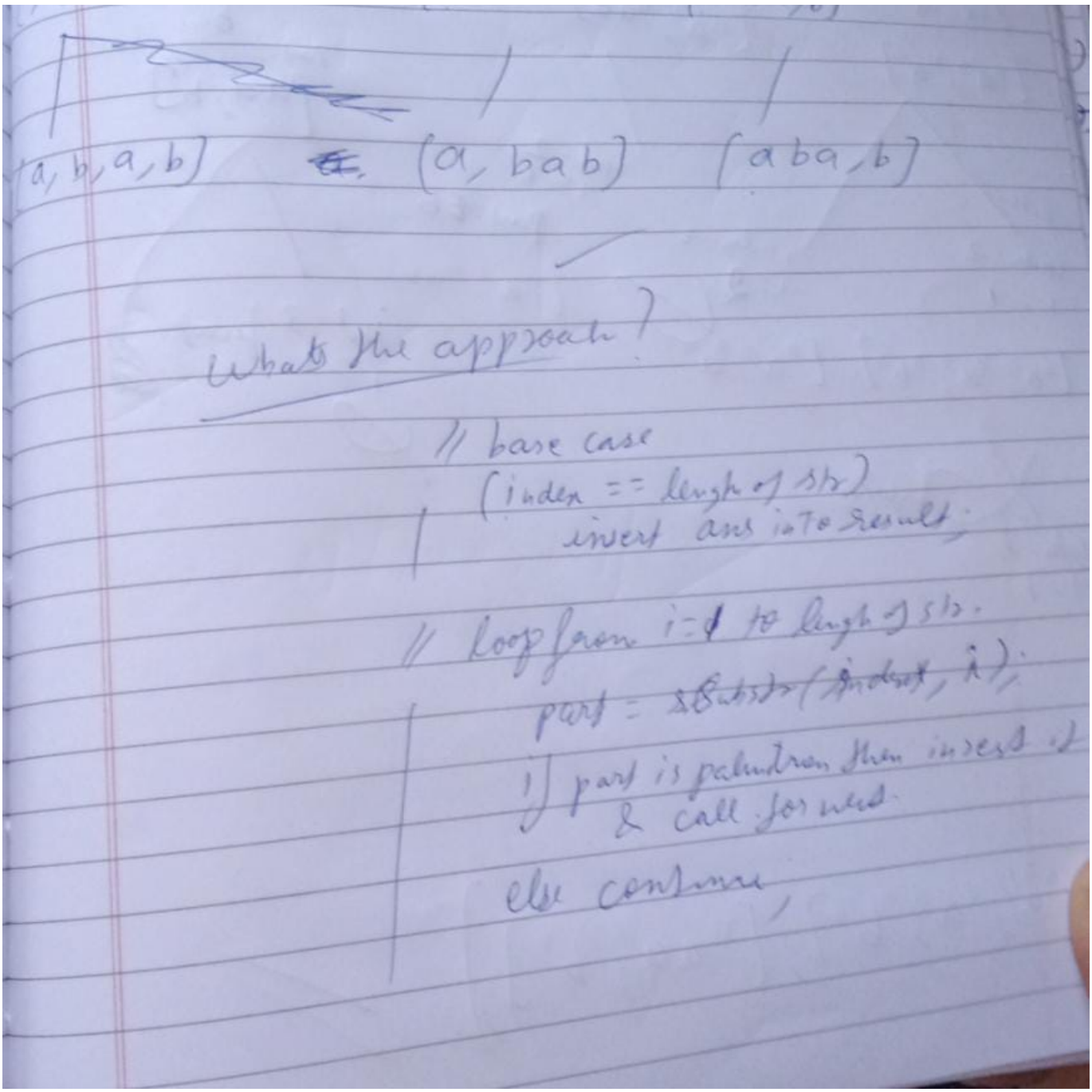

"ababab"
[a]
[ab]
[aba]
[abab]
[ababa]
[ababab]
[a,b]
[a,ba]
[a,bab]
[a,baba]
[a,babab]
[a,b,a]
[a,b,ab]
a,b,aba
a,b,abab

if ~~the~~ the part we inserting is not palindrome then don't insert.

heech

heoeh.

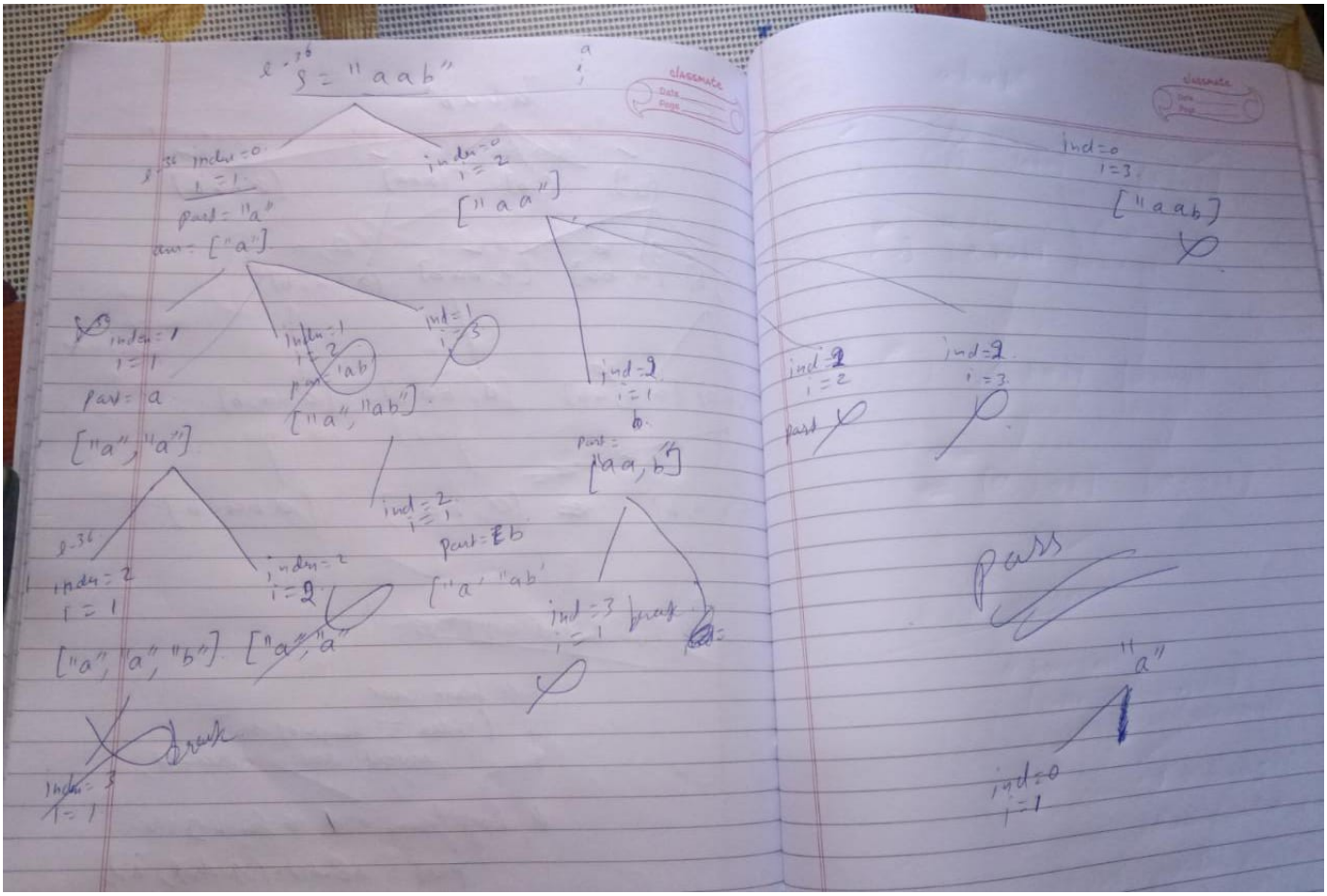




What's the approach?

// base case
(index == length of str)
insert ans into result;

// loop from i=1 to length of str.
part = substr(index, i);
// part is palindrom then insert it
& call for rest.
else continue;


s = "aab"
ind=0
i=1
part="a"
arr=["a"]
ind=0
i=2
["aa"]
ind=0
i=3
["aab"]
ind=1
i=1
part=a
["a","a"]
ind=1
i=2
"ab"
["a","ab"]
ind=1
i=3
ind=2
i=1
b
part=
["aa,b"]
ind=2
i=1
part=b
["a","ab"]
ind=2
i=2
part
ind=2
i=3
ind=2
i=1
["a","a","b"]
ind=3
i=1
ind=3
i=1
pass
"a"
ind=0
i=1