Backtracking

Content

- Quizzes
- Subset
- Permutation

---

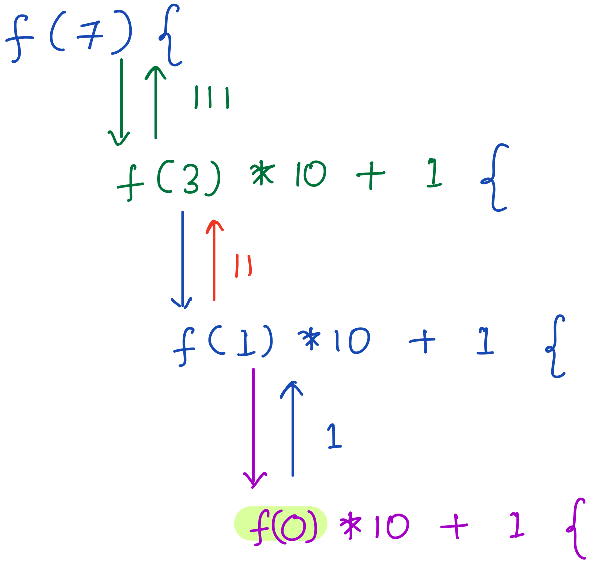
last time → Monday

64 % → 66.4 % → 75%

NOTE: Attempt all re-attempts for all contest.

```
int magicfun (int N) {
    if (N == 0) return 0
    else return magicfun (N/2) * 10 + N % 2
}
```

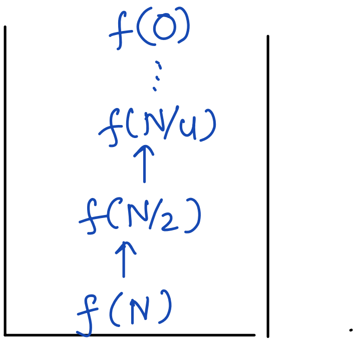
Output of above code for N = 7



TC : O(log(N))

SC : O(log(N))

NOTE : space complexity for a recursive code is never constant due to stack space.



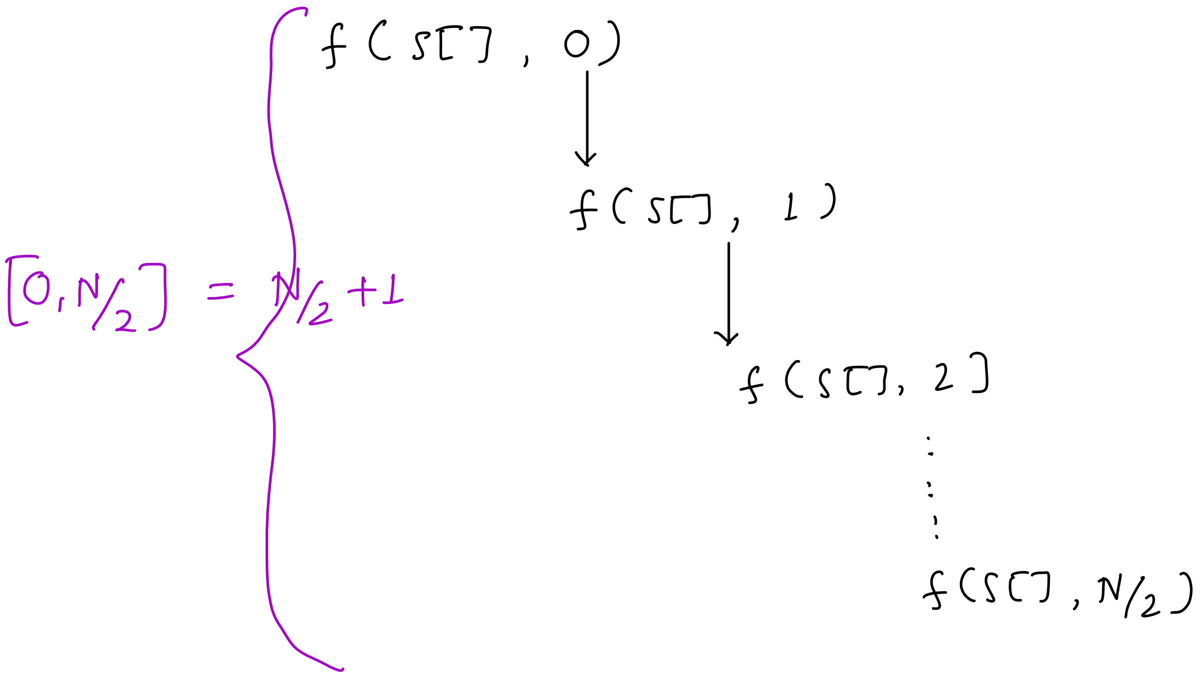
```
void  fun ( char  s[] ,  int  x ) {
        print (s)
        char  temp
        if ( x < s.length / 2 ) {
              temp  =  s[x]                              }  swap
              s[x]  =  s[s.length - x - 1 ]              }  ( x ,
              s[s.length - x - 1 ]  =  temp              }  n-x-1 )
              fun ( s , x+1 )
        }
}
```

Output for fun ("SCROLL", 0)   n = 6

```
f ( SCROLL , 0 ) {                                        output
        print ( SCROLL )                                  SCROLL
        swap ( 0 , 5 )                                    LCROLS
        f( LCROLS , 1) {                                  LLROCS
              print ( LCROLS )                            LLORCS
              swap ( 1 , 4 )
              f( LLROCS, 2 ) {
                    print ( LLROCS )
                    swap ( 2 , 3 )
                    f( LLORCS , 3 ) {
                          print ( LLORCS )
```

TC : $O(\frac{N}{2} * N)$ $O(N^2)$

SC : $O(N)$ TC = # function calls * TC per call



$TC = \#\text{ fn call} * TC \text{ per call}$

$= \left(\frac{N}{2} + 1\right) * N$

$= \frac{N^2}{2} + N \qquad O(N^2)$

SC = Max stack size at any instant

$= O\left(\frac{N}{2} + 1\right)$

$= O(N)$

Subset

* subset is not a subarray.

A = [1 2 3]

subsets = [ ]　　[1 2]
[1]　　[1 3]
[2]　　[2 3]
[3]　　[1 2 3]

⟶ 1 2 3　3 2 1　2 1 3 are same subset.

⟶ Every subarray is a subset ⟶ true

Every subset is a subarray ⟶ false

Given an array of distinct integer. Print all subsets using recursion.

I/P A = [1 2 3]

subsets = [ ]
[1]
[2]
[3]
[1 2]
[1 3]
[2 3]
[1 2 3]

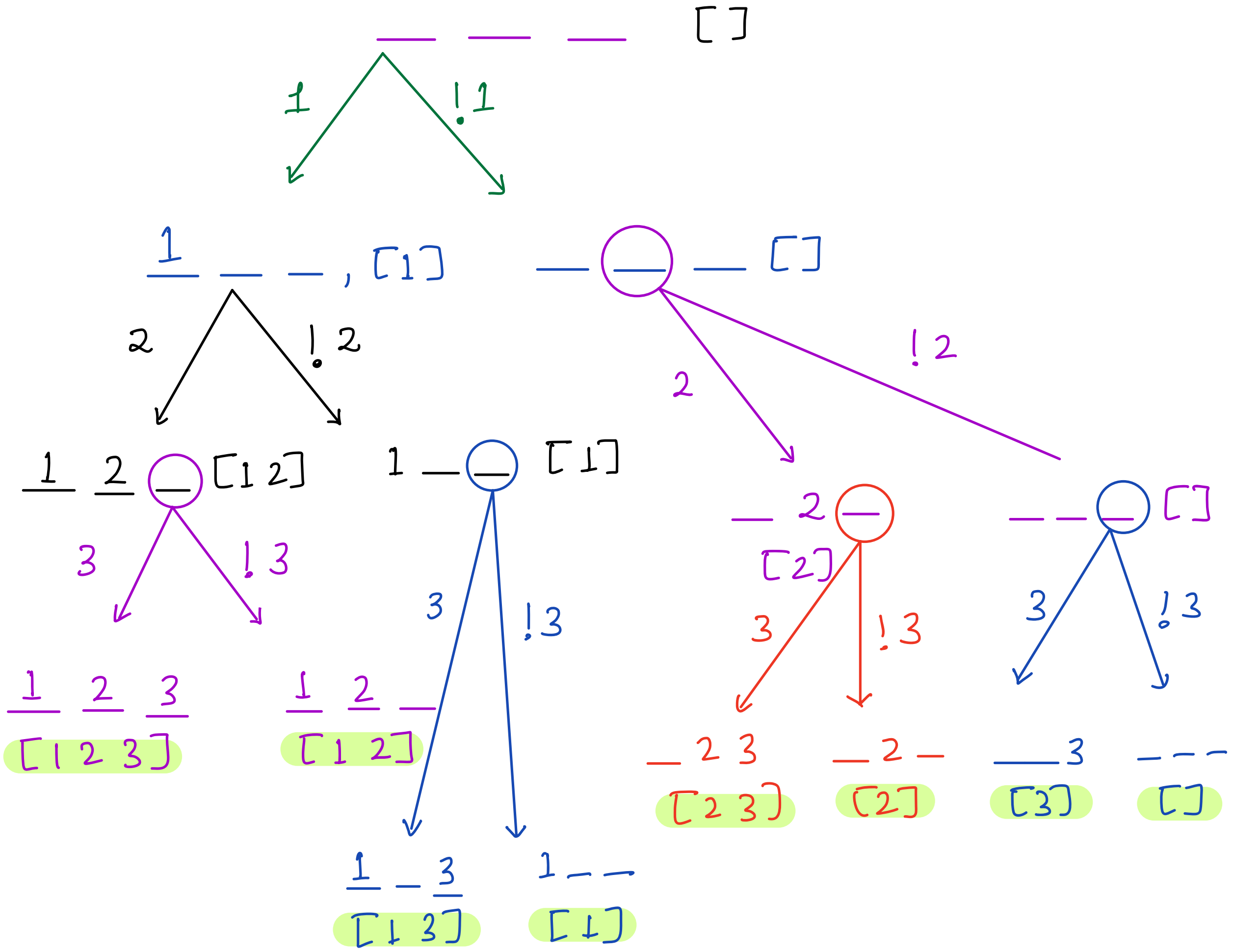
I/P A = [5 4]

subsets = [ ]
[4]
[5]
[4 5]

$n = 1$ subsets $= 2$
$n = 2$ subsets $= 4$
$n = 3$ subsets $= 8$
$n = n$ subsets $= 2^n$

Technique ***** ⟶ TAKE / DONT TAKE
select / dont select.
pick / dont pick

A = [1 2 3]



Pseudocode

```
List<List<Integer>>   subsets   // init . global

void   solve ( A[] , index , subset) {
        // Base condition
        if ( index == N) {
              subsets . add (~~subset~~)    subset.copy()
              return
        }
```

```
    // take A[index]
    subset.add (A[index])
    solve ( A , index+1 , subset )
    subset. remove () // remove last element

    // dont take A[index]
    solve ( A , index+1 , subset )
}
```

solve ([1 2 3] , 0 , [])

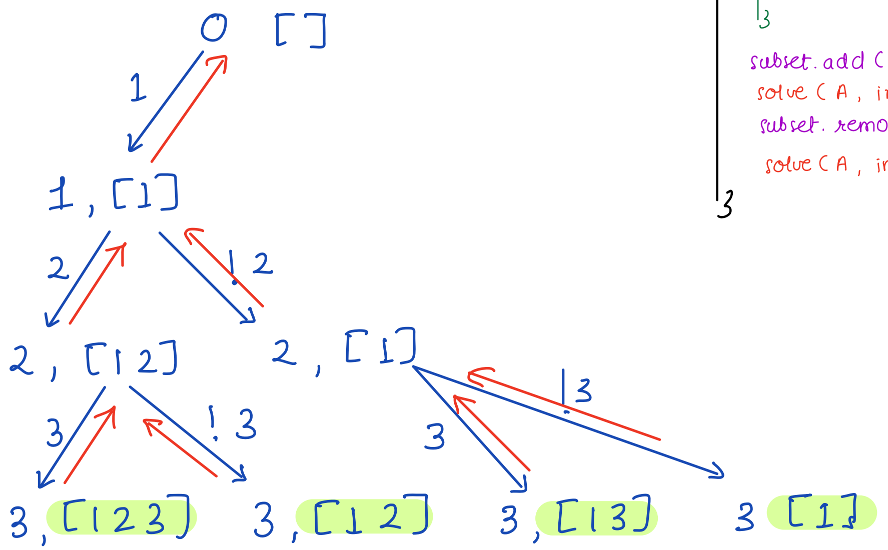
↓

subsets ⟶ [] [] [] [] [] [] [] []

$$A = \begin{array}{ccc} 0 & 1 & 2 \\ [1 & 2 & 3] \end{array}$$

```
void solve ( A[] , index , subset ) {
    if ( index == N ) {
        subsets . add ( ~~subset~~ )     subset.copy()
        return
    }

    subset.add ( A[index] )
    solve ( A , index+1 , subset )
    subset. remove () // remove last element
    solve ( A , index+1 , subset )
}
```



TC : $O(N * 2^N)$

SC : $O(N)$

Break : 22 : 50

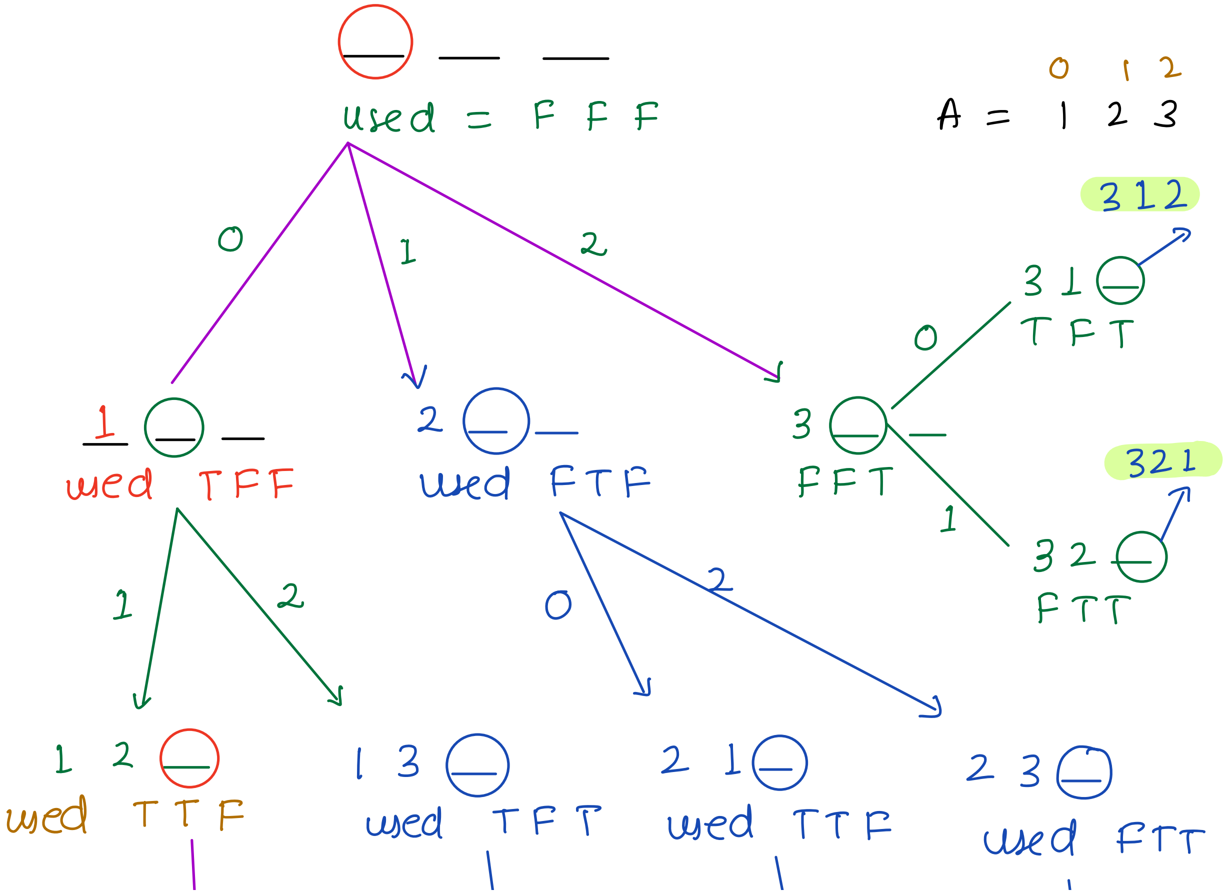
Permutation ways to arrange an array.

A = [1 2 3]

Permutations of A.

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Given an array A[] . Print all the permutation of A[] , { distinct integer } .



2 → 1 2 3 used = T T T

1 → 1 3 2 used T T T

2 → 2 1 3 used T T T

0 → 2 3 1 used T T T

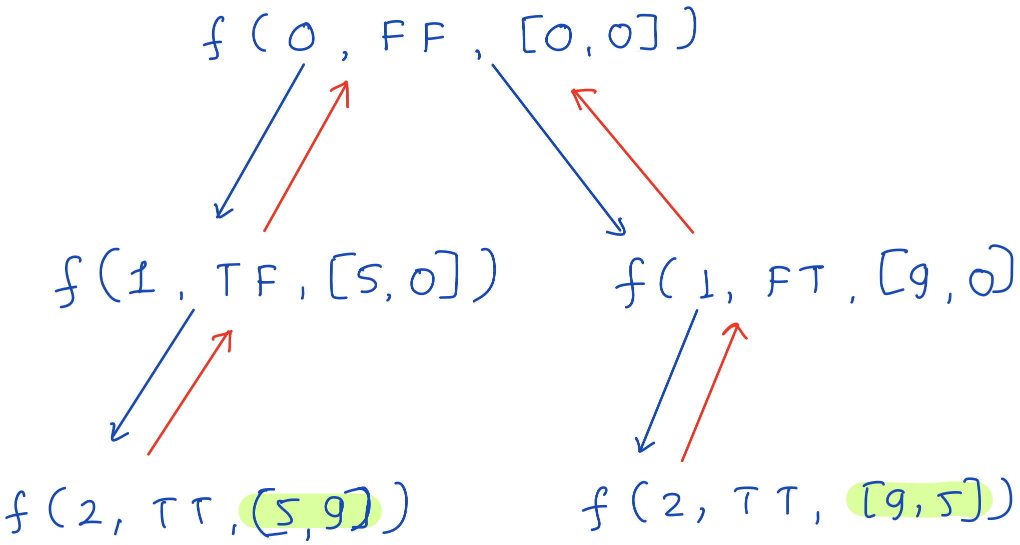
Pseudocode

```
                                    0           [F F F]   [000]
void permutation ( A[] , pos , used[] , perm[])
    if ( pos == N) {
        print ( perm )
        return
    }

    // identify free values.
    for i → 0 to N-1 {
        if (used[i] == false) {
            used[i] = true       }
            perm[pos] = A[i]     } do
            permutation ( A , pos + 1 , used , perm)
            used[i] = false      }
            perm[pos] = 0        } undo
        }
    }
}
```

0 1

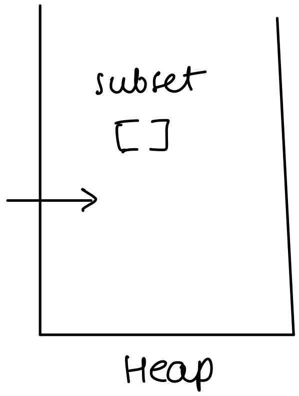
$A = [\overset{\smile}{5} \ \overset{\perp}{9}]$ used = F F



output
[5, 9]
[9, 5]

TC : $O(N \cdot N!)$

SC : $O(\underset{\text{stack}}{N} + \underset{\text{used}}{N} + \underset{\text{perm}}{N}) = O(N)$



Doubt Session

Using dynamic array.

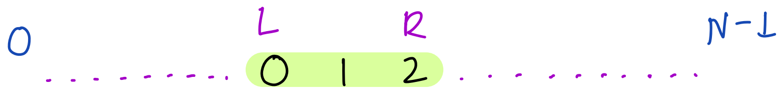
```
                                                        List
                                                      ↗
void permutation ( A[] , used[], , perm)
    if ( perm.size()= = N) {
        print ( perm)
        return
    }

    // identify free values.
    for i ⟶ 0 to N-1 {
        if (used [i]  = = false) {
            used [i]  = true              }  do
            perm.add(A[i])                }
            permutation ( A , used , perm)
            used [i]  = false             }  undo
            perm.remove ( )               }
        }                  ↘
    }                        remove last index .
}
```

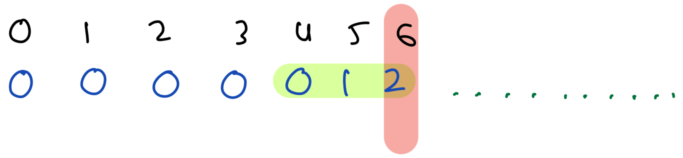
Nice Subarray Count

A = 0 1 2

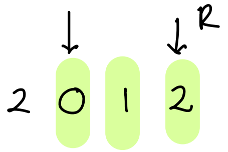
How many subarrays are there with atleast one 0, 1, 2 ?



$(l+1) * (N-R)$



2 0 1 2 → 3



ans += L+1

1 + 2 = 3

```
f → R 0 to N-1
    while subarray is nice
        left++
    }
    // subarray is not nice
```

```
        count += 1
}
```