GeeksforGeeks
A computer science portal for geeks

Practice   GATE CS   Placements   GeeksQuiz

Login/Register

# 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.
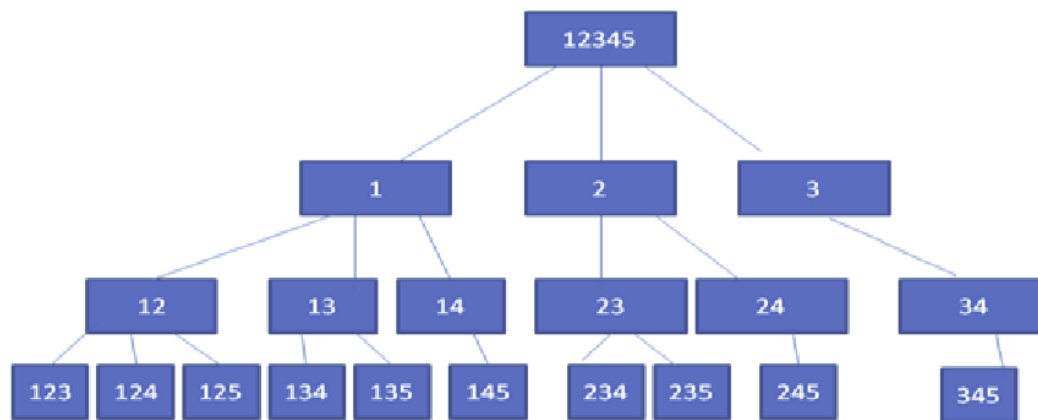
Following is C++ implementation of above approach.

C

```c
// 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, int r);

// The main function that prints all combinations of size r
```

```c
// 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, data, 0, n-1, 0, r);
}

/* arr[]  ---> Input Array
   data[] ---> Temporary array to store current combination
   start & end ---> Staring 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, int r)
{
    // 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);
}
```

Run on IDE

Java

```java
// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

    /* arr[]  ---> Input Array
       data[] ---> Temporary array to store current combination
       start & end ---> Staring and Ending indexes in arr[]
       index   ---> Current index in data[]
       r ---> Size of a combination to be printed */
    static void combinationUtil(int arr[], int data[], int start,
                                int end, int index, int r)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
```

```java
        {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            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);
        }
    }

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

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

    /*Driver function to check for above function*/
    public static void main (String[] args) {
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}

/* This code is contributed by Devesh Agrawal */
```

Run on IDE

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
        // 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. $n_{c_r} = n\text{-}1_{c_r} + n\text{-}1_{c_{r-1}}$

Following is C++ implementation of method 2.

C

```c
// 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
   i      ---> index of current element in arr[]      */
void combinationUtil(int arr[], int n, int r, int index, int data[], int i)
{
    // Current cobination 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;
}
```

Run on IDE

Java

```
// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

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

    // The main function that prints all combinations of size r
    // in arr[] of size n. This function mainly uses combinationUtil()
    static void printCombination(int arr[], int n, int r)
```

```
    {
        // A temporary array to store all combination one by one
        int data[]=new int[r];

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

    /*Driver function to check for above function*/
    public static void main (String[] args) {
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}
/* This code is contributed by Devesh Agrawal */
```

Run on IDE

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 in method 2?

Like method 1, we can following 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

GATE CS Corner   Company Wise Coding Practice

Mathematical MathematicalAlgo Recursion

Related Posts:

- N-th multiple in sorted list of multiples of two numbers
- Multiplicative order
- Sum of product of x and y such that floor(n/x) = y
- Primitive root of a prime number n modulo n
- k-th prime factor of a given number
- Partition a number into two divisble parts
- Double factorial
- Count pairs of natural numbers with GCD equal to given number

(Login to Rate and Mark)

4.1 Average Difficulty : **4.1/5.0**
Based on **65** vote(s)

☐ Add to TODO List

☐ Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org generate link and share the link here.

Load Comments Share this post!