Permutations

→ Given a string, we have to print all the different permutations of the string.

→ "abc" →

a b c
a c b
b a c
b c a
c a b
c b a

3! → 6 permutations

"abaa" → abaa
aaba
aaab
baaa

4 permutation

2

$\frac{4!}{3!} \rightarrow 4$

aabacc → $\frac{6!}{3!\ 2!}$

abc →
- abc
- acb

- bac
- bca

- cab
- cba

→ every unique character of the string becomes the first character of some permutation

Then calculating permutations of the remaining chars

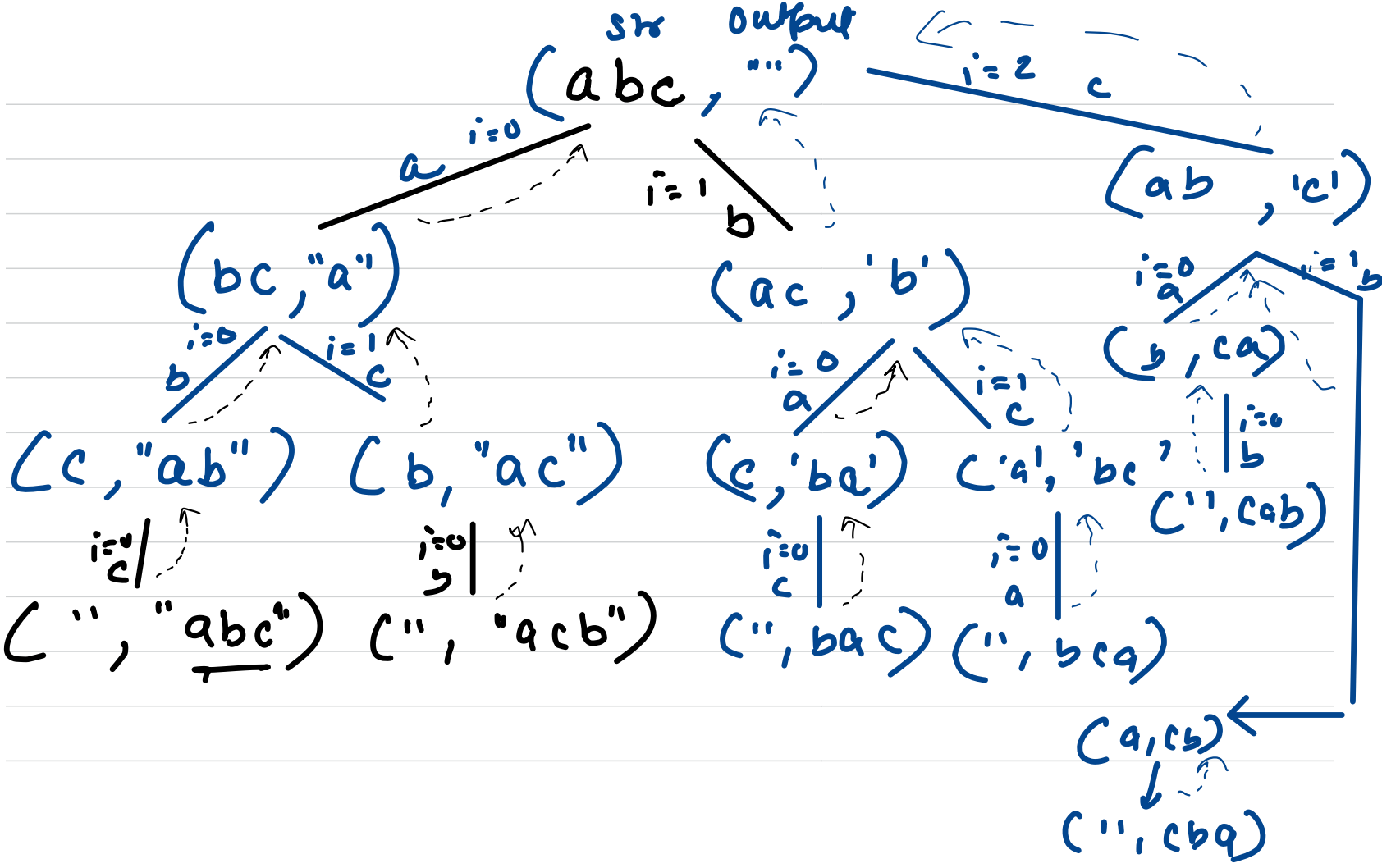
f (str, output) =

2

function prints all permutation of str

and every perm is stored in output

```
for (i=0; i<str.length(); i++){
    ch = str[i];
    ros = str.substr(0,i) + str.substr(i+1);
    f(ros, output+ch)
}
```


str
output
(abc, "")
i=0
a
i=1
b
i=2
c
(bc, "a")
(ac, 'b')
(ab, 'c')
i=0
b
i=1
c
i=0
a
i=1
c
i=0
a
(c, "ab")
(b, "ac")
(c, 'ba')
('a', 'bc')
(b, 'ca')
i=0
b
('', 'cab')
i=0
c
i=0
b
i=0
c
i=0
a
('', "abc")
('', "acb")
('', bac)
('', bca)
(a, cb)
('', cba)

S = a b c d → acd

S.substr(0,1) + S.substr(2)

a + cd

b

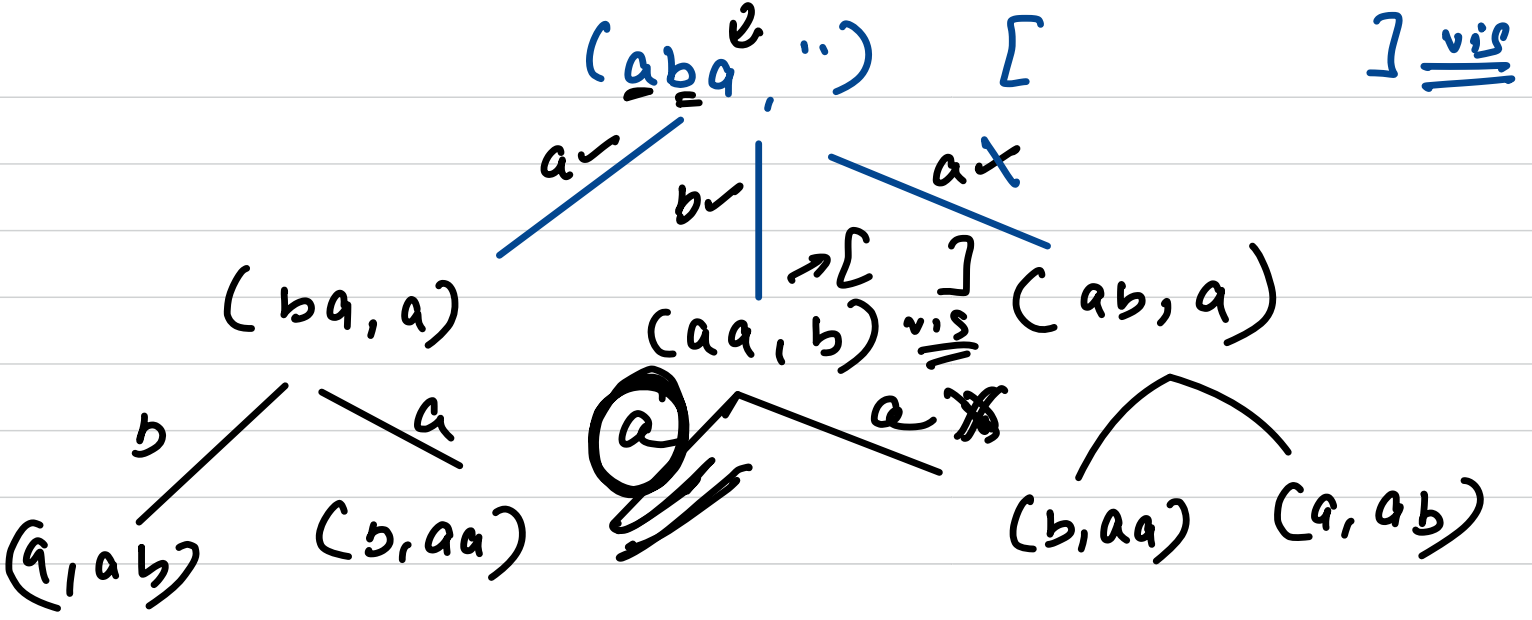
acd

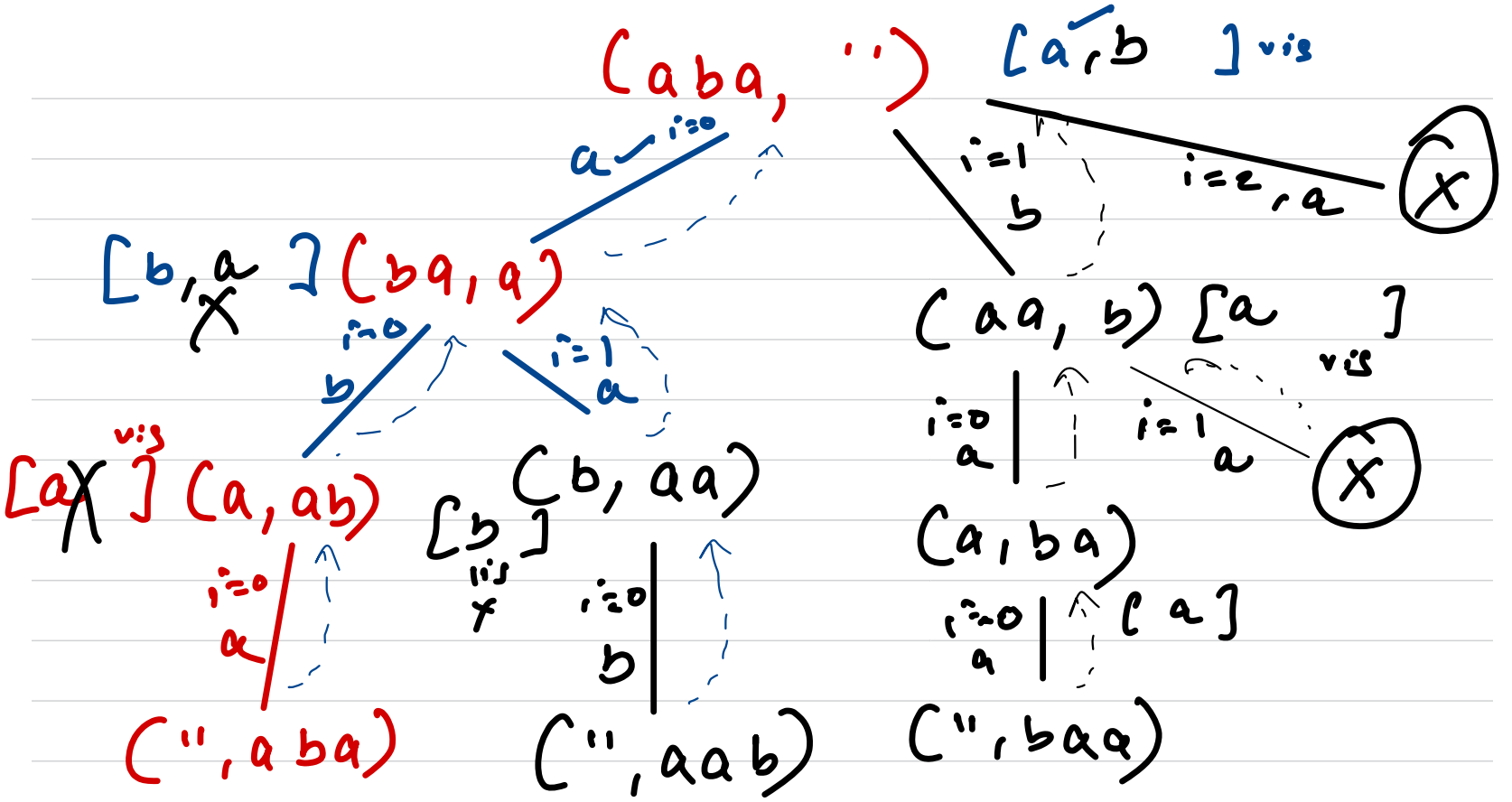
S.substr(0,2) + S.substr(3) abd

ab + d

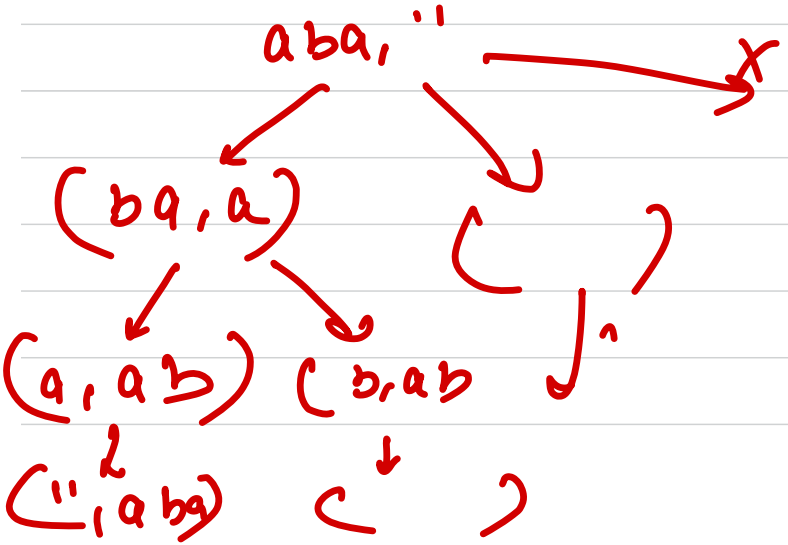
↓

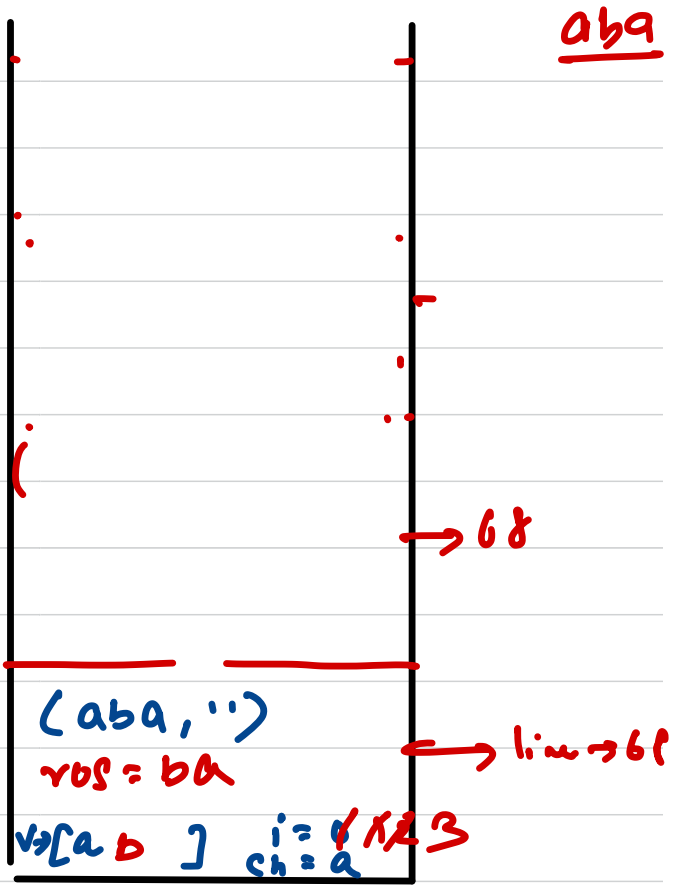
abd


(aba, ..)
[
]
vis
a
b
a
(ba, a)
(aa, b)
vis
(ab, a)
b
a
a
(a, ab)
(b, aa)
(a, ab)
(b, aa)


(aba, '')
[a, b] vis
a i=0
i=1 b
i=2, a
X
[b, a]
(ba, a)
i=0 b
i=1 a
(aa, b) [a]
vis
i=0 a
i=1 a
X
[a vis]
(a, ab)
(b, aa)
(a, ba)
[b vis]
i=0 a
i=0 b
i=0 a
[a]
('', aba)
('', aab)
('', baa)

```cpp
void permutations(std::string str, std::string output) { // abcde
    if(str.size() == 0) {
        std::cout<<output<<"\n";
        return;
    }
    std::vector<bool> vis(26, false);
    for(int i = 0; i < str.size(); i++) {
        char ch = str[i];
        if(not vis[ch - 'a']) {
            vis[ch - 'a'] = true;
            std::string ros = str.substr(0, i) + str.substr(i+1);
            permutations(ros, output + ch);
        }
    }
}
```



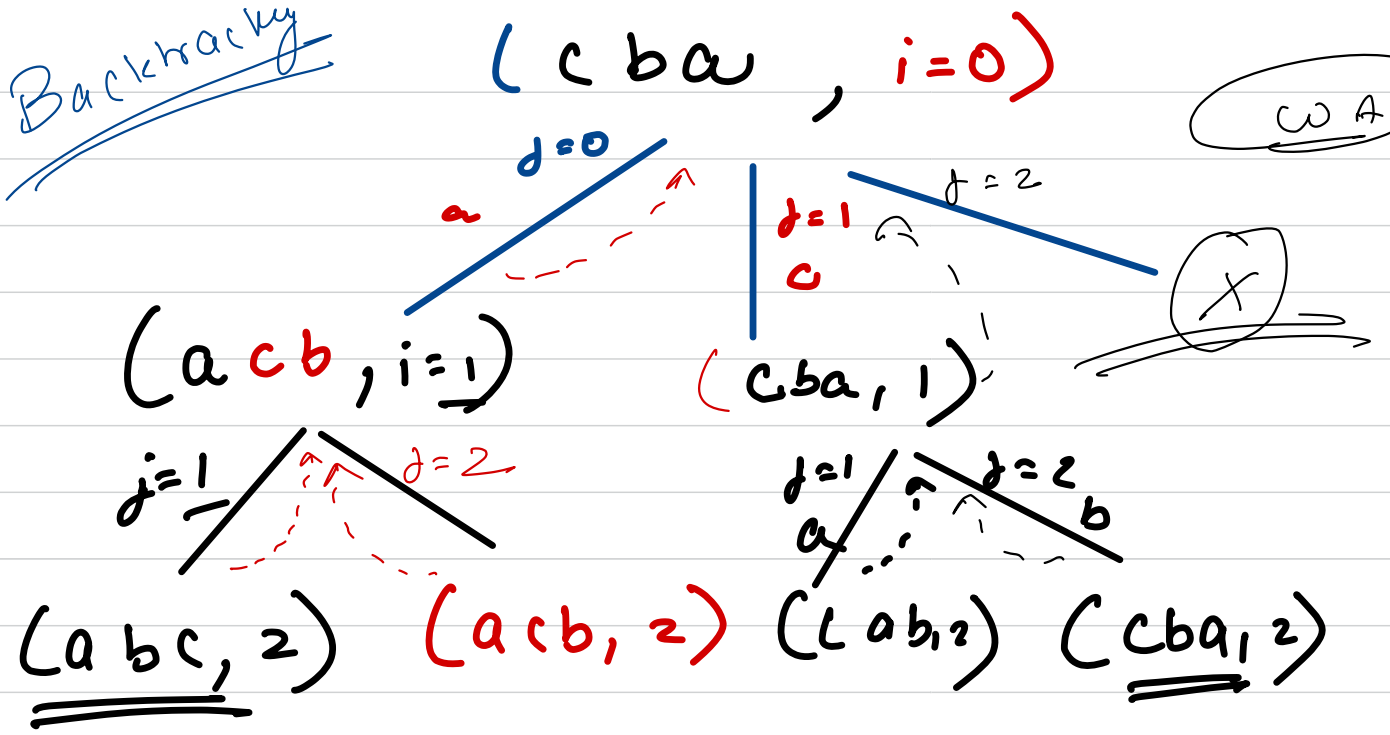


$T(n) = n \times T(n-1) + n^2 C$

$T(n)$ ↓

no. of ops for permutation of string of length n.

$\hookrightarrow O(n!)$


Backtracking
(cba, i=0)
WA
j=0
a
j=1
c
j=2
X
(acb, i=1)
(cba, 1)
j=1
j=2
j=1
j=2
a
b
(abc, 2)
(acb, 2)
(cab, 2)
(cba, 2)

f(str, i)

↓

return all permutat of str starting from index i

=

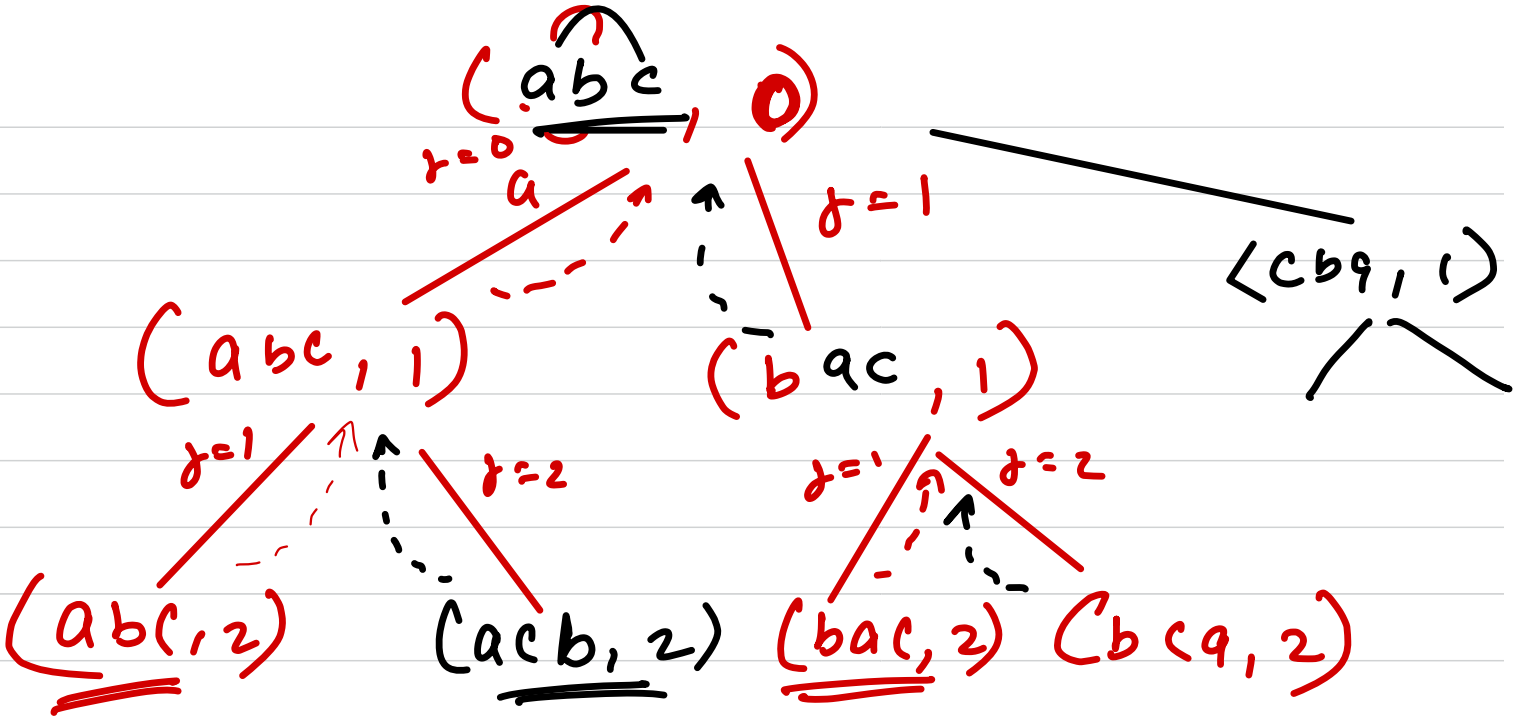
for(j=i; j<arr.length; j++)

swap(str, j, i)

f(str, i+1)

swap(str, j, i)

ans = f(str, 0)


(abc, 0)
r=0
a
r=1
(abc, 1)
(bac, 1)
(cbq, 1)
r=1
r=2
r=1
r=2
(abc, 2)
(acb, 2)
(bac, 2)
(bcq, 2)

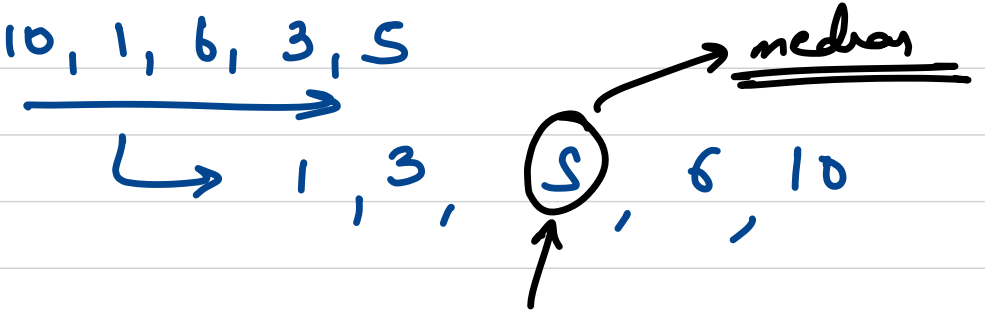
Median of 2 Sorted Arrays

We will be given 2 sorted (asc) arrays, our task is to find the median of these sorted arrays, where the no. of elements in the first array can be different than the other.

[7, 12, 14, 15]$_n$ }
[1, 2, 3, 4, 9, 11]$_m$ } → (8) and

→ 1, 2, 3, 4, 7, 9, 11, 12, 14, 15

$(n, m) \leq 10^6$



10, 1, 2, 3, 5, 6

1 2 3 5 6 10

middle

Brute force → 1) place second array after first array

2) Sort this combined data

3) Return the mid.

[ ]$_n$ [ ]$_m$

[ ]$_{n+m}$

$O((n+m)\log(n+m))$

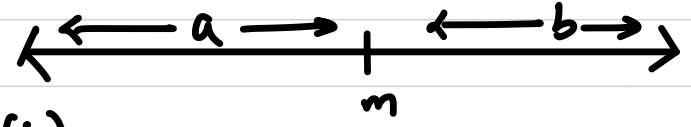
Merge two sorted arrays =

↳ O(n+m)

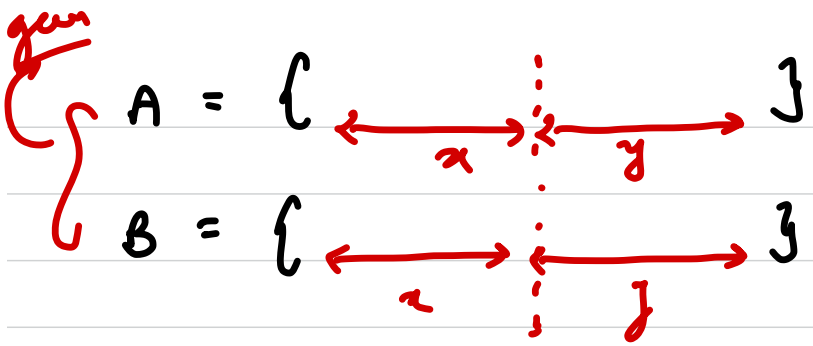
↳ Return the middle.

How can we optimise

→ Observation → In our final sorted array, median is the middle most element. So the no. of elements to the left of mid is going to be equal to no. of elements to the right of mid.

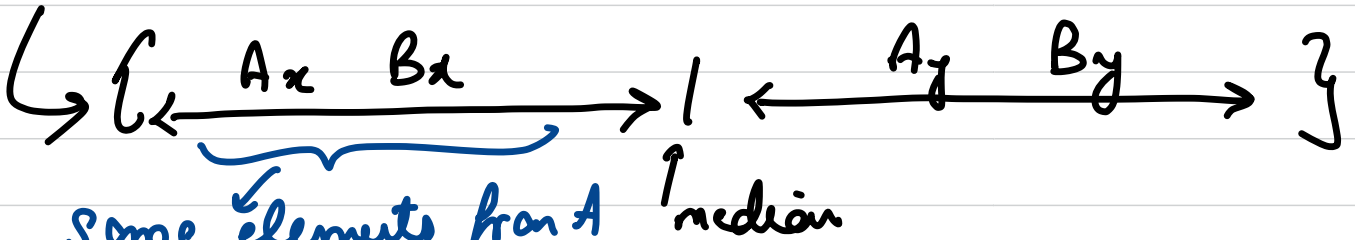


len(a) == len(b)

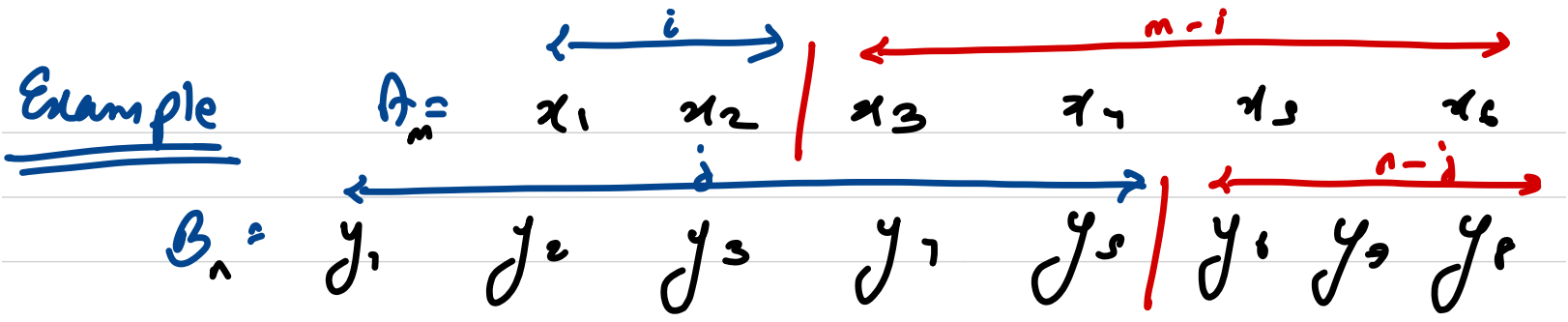


final array → merge (A, B)

C



after merging in sorted fashion, before median, we will be having some elements from A & some elements from B will be on the left of median. & Similarly for right side.



$C =$ $y_1\ y_2\ x_1\ y_3\ y_4\ x_2\ y_5$ M $x_3\ x_4\ y_1\ y_2\ x_5\ y_3\ x_6$

$i + j == (m - i) + (n - j)$

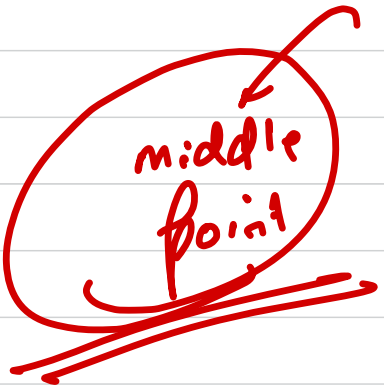
$len(left) == len(right)$

} Property

→ Length of array C will be m+n

and median will lie on the position

$\left(\frac{m+n+1}{2}\right)$ → median

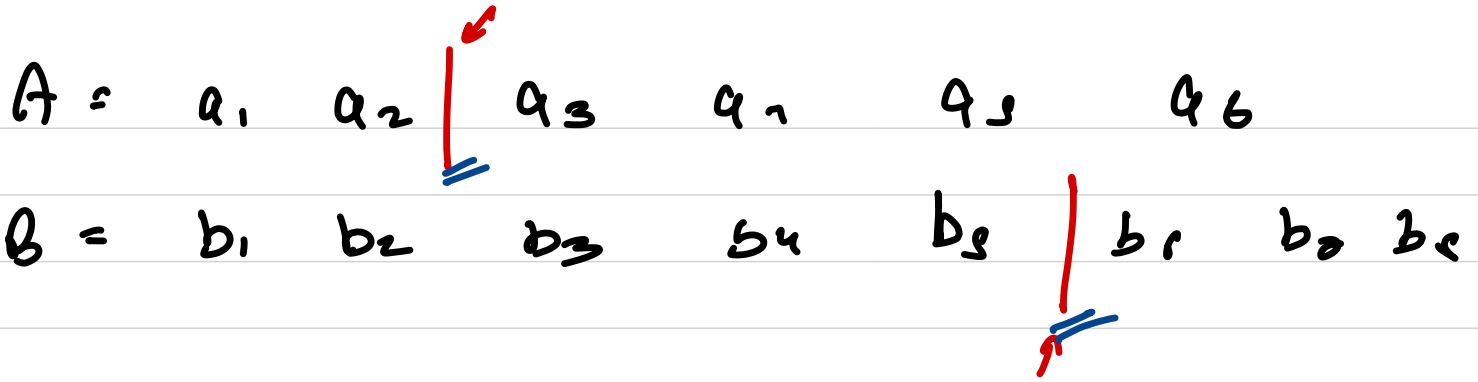


Now if we know how many elements from array A are going to contribute to the left of median, then we can easily calc the no. of elements that will participate on the left side of median from array B.

So we know length of final array → m+n

So we also know how many elements on left half of the array:

If you say no. of elements in A which will be part of left half is x then no of elements in B which will be part of left half is: $\frac{m+n+1}{2} - x$



$a_2 \leq a_5$ and $a_2 \leq b_6$

$b_5 \leq b_6$ $b_5 \leq a_3$

$\langle a_2\ b_5 \rangle$ | $\langle a_3,\ b_6 \rangle$

m

rightmost of leftside

leftmost of rightside.

These relations will only work when the split is correct.

during the partition we can check for the inequalities if they work then we have the ans otherwise not.

if $a_2 \neq b_6$ → now because a_2 is more than b_6, we need to reduce it's value. So we go left.

if $b_5 \neq a_3$ → now a_3 is less than b_5 so we need to increase it, that mean move pointer to right.

Using this we will be able to make correct partition on array A & then based on that partition we can get partition on array B.

[7, | 8, 12, 14, 15] 5

[1, 2, 3, 4, | 9, 11] 6

6+5 = 11

(5) M (5)

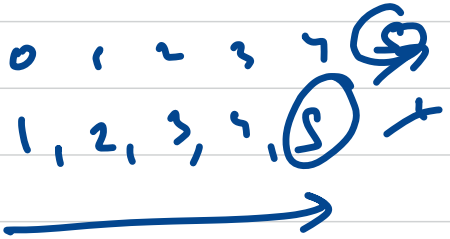
14 ≤ 2 ?! → No

8 ≤ 4 ?! No

7 ≤ 9 ?! Yes

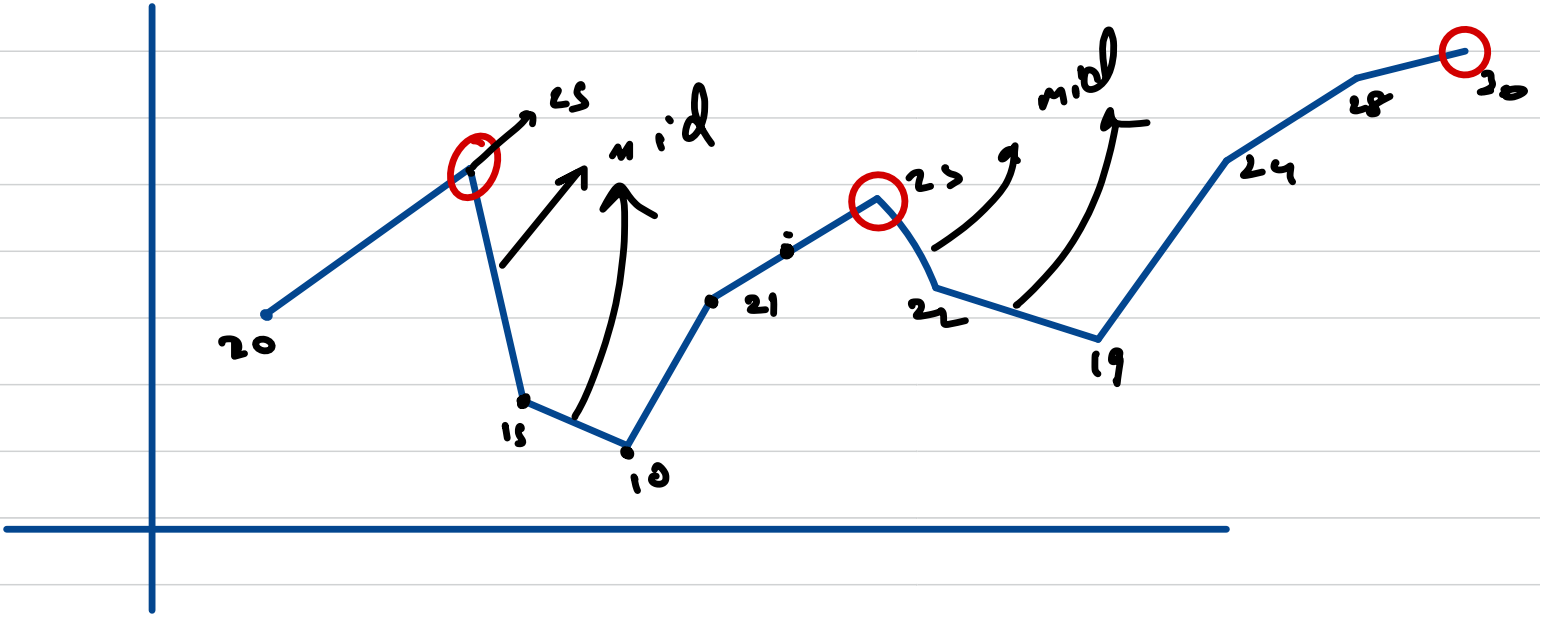
4 ≤ 8 → Yes

peak → a_i → peak → if $a[i-1] \leq a_i \geq a[i+1]$



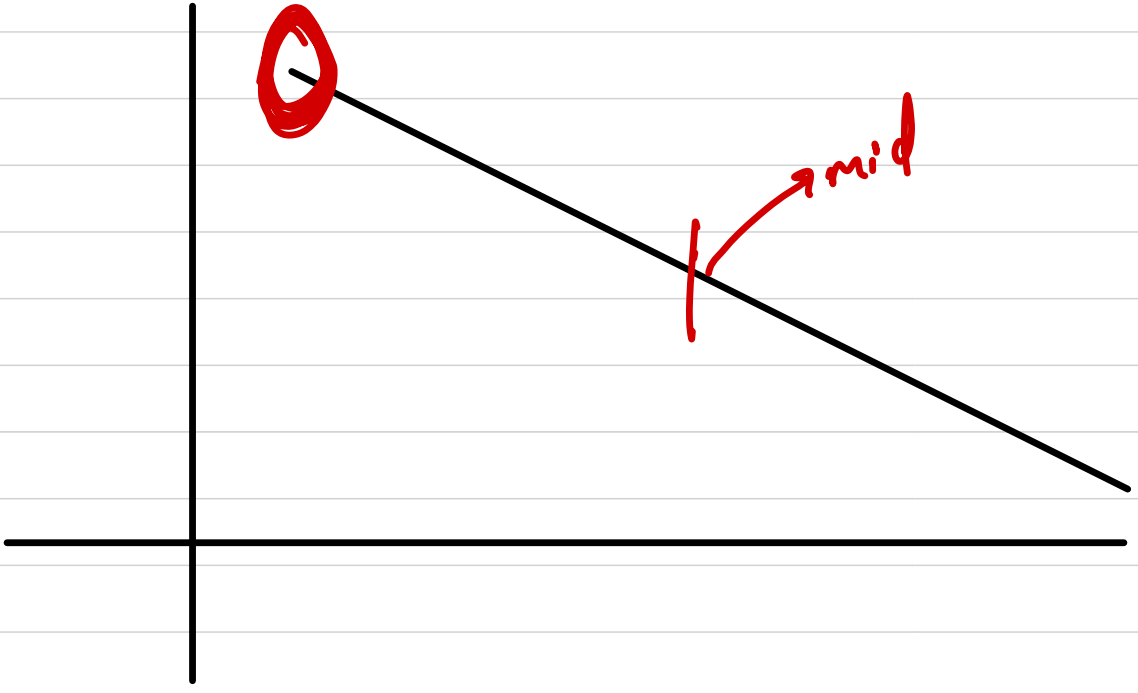
$a_4 \rightarrow 5$

$a_1 > a_3$



Search for peak → peak will be in increasing curve


mid


mid

if our mid point lands on an inc case, doesn't matter what the left hand side has, we are always going to get a peak on right.

if our mid point lands on a dec case, doesn't matter, what the right hand side has, we will always find a peak on left.