

Preeti Kakuru

## 1. Solve the Leet code question no. 78 (Subsets)

Implement a solution that can be accepted.

Provide screen shot of your submission. Also post your source code here.

**Success** Details >

Runtime: 4 ms, faster than 92.94% of C online submissions for Subsets.

Memory Usage: 6.5 MB, less than 50.19% of C online submissions for Subsets.

Next challenges:

- Subsets II
- Generalized Abbreviation
- Letter Case Permutation
- Find Array Given Subset Sums
- Count Number of Maximum Bitwise-OR Subsets

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
02/15/2022 19:15	Accepted	4 ms	6.5 MB	c

```
1  /**
2   * Return an array of arrays of size *returnSize.
3   * The sizes of the arrays are returned as *returnColumnSizes array.
4   * Note: Both returned array and *columnSizes array must be malloced, assume caller calls
5   * free().
6   */
7
8  int** subsets(int* numbers, int numbersSize, int* returnSize, int** returnColumnSizes){
9      int length = (1<<numbersSize);
10     int** result = (int**)malloc(length*sizeof(int*));
11     int i,pos,idx,temp;
12
13     (*returnColumnSizes) = (int*)calloc(length,sizeof(int));
14
15     for(i=0;i<length;i++)
16     {
17         result[i] = (int*)malloc(numbersSize*sizeof(int));
18     }
19
20     for(i=0;i<length;i++)
21     {
22         temp = i;
23         pos = numbersSize-1;
```

Testcase Run Code Result Debugger

**Accepted** Runtime: 0 ms

Your input [1,2,3]

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

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

Diff

Run Code Submit

```
/**
```

```
* Return an array of arrays of size *returnSize.
```

```
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume caller calls
free().
```

```
*/
```

```
int** subsets(int* numbers, int numbersSize, int* returnSize, int** returnColumnSizes){
```

```
    int length = (1<<numbersSize);
```

```
    int** result = (int**)malloc(length*sizeof(int*));
```

```

int i,pos,idx,temp;

(*returnColumnSizes) = (int*)calloc(length,sizeof(int));

for(i=0;i<length;i++)
{
    result[i] = (int*)malloc(numbersSize*sizeof(int));
}

for(i=0;i<length;i++)
{
    temp = i;
    pos = numbersSize-1;
    idx = 0;
    while(temp)
    {
        if(temp&1)
            result[i][idx++] = numbers[pos];
        temp>>=1;
        pos--;
    }
    (*returnColumnSizes)[i] = idx;
}

*returnSize = length;
return result;
}

```

## 2. Solve the Leet code question no. 46 (Permutations)

Implement a solution that can be accepted.

Provide screen shot of your submission. Also post your source code here.

**Success** Details >

Runtime: 7 ms, faster than 98.29% of C online submissions for Permutations.

Memory Usage: 7.1 MB, less than 84.00% of C online submissions for Permutations.

Next challenges:

[Next Permutation](#) [Permutations II](#) [Permutation Sequence](#)

[Combinations](#)

Show off your acceptance: [f](#) [t](#) [in](#)

Time Submitted	Status	Runtime	Memory	Language
02/15/2022 19:27	Accepted	7 ms	7.1 MB	c
02/15/2022 19:20	Compile Error	N/A	N/A	c

```
1 int factorial(int n){
2     if(n==0)
3         return 1;
4     return n*factorial(n-1);
5 }
6
7 void add_curr_permutation(int** res, int *ret_size, int* nums, int n_size){
8     int i=0;
9     for(i=0;i<n_size;i++){
10         res[*ret_size][i]=nums[i];
11     }
12     (*ret_size)++;
13 }
14
15 void swap(int *a, int *b){
16     int temp=*a;
17     *a=*b;
18     *b=temp;
19 }
20
21
22
```

Testcase Run Code Result Debugger

**Accepted** Runtime: 0 ms

Your input [1,2,3]

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

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

Run Code Submit

```
int factorial(int n){
```

```
if(n==0)
```

```
return 1;
```

```
return n*factorial(n-1);
```

```
}
```

```
void add_curr_permutation(int** res, int *ret_size, int* nums, int n_size){
```

```
int i=0;
```

```
for(i=0;i<n_size;i++){
```

```
res[*ret_size][i]=nums[i];
```

```
}
```

```
(*ret_size)++;
```

```
return;
```

```
}
```

```
void swap(int *a, int *b){
```

```
int temp=*a;
```

```
*a=*b;
```

```
*b=temp;
```

```
}
```

```
// The method which recursively generates the permutations
```

```
void permute_all(int *nums, int n_size, int start, int* ret_size, int** result){
```

```
if(start>=n_size){
```

```
add_curr_permutation(result, ret_size, nums, n_size );
```

```
return;
```

```
}
```

```
int i=0;
```

```
for(i=start;i<n_size;i++){
```

```
swap(&nums[start], &nums[i]);
```

```
permute_all(nums, n_size, start+1, ret_size, result);
```

```
swap(&nums[start], &nums[i]);
```

```
}
```

```
}
```

```
int** permute(int* nums, int numsSize, int* returnSize, int** returnColumnSizes){
```

```
if(numsSize==0)
```

```
return NULL;
```

```
int res_size=factorial(numsSize); //permutations will be factorial of numsSize

int i=0;

int **result=(int**)(calloc(sizeof(int*), res_size));

*returnColumnSizes=(int*)calloc(res_size,sizeof(int));


for(i=0;i<res_size;i++){
result[i]=(int*)(calloc(sizeof(int), numsSize));
(*returnColumnSizes)[i]=numsSize;
}


*returnSize=0;

permute_all(nums, numsSize, 0, returnSize, result);

return result;


}
```