Recursion Subset, Subsequence, String Questions :-

* Basic Questions.

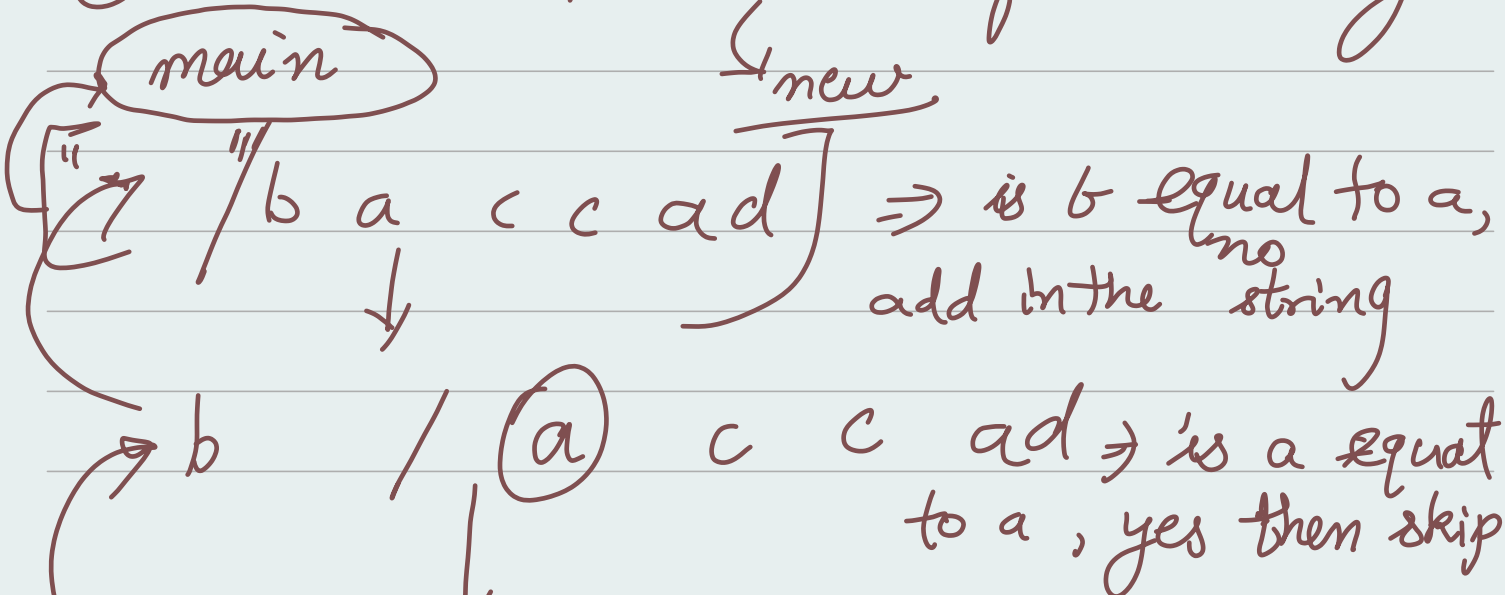
Q: Given a string, create another string that has removed all the 'a' from it (skip a character)

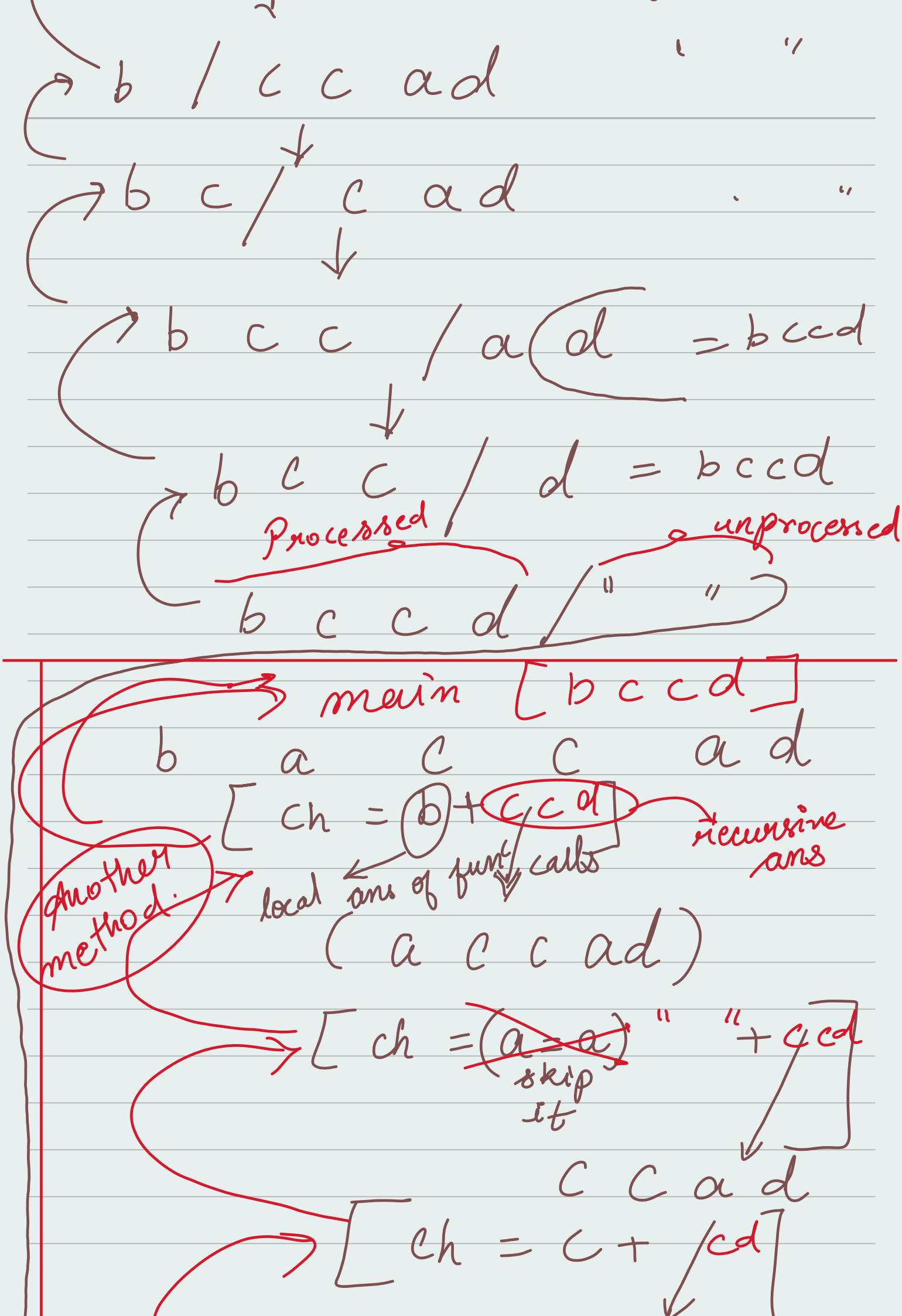
string = b a c c ad

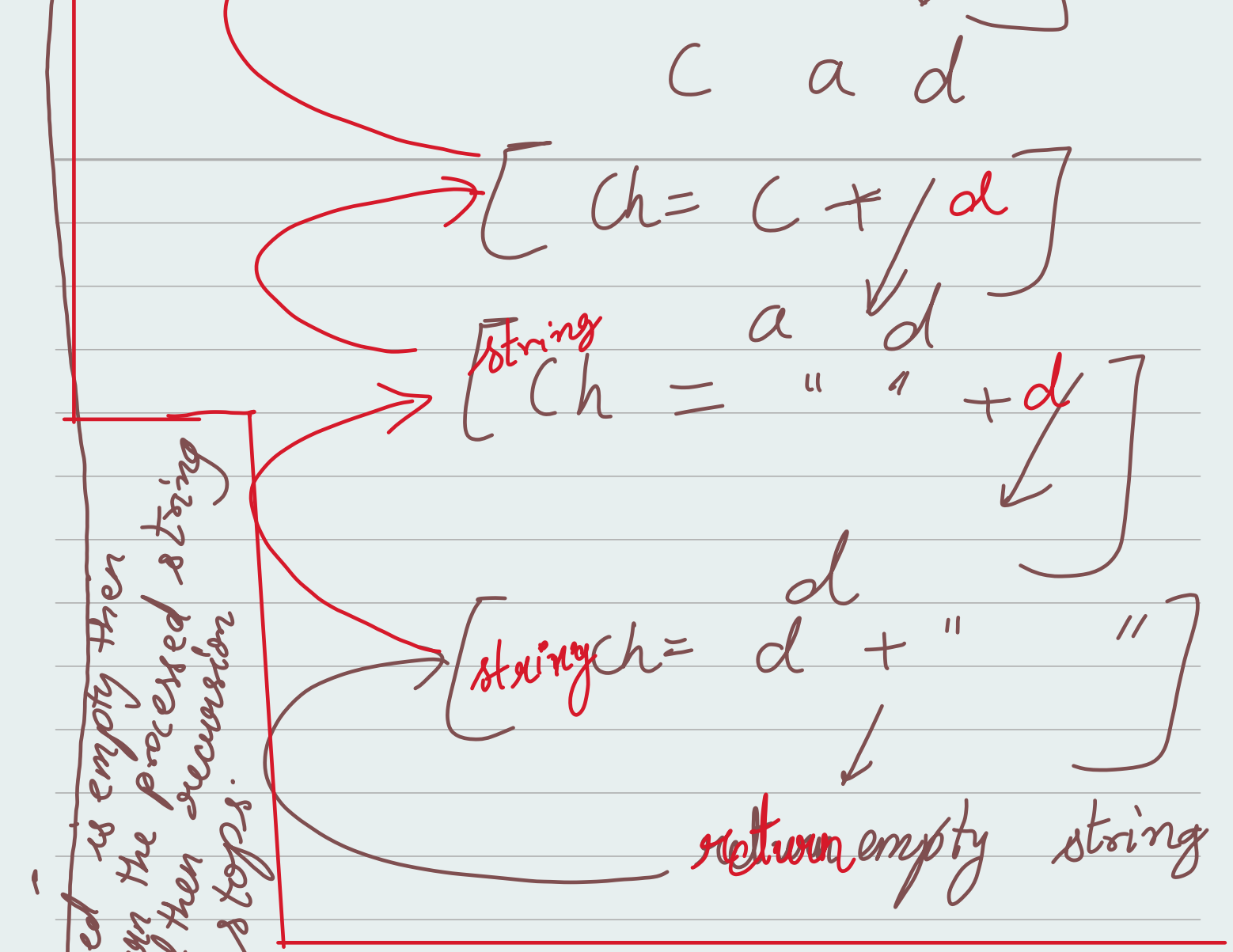
ans = b c c d

① pass the answer string in argument. (can be passed to future calls)

② create the ans var in function body.




b / c c a d
b c / c a d
b c c / a d = b c c d
b c c / d = b c c d
Processed
unprocessed
b c c d / " "
main [b c c d]
b a c c a d
[ch = b + c c d]
recursive ans
local ans of fun
calls
another method
(a c c a d)
[ch = (a = a) skip it " " + c c d
c c a d
[ch = c + c d



p = processed; up = unprocessed

Code: method 1 code

```
psvm ( ){
    skip(p: " " , up" b a e c a h d");
```

```
static void skip (string p, string up){
    if ( up isEmpty( ) ){
        System.out.println(p);
        return;
    }
```

```
char ch = up.charAt(0);
if (ch == 'a'){
    // skip it
    skip(p, up.substring(1));
} else {
    // don't skip it.
    skip(p+ch, up.substring(1));
}
```

Here substring calls 1st character by skipping the 'a'

if it is ch == a

ex: b c e / a d "it is called" skipp

So skip = p, up.substring(1);

for ch != a skip = p+ch, up.substring(1);

Method 2 code

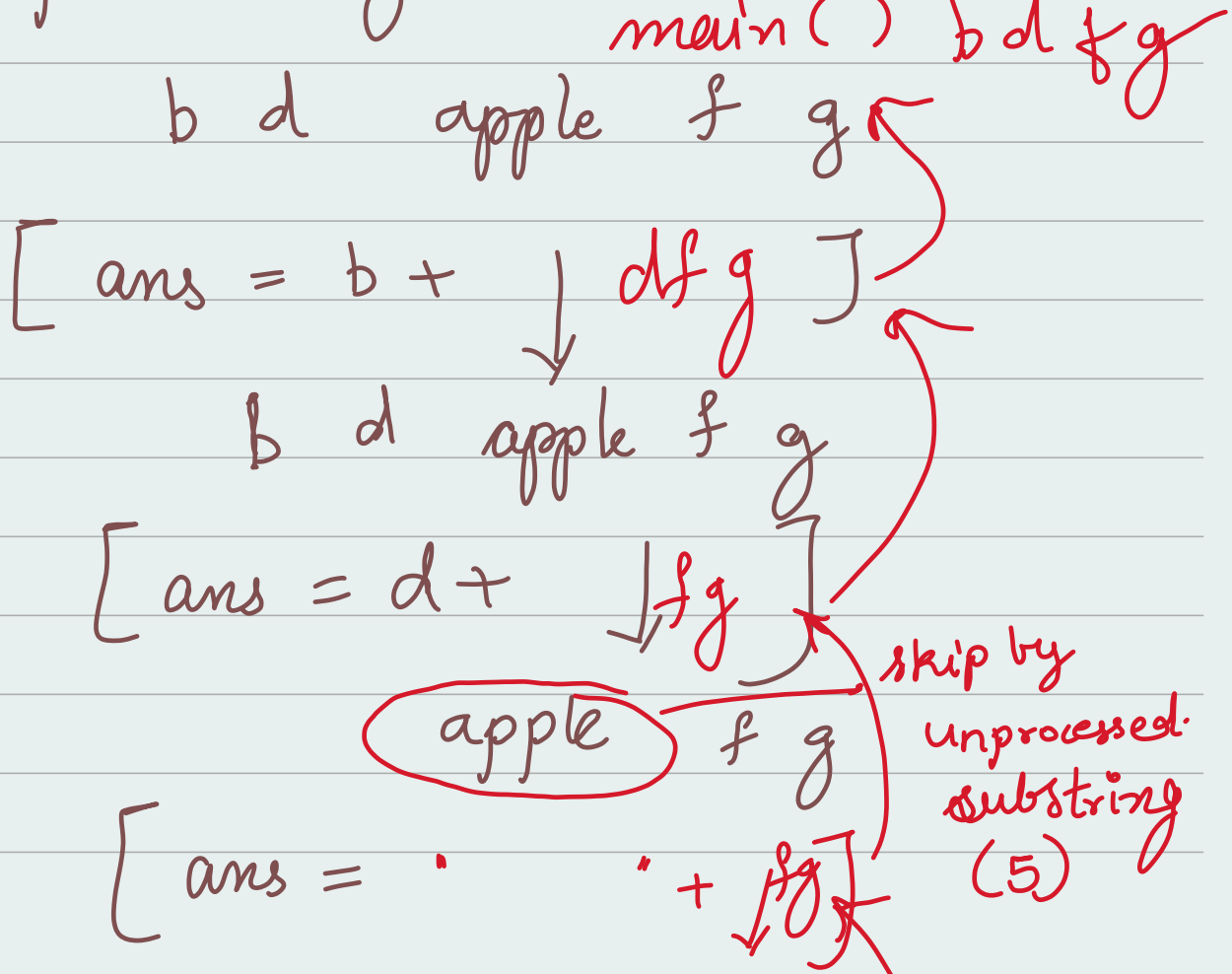
```
static String skip(String
                   unprocessed) {
```

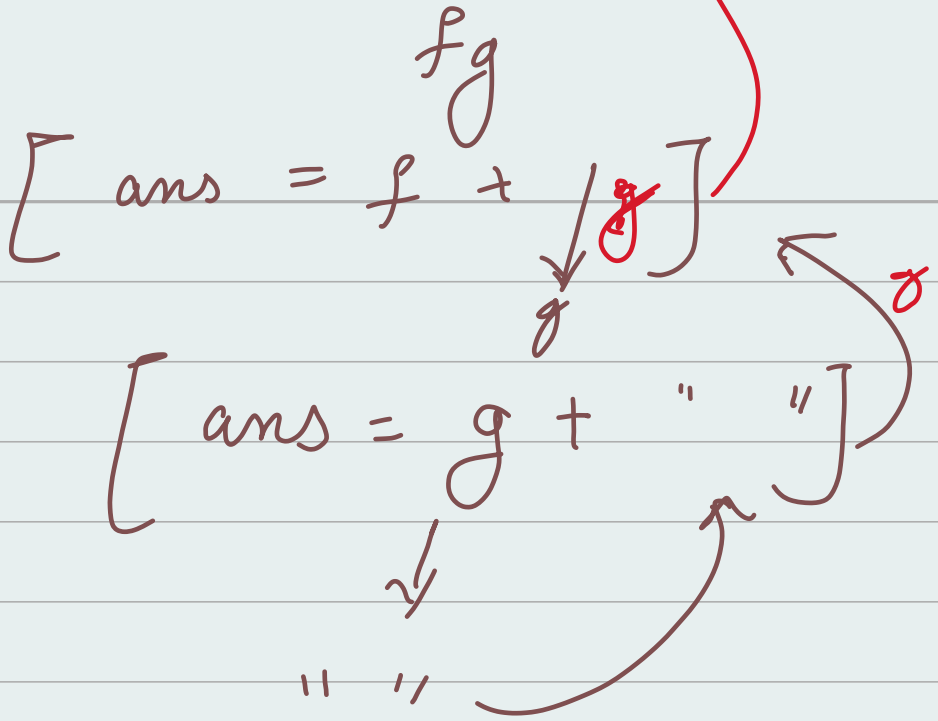
```
if (unprocessed.isEmpty()){
    return "";
}

Char ch = unprocessed.charAt(0);

if (ch == 'a'){
return skip(unprocessed.substring(1));
} else{
return ch + skip(unprocessed.substring(1));
}
}
```

8) Skip a string.





Code:

```
public static void main (String [], args){
System.out.println (skipApple ( baapplefg));
}
static String skipApple (String up){
    if( up.isEmpty() ){
        return " ";
    }
    if (up.startsWith ("apple")){
    // skip it
    return skipApple(up.substring(5));
}else{
    // don't skip it → Adding the 1st character
return up.charAt(0) + skipApple
    (up.substring(1));
}
}
```

8) Skip a string if its not the required string.

Only skip the app when it's not equal to Apple

means up.startsWith("app") && !
up.startsWith("apple"))

```
static String skipAppNotApple(String up){
    if(up.isEmpty()){
        return "";
    }
    if(up.startsWith("app") && !
       up.startsWith("apple")){
        return skipAppNotApple(up.substring(3));
    }
    return up.charAt(0) + skipAppNotApple
        (up.substring(1));
}
```

Subsets := [For Arrays]
Subsequence := [For Strings]
★ Permutations & combinations

★ Subsets → Non-adjacent collection.

[3, 5, 9] → [3], [3,5], [3,9]
[3, 5, 9], [5, 9] [5]
[9]

{1} {2} {3}
{1,2} {1,3}
{2,3}
{1,2,3}
{}

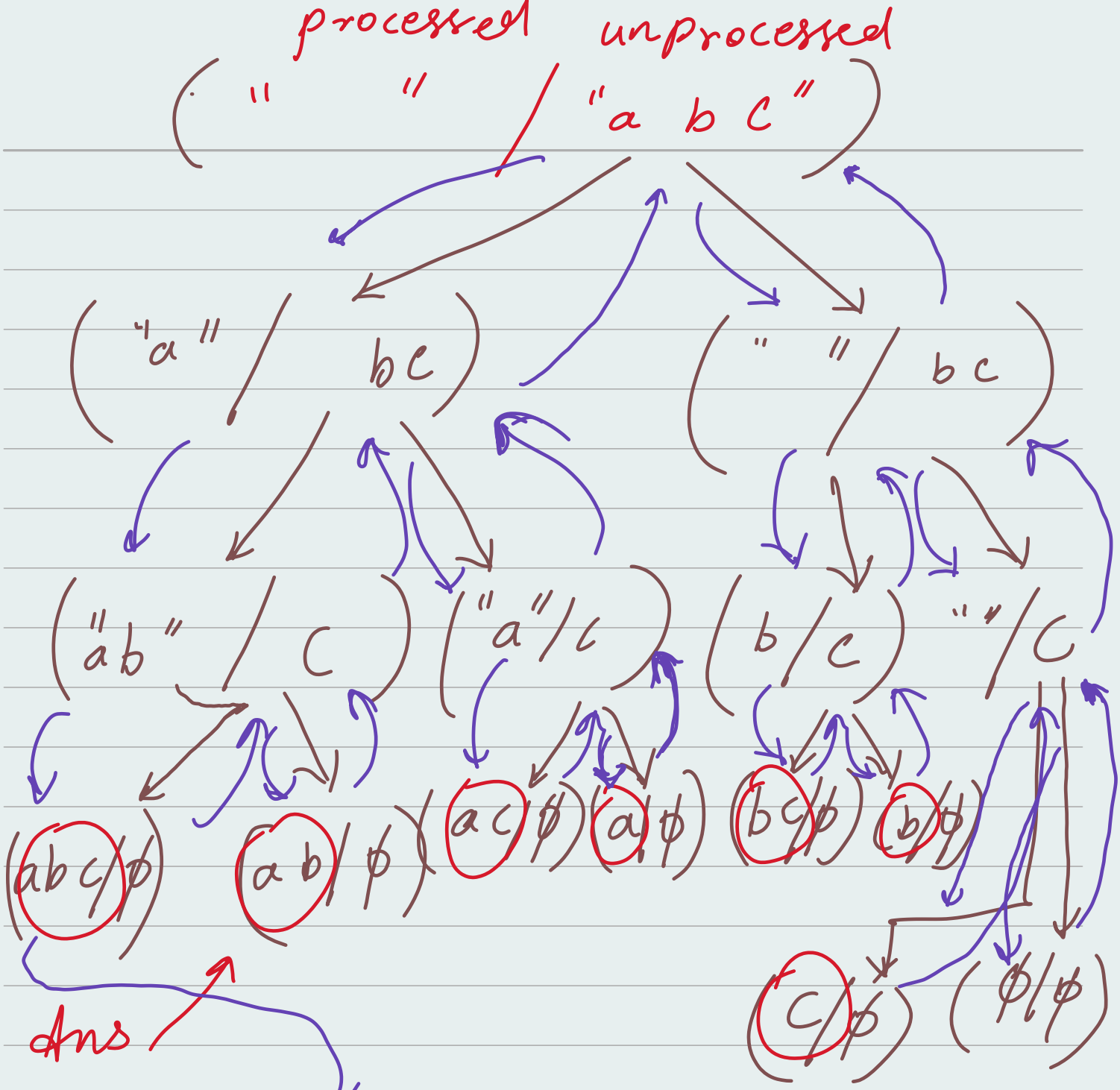
★ Recursion and Iteration

str = "a b c"

ans = ["a", "b", "c", "ab",
"bc", "ac", "abc"]

What & How? → focus

VVIP
★ This pattern of taking some elements and removing some elements is known as subset patterns.
For ex: "a" → bc is removed
Similarly in "bc" → a is removed
"ac" → b is removed.



Print (ABC, AB, AC, A, BC, B,

( )

Code:

```
public static void main(String[] args){
    subSeq("", "abc");
}
```

```
static void subSeq(String p, String up){
    if(up.isEmpty()){
        System.out.println(p);
        return;
    }
    char ch = up.charAt(0);
    subSeq(p+ch, up.substring(1));
    subSeq(p, up.substring(1));
}
```

output is given above it will be in the same order as above.

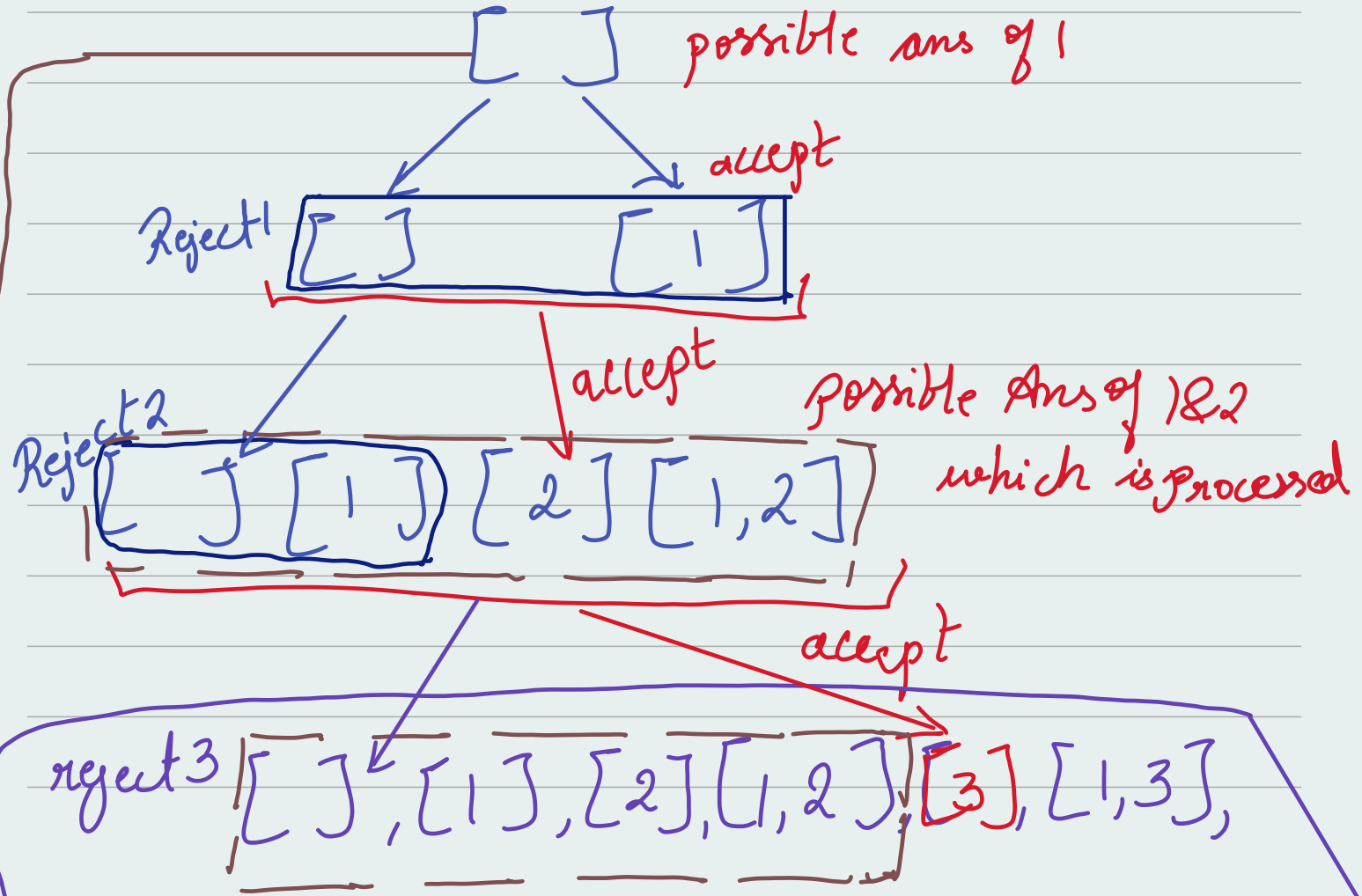
8) What if we return an arraylist of the above string?

```
static ArrayList<String> subSeqRet(String p, String up){
    if(up.isEmpty()){
        ArrayList<String> list = new ArrayList<>();
        list.add(p);
```

```
        return list;
    }
    char ch = up.charAt(0);
    ArrayList<String> left = subSeqRet(p + ch,
                                       up.substring(1));
    ArrayList<String> right = subSeqRet(p,
                                        up.substring(1));
    left.addAll(right);
    return left;
  }
}
```

3) Iterative program to print Subsequence

arr = [1, 2, 3]



[2, 3], [1, 2, 3]

Ans

[[ ], [ ], [ ], [ ], ...]

Code: This code is without recursion

```
static List<List<Integer>> subset(int[] arr){
    List<List<Integer>> outer = new ArrayList<>();
    outer.add(new ArrayList<>());
    for (int num : arr){
        int n = outer.size();
        for (int i=0; i<n; i++){
            List<Integer> internal = new
                    ArrayList<>(outer.get(i));
            internal.add(num);
            outer.add(internal);
        }
    }
    return outer
}
}
```

```
psvm () {
int[] arr = {1, 2, 3};

List<List<Integer>> ans = subset(arr);

for (List<Integer> list : ans) {
    System.out.println(list);
}
```

Time Complexity:

$O(N * 2^N)$ ← number of subsets we will have.

Space complexity: $O(2^N \times N)$

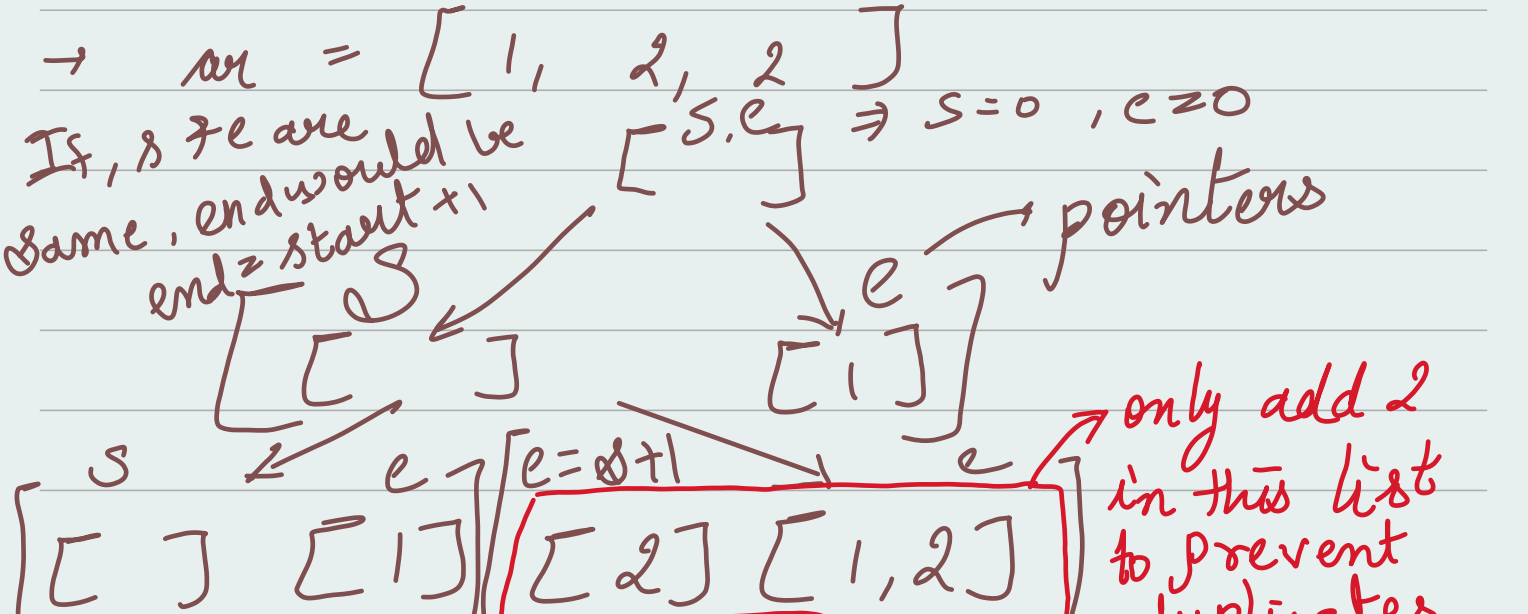
- 2^N → Total subsets
- N → space taken by each subset $= O(N)$

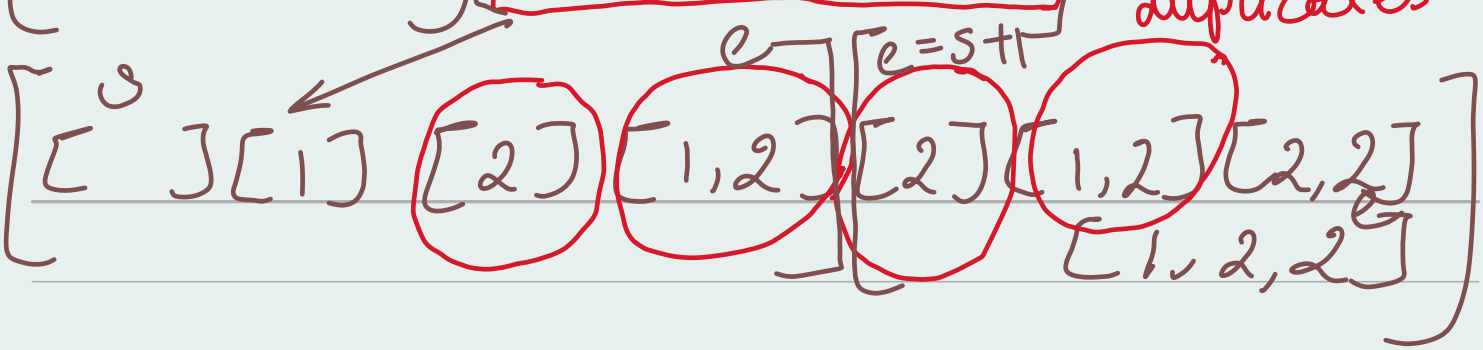
9) Subsequences of a string with duplicate elements

→ arr = [1, 2, 2]

If, s ≠ e are same, end would be end = start + 1

e.g. ⇒ s = 0, e = 0





* When you find a duplicate element, only add it in the newly created subset of previous step

* There is a condition to the above statement, the duplicate elements should be adjacent to each other Ex: [1 2 2]
Not ~~[2, 1, 2]~~
One way to achieve this is to Sort the array

```
static List<List<Integer>> SubsetDuplicate
                    (int[] arr){
    Arrays.sort();
    List<List<Integer>> outer = new
                        ArrayList<>();
    outer.add(new ArrayList<>());
    int start = 0,
    int end = 0;

    for(int i=0; i<arr.length; i++){
        start = 0;
        // If current and previous element is
```

```
same, s = e+1; //
if(i>0 && arr[i]==[arr i-1]){
    start = end+1;
}
end = outer.size()+1;
int n = outer.size()
for(int j=0; j<n; j++){
  List<Integer> internal =
      new ArrayList<>(outer.get(j));
  internal.add(arr[i]);
  outer.add(internal);
}
}
return outer;
}
}
ps vm(){
  int[] arr = {1, 2, 2}
  List<List<Integer>> ans =
                    subsetsDuplicate
                    (arr);
  for(List<Integer> list : ans){
    System.out.println(list);
  }
```