

Print all possible combinations of r elements in a given array of size n

Given an array of size n, generate and print all possible combinations of r elements in array. For example, if input array is {1, 2, 3, 4} and r is 2, then output should be {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4} and {3, 4}.

Following are two methods to do this.

Method 1 (Fix Elements and Recur)

We create a temporary array 'data[]' which stores all outputs one by one. The idea is to start from first index (index = 0) in data[], one by one fix elements at this index and recur for remaining indexes. Let the input array be {1, 2, 3, 4, 5} and r be 3. We first fix 1 at index 0 in data[], then recur for remaining indexes, then we fix 2 at index 0 and recur. Finally, we fix 3 and recur for remaining indexes. When number of elements in data[] becomes equal to r (size of a combination), we print data[].

Following diagram shows recursion tree for same input.

Google™ Custom Search



GeeksforGeeks

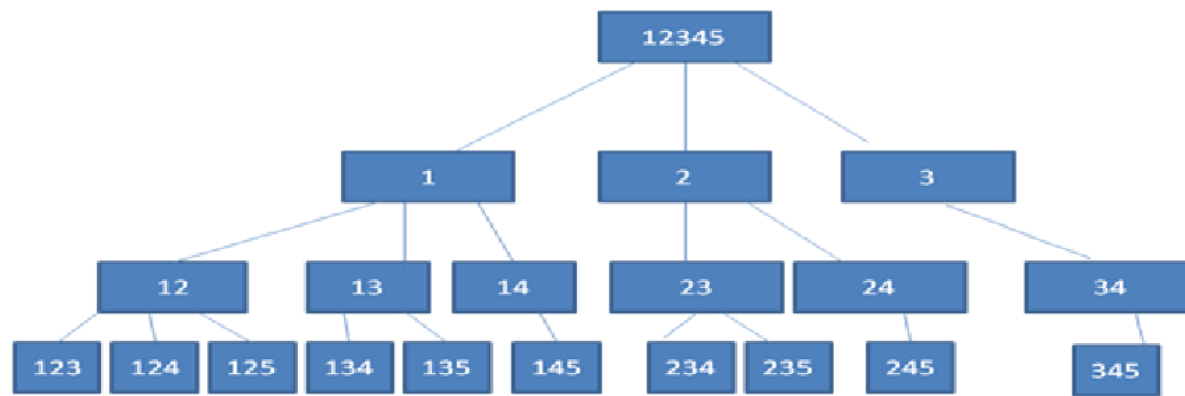


53,525 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)



Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding "extern" keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Following is C++ implementation of above approach.

```

// Program to print all combination of size r in an array of size n
#include <stdio.h>
void combinationUtil(int arr[], int data[], int start, int end, int index)
{
    // The main function that prints all combinations of size r
    // in arr[] of size n. This function mainly uses combinationUtil()
    void printCombination(int arr[], int n, int r)
    {
        // A temporary array to store all combination one by one
        int data[r];

        // Print all combination using temporary array 'data[]'
        combinationUtil(arr, data, 0, n-1, 0, r);
    }

    /* arr[] ---> Input Array
       data[] ---> Temporary array to store current combination
       start & end ---> Starting and Ending indexes in arr[]
       index ---> Current index in data[]
       r ---> Size of a combination to be printed */
    void combinationUtil(int arr[], int data[], int start, int end, int index)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
        {
            for (int j=0; j<r; j++)
                printf("%d ", data[j]);
            printf("\n");
            return;
        }
    }
}

```

```
// replace index with all possible elements. The condition
// "end-i+1 >= r-index" makes sure that including one element
// at index will make a combination with remaining elements
// at remaining positions
for (int i=start; i<=end && end-i+1 >= r-index; i++)
{
    data[index] = arr[i];
    combinationUtil(arr, data, i+1, end, index+1, r);
}
}
```

```
// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
}
```

Output:

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

How to handle duplicates?

Note that the above method doesn't handle duplicates. For example, if input array is {1, 2, 1} and r is 2, then the program prints {1, 2} and {2, 1} as two different combinations. We can avoid duplicates by adding following two additional things to above code.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines at the end of for loop in combinationUtil()

```
// Since the elements are sorted, all occurrences of an element
```

Structure Member Alignment, Padding and

Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST



```
// must be together
while (arr[i] == arr[i+1])
    i++;
```

See [this](#) for an implementation that handles duplicates.

Method 2 (Include and Exclude every element)

Like the above method, We create a temporary array data[]. The idea here is similar to [Subset Sum Problem](#). We one by one consider every element of input array, and recur for two cases:

- 1) The element is included in current combination (We put the element in data[] and increment next available index in data[])
- 2) The element is excluded in current combination (We do not put the element and do not change index)

When number of elements in data[] become equal to r (size of a combination), we print it.

This method is mainly based on [Pascal's Identity](#), i.e. $nC_r = n-1C_r + n-1C_{r-1}$

Following is C++ implementation of method 2.

```
// Program to print all combination of size r in an array of size n
#include<stdio.h>
void combinationUtil(int arr[],int n,int r,int index,int data[],int i)

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[r];

    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, n, r, 0, data, 0);
}

/* arr[]    ---> Input Array
   n        ---> Size of input array
   r        ---> Size of a combination to be printed
   index    ---> Current index in data[]
   data[]   ---> Temporary array to store current combination
```



Subscribe

Recent Comments

Abhi You live US or India?

[Google \(Mountain View\) interview](#) · 12 minutes ago

[Aman](#) Hi, Why arent we checking for conditions...

[Write a C program to Delete a Tree.](#) · 52 minutes ago

kzs please provide solution for the problem...

[Backtracking | Set 2 \(Rat in a Maze\)](#) · 56 minutes ago

[Sanjay Agarwal](#) bool

tree::Root_to_leaf_path_given_sum(tree...
Root to leaf path sum equal to a given number · 1 hour ago

[GOPI GOPINATH](#) @admin Highlight this sentence "We can easily...

[Count trailing zeroes in factorial of a number](#) · 1 hour ago

[newCoder3006](#) If the array contains negative numbers also. We...

```

i      ---> index of current element in arr[]      */
void combinationUtil(int arr[], int n, int r, int index, int data[], int i)
{
    // Current combination is ready, print it
    if (index == r)
    {
        for (int j=0; j<r; j++)
            printf("%d ", data[j]);
        printf("\n");
        return;
    }

    // When no more elements are there to put in data[]
    if (i >= n)
        return;

    // current is included, put next at next location
    data[index] = arr[i];
    combinationUtil(arr, n, r, index+1, data, i+1);

    // current is excluded, replace it with next (Note that
    // i+1 is passed, but index is not changed)
    combinationUtil(arr, n, r, index, data, i+1);
}

```

// Driver program to test above functions

```

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
    return 0;
}

```

Output:

```

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4

```

Find subarray with given sum · 1 hour ago

AdChoices

► [C++ Code](#)

► [Java Source Code](#)

► [Java Array](#)

AdChoices

► [An Array](#)

► [Int C++](#)

► [C++ Data String](#)

AdChoices

► [An Array](#)

► [Int C++](#)

► [Combinations](#)

```
2 3 5
2 4 5
3 4 5
```

How to handle duplicates in method 2?

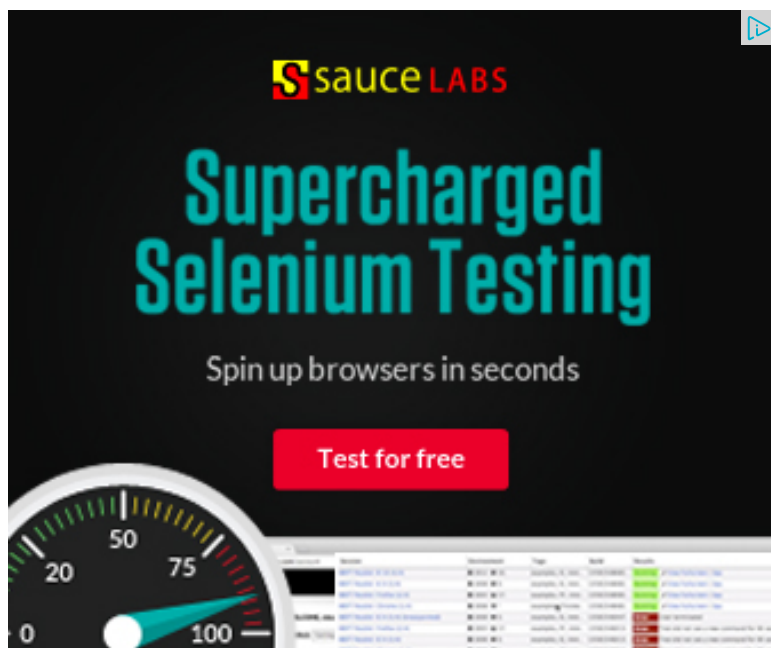
Like method 1, we can follow two things to handle duplicates.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines between two recursive calls of combinationUtil() in combinationUtil()

```
// Since the elements are sorted, all occurrences of an element
// must be together
while (arr[i] == arr[i+1])
    i++;
```

See [this](#) for an implementation that handles duplicates.

This article is contributed by **Bateesh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



Related Topics:

- Remove minimum elements from either side such that $2 \times \text{min}$ becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3



44



Tweet

3



1

Writing code in comment? Please use ideone.com and share the link here.

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team