3. Given a string S, find out minimum number of deletions required to make the string a palindrome.

Note : Palindrome is a string which is same when read backwards and forward. For example : Civic, Kayak, Level etc

KAZAYAKE

We can delete Z and E, to make the word = KAYAK

Hint : Start from first and last.

1. State

1. State

Parameters

i - Starting index

j - ending index

Cost function

minDeletion(i,j,S) - Returns the minimum number of deletions required to make the substring starting at index i and ending at index j a palindrome.

2. Transitions

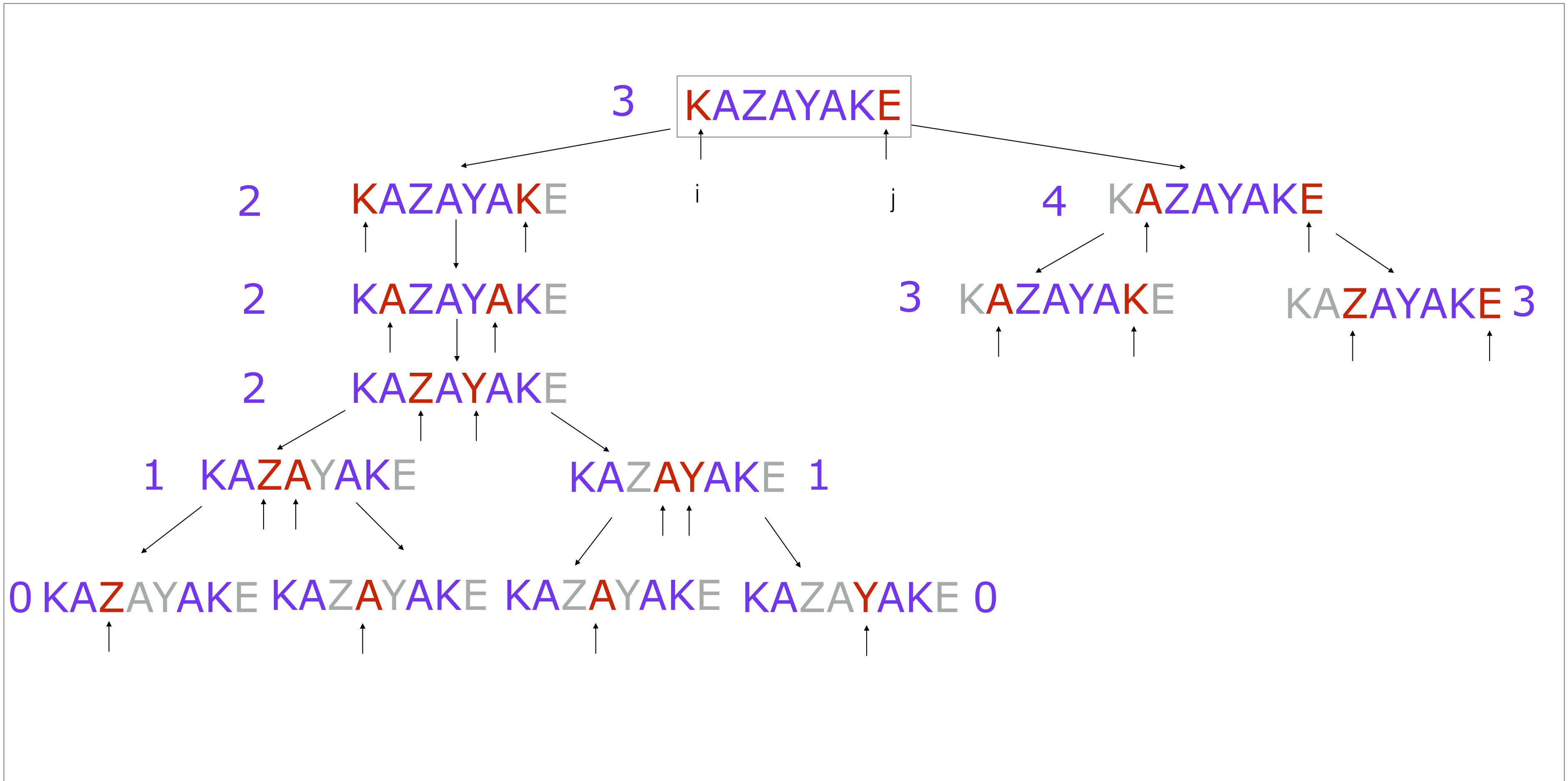
2. Transitions

Base case

if i >= j, return 0

minDeletion(i,j,S)

Make palindrome



2. Transitions

minDeletion(i,j,S)

Case 1:

S[i] = S[j]

minDeletion(i+1,j-1,S)

Case 2:

S[i] != S[j]

Choices

Delete character at i

minDeletion(i+1,j,S)

Delete character at j

minDeletion(i,j-1,S)

Recurrence relation

minCost(i,j,S) = 0 , if i>=j

minCost(i,j,S) = minCost(i-1,j+1) , if S[i] == S[j]

minCost(i,j,S) = MIN(minCost(i+1,j,S),minCost(i,j-1,S))+1

3. Recursive solution

3. Recursive solution

Pseudo code

```
minCost(i,j,S)
   if i >= j
      return 0
  if S[i] == S[j]
      return minCost(i+1,j-1,S)
  else
     return MIN(minCost(i+1,j),minCost(i,j-1))+1
```

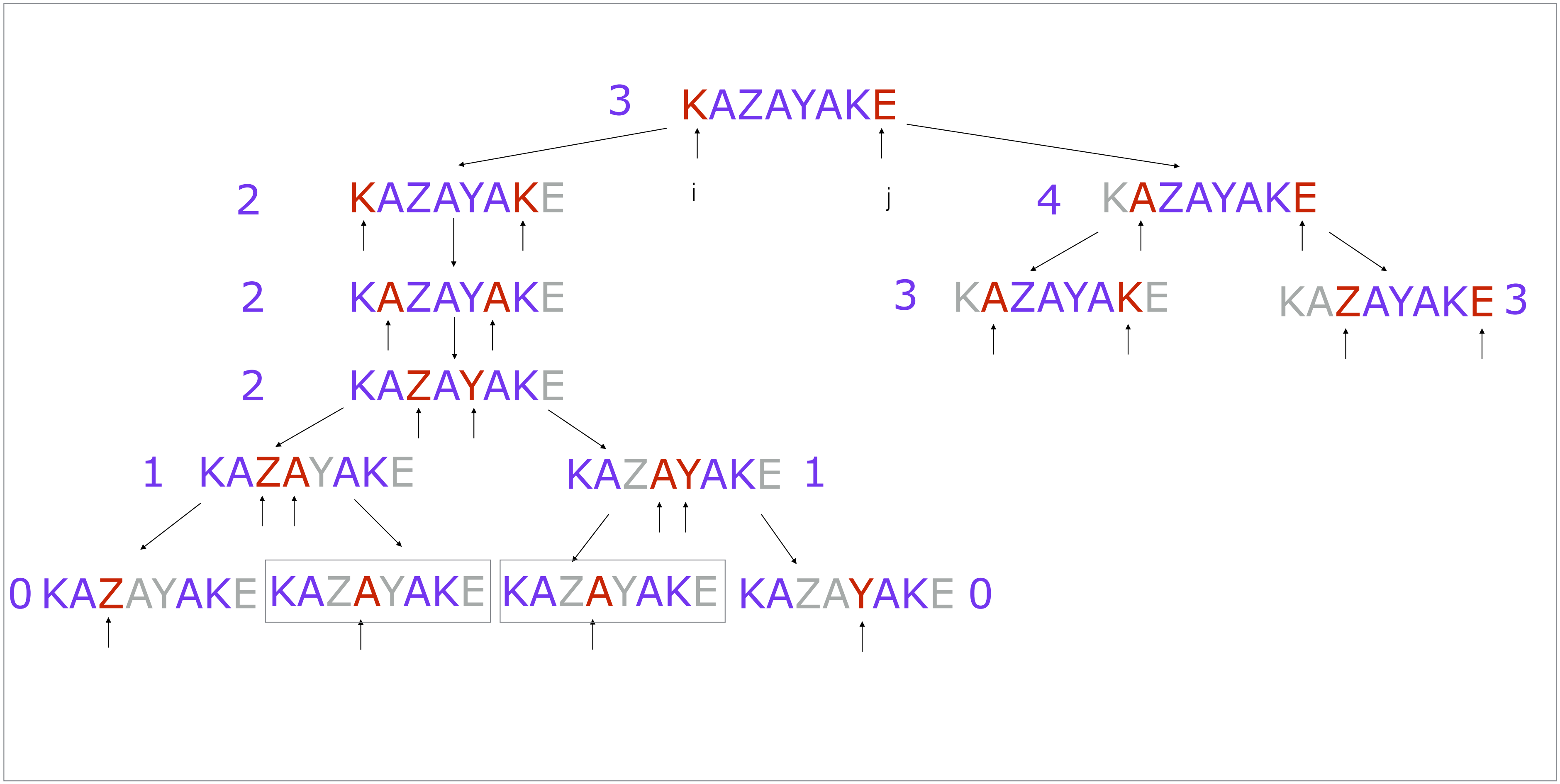
Java

```java
public static int minDeletionsPalindrome(int i,int j,String s){
    if(i >= j){
        return 0;
    }
    if(s.charAt(i) == s.charAt(j)){
        return minDeletionsPalindrome(i+1,j-1,s);
    }else{
        return
Math.min(minDeletionsPalindrome(i+1,j,s),minDeletionsPalindrome
(i,j-1,s))+1;
    }
}
```

Python

```python
def min_deletions(i, j, S):
    if i >= j:
        return 0
    if S[i] == S[j]:
        return min_deletions(i+1,j-1,S)
    else:
        return
min(min_deletions(i+1,j,S),min_deletions(i,j-1,S))
```

Make palindrome



4. Memoize

4. Memoize

We can cache the results in a 2D array.

Key -> (i,j) , starting and ending index of the substring

Value -> Minimum deletions required to make the substring palindrome.

Java

```java
public static int minDeletionsPalindromeMemo(int i,int j,String
s,int[][] cache){
    if(i >= j){
        return 0;
    }
    if(cache[i][j] != -1){
        return cache[i][j];
    }
    if(s.charAt(i) == s.charAt(j)){
        cache[i][j] = minDeletionsPalindrome(i+1,j-1,s);
        return cache[i][j];
    }else{
        cache[i][j] =
Math.min(minDeletionsPalindromeMemo(i+1,j,s,cache),minDeletionsPalindr
omeMemo(i,j-1,s,cache))+1;
        return cache[i][j];
    }
}
```

Python

```python
def min_deletions_memo(i, j, S, cache):
    if i >= j:
        return 0
    if cache[i][j] != -1:
        return cache[i][j]
    if S[i] == S[j]:
        cache[i][j] = min_deletions_memo(i + 1, j - 1, S,
cache)
        return cache[i][j]
    else:
        cache[i][j] = min(min_deletions_memo(i + 1, j, S,
cache), min_deletions_memo(i, j - 1, S, cache)) + 1
        return cache[i][j]
```

5. Bottom up approach

5. Bottom up approach

minCost(i,j,S) = 0 , if i==j

minCost(i,j,S) = minCost(i-1,j+1) , if S[i] == S[j]

minCost(i,j,S) = MIN(minCost(i+1,j,S),minCost(i,j-1,S))+1

dp[i][j] = 0 if i=j

dp[i][j] = dp[i+1][j-1] if S[i] = S[j]

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1]) if S[i] != S[j]

How do we determine which order the problems should be solved ?

Minimum deletions to make palindrome

K	A	Z	Y	A	K	E

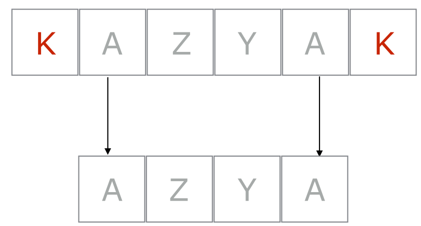
Minimum deletions to make palindrome

K	A	Z	Y	A	K	E

Minimum deletions to make palindrome

Minimum deletions to make palindrome

Minimum deletions to make palindrome



Minimum deletions to make palindrome

K	A	Z	Y	A	K	E

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K						
1	A						
2	Z						
3	Y						
4	A						
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A						
2	Z						
3	Y						
4	A						
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z						
3	Y						
4	A						
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z			0			
3	Y						
4	A						
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z			0			
3	Y				0		
4	A						
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z			0			
3	Y				0		
4	A					0	
5	K						

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z			0			
3	Y				0		
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0					
1	A		0				
2	Z			0			
3	Y				0		
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1				
1	A		0				
2	Z			0			
3	Y				0		
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1				
1	A		0	1			
2	Z			0			
3	Y				0		
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1				
1	A		0	1			
2	Z			0	1		
3	Y				0		
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1				
1	A		0	1			
2	Z			0	1		
3	Y				0	1	
4	A					0	
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1				
1	A		0	1			
2	Z			0	1		
3	Y				0	1	
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

$$dp[i][j] = MIN(dp[i+1][j], dp[i][j-1])+1$$

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2			
1	A		0	1			
2	Z			0	1		
3	Y				0	1	
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2			
1	A		0	1	2		
2	Z			0	1		
3	Y				0	1	
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2			
1	A		0	1	2		
2	Z			0	1	2	
3	Y				0	1	
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2			
1	A		0	1	2		
2	Z			0	1	2	
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3		
1	A		0	1	2		
2	Z			0	1	2	
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = dp[i+1][j-1]

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3		
1	A		0	1	2	1	
2	Z			0	1	2	
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3		
1	A		0	1	2	1	
2	Z			0	1	2	3
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

$$dp[i][j] = MIN(dp[i+1][j], dp[i][j-1]) + 1$$

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3	2	
1	A		0	1	2	1	
2	Z			0	1	2	3
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = MIN(dp[i+1][j],dp[i][j-1])+1

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3	2	
1	A		0	1	2	1	2
2	Z			0	1	2	3
3	Y				0	1	2
4	A					0	1
5	K						0

Minimum deletions to make palindrome

K	A	Z	Y	A	K

dp[i][j] = dp[i+1][j-1]

		0	1	2	3	4	5
		K	A	Z	Y	A	K
0	K	0	1	2	3	2	1
1	A		0	1	2	1	2
2	Z			0	1	2	3
3	Y				0	1	2
4	A					0	1
5	K						0

5. Bottom up approach

Pseudo code

```
minDeletion(i,j,S)
    N = S.length
    dp = [N][N]
   for L=1;L<=N;L++
       for i=0;i<=N-L;i++
           j = i+L-1
           if i == j
                 continue
           if S[i] == S[j]
                dp[i][j] = dp[i+1][j-1]
           else
                dp[i][j]  = MIN(dp[i+1][j],dp[i][j-1])+1
   return dp[0][N-1]
```

Java

```java
public static int minDeletionsPalindromeDP(String s){
    int N = s.length();
    int[][] dp = new int[N][N];
    for(int l=1;l<=s.length();l++){
        for(int i=0;i<=N-l;i++){
            int j = i+l-1;
            if(i == j){
                dp[i][j] = 0;
                continue;
            }
            if(s.charAt(i) == s.charAt(j)){
                dp[i][j] = dp[i+1][j-1];
            }else{
                dp[i][j] = Math.min(dp[i+1][j],dp[i][j-1])+1;
            }
        }
    }
    return dp[0][N-1];
}
```

Python

```python
def min_deletions_dp(S):
    N = len(S)
    dp = [[0 for _ in range(0,N)] for _ in range(0,N)]
    for l in range(1,N+1):
        for i in range(0,N-l+1):
            j = i+l-1
            if i == j:
                continue
            if S[i] == S[j]:
                dp[i][j] = dp[i+1][j-1]
            else:
                dp[i][j] = min(dp[i+1][j],dp[i][j-1])+1
    return dp[0][N-1]
```

Time and space complexity

Recursive implementation

Binary tree

Height of the tree - N

Time complexity , $O(2^N)$, Exponential

Space complexity , O(1), no extra memory other than recursion.

Dynamic programming

Time complexity

There are two for loops,

the outer for loop goes from L=1…N

the inner for loop goes from 0..N-L

There are N subproblems of size 1

There are N-1 subproblems of size 2

.

.

There is 1 subproblem of size N

So total run time = $N+N-1+N-2+\ldots..1 = N(N+1)/2 = (N^2+N)/2$

Worst case time complexity is $O(N^2)$

Space complexity , $O(N^2)$