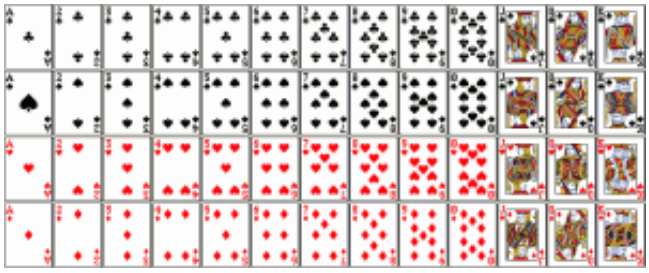
Shuffle a given array

Given an array, write a program to generate a random permutation of array elements. This question is also asked as “shuffle a deck of cards” or “randomize a given array”.



Let the given array be *arr[]*. A simple solution is to create an auxiliary array *temp[]* which is initially a copy of *arr[]*. Randomly select an element from *temp[]*, copy the randomly selected element to *arr[0]* and remove the selected element from *temp[]*. Repeat the same process n times and keep copying elements to *arr[1], arr[2], … .* The time complexity of this solution will be O(n^2).

Fisher–Yates shuffle Algorithm works in O(n) time complexity. The assumption here is, we are given a function rand() that generates random number in O(1) time.
The idea is to start from the last element, swap it with a randomly selected element from the whole array (including last). Now consider the array from 0 to n-2 (size reduced by 1), and repeat the process till we hit the first element.

Following is the detailed algorithm

```
To shuffle an array a of n elements (indices 0..n-1):
  for i from n - 1 downto 1 do
       j = random integer with 0 <= j <= i
       exchange a[j] and a[i]
```

Following is C++ implementation of this algorithm.

```
// C Program to shuffle a given array
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A utility function to swap to integers
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// A utility function to print an array
void printArray (int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// A function to generate a random permutation of arr[]
void randomize ( int arr[], int n )
{
    // Use a different seed value so that we don't get s
    // result each time we run this program
    srand ( time(NULL) );

    // Start from the last element and swap one by one.
    // need to run for the first element that's why i >
    for (int i = n-1; i > 0; i--)
    {
        // Pick a random index from 0 to i
        int j = rand() % (i+1);

        // Swap arr[i] with the element at random index
        swap(&arr[i], &arr[j]);
    }
}

// Driver program to test above function.
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n = sizeof(arr)/ sizeof(arr[0]);
    randomize (arr, n);
    printArray(arr, n);
```

```
    return 0;
}
```

Output:

```
7 8 4 6 3 1 2 5
```

The above function assumes that rand() generates a random number.

Time Complexity: O(n), assuming that the function rand() takes O(1) time.

How does this work?
The probability that ith element (including the last one) goes to last position is 1/n, because we randomly pick an element in first iteration.

The probability that ith element goes to second last position can be proved to be 1/n by dividing it in two cases.
Case 1: i = n-1 (index of last element):
The probability of last element going to second last position is = (probability that last element doesn't stay at its original position) x (probability that the index picked in previous step is picked again so that the last element is swapped)
So the probability = ((n-1)/n) x (1/(n-1)) = 1/n
Case 2: 0 < i < n-1 (index of non-last):
The probability of ith element going to second position = (probability that ith element is not picked in previous iteration) x (probability that ith element is picked in this iteration)
So the probability = ((n-1)/n) x (1/(n-1)) = 1/n

We can easily generalize above proof for any other position.