Randomized Algorithms

[**Learn more about Randomized Algorithms in DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=articles+random_algo_lp+header_link_click&utm_campaign=dsa+course+tracker)

[**Recent articles on Randomized Algorithms**](https://www.geeksforgeeks.org/category/dsa/algorithm/randomized/?ref=random_algo_lp)

**What is Randomized Algorithm?**

*An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm.*

For example, in Randomized Quick Sort, we use a random number to pick the next pivot (or we randomly shuffle the array). Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms.

**Topics :**

* [Introduction](https://www.geeksforgeeks.org/randomized-algorithms/#introduction)
* [Problems on Randomized Algorithms](https://www.geeksforgeeks.org/randomized-algorithms/#standard)

**Introduction:**

1. [Random Variables](https://www.geeksforgeeks.org/random-variable/)
2. [Binomial Random Variables](https://www.geeksforgeeks.org/binomial-random-variables/)
3. [Randomized Algorithms | Set 0 (Mathematical Background)](https://www.geeksforgeeks.org/randomized-algorithms-set-0-mathematical-background/)
4. [Randomized Algorithms | Set 1 (Introduction and Analysis)](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/)
5. [Randomized Algorithms | Set 2 (Classification and Applications)](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/)
6. [Randomized Algorithms | Set 3 (1/2 Approximate Median)](https://www.geeksforgeeks.org/randomized-algorithms-set-3-12-approximate-median/)

**Problems on Randomized Algorithms:**

* **Easy:**
  1. [Write a function that generates one of 3 numbers according to given probabilities](https://www.geeksforgeeks.org/write-a-function-to-generate-3-numbers-according-to-given-probabilities/)
  2. [Generate 0 and 1 with 25% and 75% probability](https://www.geeksforgeeks.org/generate-0-1-25-75-probability/)
  3. [Implement rand3() using rand2()](https://www.geeksforgeeks.org/implement-rand3-using-rand2/)
  4. [Birthday Paradox](https://www.geeksforgeeks.org/birthday-paradox/)
  5. [Expectation or expected value of an array](https://www.geeksforgeeks.org/expectation-expected-value-array/)
  6. [Shuffle a deck of cards](https://www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/)
  7. [Program to generate CAPTCHA and verify user](https://www.geeksforgeeks.org/program-generate-captcha-verify-user/)
  8. [Find an index of maximum occurring element with equal probability](https://www.geeksforgeeks.org/find-index-maximum-occurring-element-equal-probability/)
  9. [Randomized Binary Search Algorithm](https://www.geeksforgeeks.org/randomized-binary-search-algorithm/)
* **Medium:**
  1. [Make a fair coin from a biased coin](https://www.geeksforgeeks.org/print-0-and-1-with-50-probability/)
  2. [Shuffle a given array using Fisher–Yates shuffle Algorithm](https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/)
  3. [Expected Number of Trials until Success](https://www.geeksforgeeks.org/expected-number-of-trials-before-success/)
  4. [Strong Password Suggester Program](https://www.geeksforgeeks.org/strong-password-suggester-program/)
  5. [QuickSort using Random Pivoting](https://www.geeksforgeeks.org/quicksort-using-random-pivoting/)
  6. [Operations on Sparse Matrices](https://www.geeksforgeeks.org/operations-sparse-matrices/)
  7. [Estimating the value of Pi using Monte Carlo](https://www.geeksforgeeks.org/estimating-value-pi-using-monte-carlo/)
  8. [Implement rand12() using rand6() in one line](https://www.geeksforgeeks.org/implement-rand12-using-rand6-in-one-line/)
* **Hard:**
  1. [Generate integer from 1 to 7 with equal probability](https://www.geeksforgeeks.org/generate-integer-from-1-to-7-with-equal-probability/)
  2. [Implement random-0-6-Generator using the given random-0-1-Generator](https://www.geeksforgeeks.org/implement-random-0-6-generator-using-the-given-random-0-1-generator/)
  3. [Select a random number from stream, with O(1) space](https://www.geeksforgeeks.org/select-a-random-number-from-stream-with-o1-space/)
  4. [Random number generator in arbitrary probability distribution fashion](https://www.geeksforgeeks.org/random-number-generator-in-arbitrary-probability-distribution-fashion/)
  5. [Reservoir Sampling](https://www.geeksforgeeks.org/reservoir-sampling/)
  6. [Linearity of Expectation](https://www.geeksforgeeks.org/linearity-of-expectation/)
  7. [Introduction and implementation of Karger’s algorithm for Minimum Cut](https://www.geeksforgeeks.org/introduction-and-implementation-of-kargers-algorithm-for-minimum-cut/)
  8. [Select a Random Node from a Singly Linked List](https://www.geeksforgeeks.org/select-a-random-node-from-a-singly-linked-list/)
  9. [Select a Random Node from a tree with equal probability](https://www.geeksforgeeks.org/select-random-node-tree-equal-probability/)
  10. [Freivald’s Algorithm to check if a matrix is product of two](https://www.geeksforgeeks.org/freivalds-algorithm/)
  11. [Random Acyclic Maze Generator with given Entry and Exit point](https://www.geeksforgeeks.org/random-acyclic-maze-generator-with-given-entry-and-exit-point/)

**Recomended:**

* [**Learn Data Structure and Algorithms | DSA Tutorial**](https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial?utm_source=Website&utm_medium=Landing+Page+Click&utm_campaign=DSA+Page+Tracker&utm_id=DSA-Page-Tracker&utm_term=DSA+Page+Promo&utm_content=Course+Page)

**Mathematics | Random Variables**

Random variable is basically a function which maps from the set of sample space to set of real numbers. The purpose is to get an idea about result of a particular situation where we are given probabilities of different outcomes. See below example for more clarity.

**Example :**

Suppose that two coins (unbiased) are tossed

X = number of heads. [X is a random variable   
 or function]

Here, the sample space S = {HH, HT, TH, TT}.

The output of the function will be :  
 X(HH) = 2  
 X(HT) = 1  
 X(TH) = 1  
 X(TT) = 0

**Formal definition :**

**X: S -> R**

X = random variable (It is usually denoted using capital letter)

S = set of sample space

R = set of real numbers

Suppose a random variable X takes m different values i.e. sample space X = {x1, x2, x3………xm} with probabilities P(X=xi) = pi; where 1 ≤ i ≤ m. The probabilities must satisfy the following conditions :

1. 0 <= pi <= 1; where 1 <= i <= m
2. p1 + p2 + p3 + ……. + pm = 1 Or we can say 0 ≤ pi ≤ 1 and ∑pi = 1.

Hence possible values for random variable X are 0, 1, 2.

X = {0, 1, 2} where m = 3

P(X=0) = probability that number of heads is 0 = P(TT) = 1/2\*1/2 = 1⁄4.

P(X=1) = probability that number of heads is 1 = P(HT | TH) = 1/2\*1/2 + 1/2\*1/2 = 1⁄2.

P(X=2) = probability that number of heads is 2 = P(HH) = 1/2\*1/2 = 1⁄4.

Here, you can observe that

1) 0 ≤ p1, p2, p3 ≤ 1

2) p1 + p2 + p3 = 1/4 + 2/4 + 1/4 = 1

**Example :**

Suppose a dice is thrown X = outcome of the dice. Here, the sample space S = {1, 2, 3, 4, 5, 6}. The output of the function will be:

1. P(X=1) = 1/6
2. P(X=2) = 1/6
3. P(X=3) = 1/6
4. P(X=4) = 1/6
5. P(X=5) = 1/6
6. P(X=6) = 1/6

See if there is any random variable then there must be some distribution associated with it.

RandomVariable

**Discrete Random Variable:**

A random variable X is said to be discrete if it takes on finite number of values. The probability function associated with it is said to be PMF = Probability mass function.

P(xi) = Probability that X = xi = PMF of X = pi.

1. 0 ≤ pi ≤ 1.
2. ∑pi = 1 where sum is taken over all possible values of x.

The examples given above are discrete random variables.

**Example:-** Let S = {0, 1, 2}

randon_var_2

**Find the value of P (X=0)**:

**Sol:-** We know that sum of all probabilities is equals to 1.

==> p1 + p2 + p3 = 1

==> p1 + 0.3 + 0.5 = 1

==> p1 = 0.2

**Continuous Random Variable:**

A random variable X is said to be continuous if it takes on infinite number of values. The probability function associated with it is said to be PDF = Probability density function

**PDF:** If X is continuous random variable.

P (x < X < x + dx) = f(x)\*dx.

1. 0 ≤ f(x) ≤ 1; for all x
2. ∫ f(x) dx = 1 over all values of x

Then P (X) is said to be PDF of the distribution.

**Example:- Compute the value of P (1 < X < 2).**

Such that f(x) = k\*x^3; 0 ≤ x ≤ 3  
 = 0; otherwise  
f(x) is a density function

**Solution:-** If a function f is said to be density function, then sum of all probabilities is equals to 1. Since it is a continuous random variable Integral value is 1 overall sample space s.

==> K\*[x^4]/4 = 1 [Note that [x^4]/4 is integral of x^3]

==> K\*[3^4 – 0^4]/4 = 1

==> K = 4/81

The value of P (1 < X < 2) = k\*[X^4]/4 = 4/81 \* [16-1]/4 = 15/81.

**Next Topic :**

[Linearity of Expectation](https://www.geeksforgeeks.org/linearity-of-expectation/)

**Binomial Random Variables**

In this post, we’ll discuss Binomial Random Variables.

**Prerequisite :**[Random Variables](https://www.geeksforgeeks.org/random-variable/)

A specific type of **discrete** random variable that counts how often a particular event occurs in a fixed number of tries or trials.

For a variable to be a binomial random variable, ALL of the following conditions must be met:

1. There are a fixed number of trials (a fixed sample size).
2. On each trial, the event of interest either occurs or does not.
3. The probability of occurrence (or not) is the same on each trial.
4. Trials are independent of one another.

Mathematical Notations

n = number of trials  
p = probability of success in each trial  
k = number of success in n trials

Now we try to find out the probability of k success in n trials.

Here the probability of success in each trial is p independent of other trials.

So we first choose k trials in which there will be a success and in rest n-k trials there will be a failure. Number of ways to do so is

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image003.png

Since all n events are independent, hence the probability of k success in n trials is equivalent to multiplication of probability for each trial.

Here its k success and n-k failures, So probability for each way to achieve k success and n-k failure is

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image004.png

Hence final probability is

(number of ways to achieve k success  
 and n-k failures)  
 \*  
(probability for each way to achieve k  
 success and n-k failure)

Then Binomial Random Variable Probability is given by:

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image005.png

Let X be a binomial random variable with the number of trials n and probability of success in each trial be p.

Expected number of success is given by

E[X] = np

Variance of number of success is given by

Var[X] = np(1-p)

**Example 1** : Consider a random experiment in which a biased coin (probability of head = 1/3) is thrown for 10 times. Find the probability that the number of heads appearing will be 5.

Solution :

Let X be binomial random variable   
with n = 10 and p = 1/3  
P(X=5) = ?

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image005.png

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image006.png

Here is the implementation for the same

# Python3 program to compute Binomial

# Probability

# function to calculate nCr i.e.,

# number of ways to choose r out

# of n objects

**def** nCr(n, r):

    # Since nCr is same as nC(n-r)

    # To decrease number of iterations

**if** (r > n **/** 2):

        r **=** n **-** r;

    answer **=** 1;

**for** i **in** range(1, r **+** 1):

        answer **\*=** (n **-** r **+** i);

        answer **/=** i;

**return** answer;

# function to calculate binomial r.v.

# probability

**def** binomialProbability(n, k, p):

**return** (nCr(n, k) **\*** pow(p, k) **\***

                        pow(1 **-** p, n **-** k));

# Driver code

n **=** 10;

k **=** 5;

p **=** 1.0 **/** 3;

probability **=** binomialProbability(n, k, p);

**print**("Probability of", k,

      "heads when a coin is tossed", end **=** " ");

print(n, "times where probability of each head is",

                                      round(p, 6));

print("is = ", round(probability, 6));

# This code is contributed by mits

**Output:**

Probability of 5 heads when a coin is tossed 10 times where probability of each head is 0.333333  
 is = 0.136565

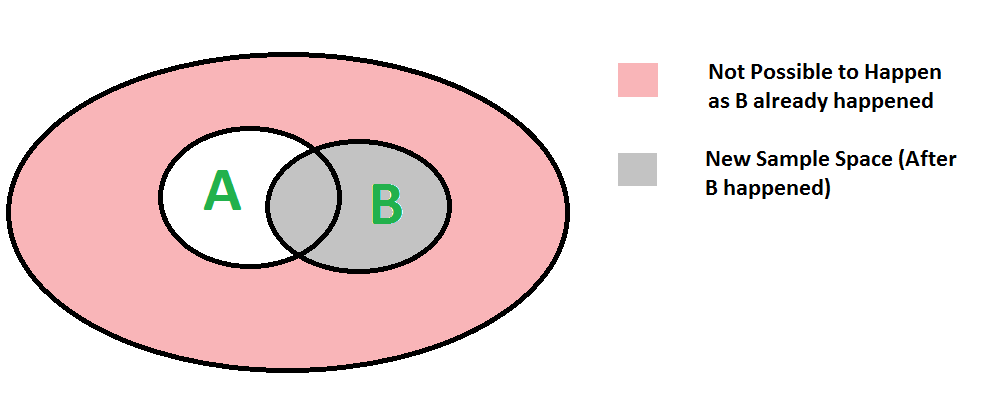
**Reference** :

[stat200](https://onlinecourses.science.psu.edu/stat200/node/37)

**Randomized Algorithms | Set 0 (Mathematical Background)**

**Conditional Probability** Conditional probability P(A | B) indicates the probability of even ‘A’ happening given that the even B happened.

We can easily understand above formula using below diagram. Since B has already happened, the sample space reduces to B. So the probability of A happening becomes P(A ∩ B) divided by P(B)



Below is [Bayes’s formula](https://www.geeksforgeeks.org/bayess-theorem-for-conditional-probability/) for conditional probability.

The formula provides relationship between P(A|B) and P(B|A). It is mainly derived from conditional probability formula discussed in the [previous post](https://www.geeksforgeeks.org/conditional-probability/).

Consider the below formulas for conditional probabilities P(A|B) and P(B|A)

Since P(B ∩ A) = P(A ∩ B), we can replace P(A ∩ B) in first formula with P(B|A)P(A)

After replacing, we get the given formula. Refer [this](https://www.geeksforgeeks.org/bayess-theorem-for-conditional-probability/) for examples of Bayes’s formula.

**Random Variables:**

A random variable is actually a function that maps outcome of a random event (like coin toss) to a real value.

Example :

Coin tossing game :   
A player pays 50 bucks if result of coin  
toss is "Head"

The person gets 50 bucks if the result is  
Tail.

A random variable profit for person can   
be defined as below :

Profit = +50 if Head  
 -50 if Tail

Generally gambling games are not fair for players,   
the organizer takes a share of profit for all   
arrangements. So expected profit is negative for   
a player in gambling and positive for the organizer.   
That is how organizers make money.

**Expected Value of Random Variable :**

Expected value of a random variable R can be defined as following

E[R] = r1\*p1 + r2\*p2 + ... rk\*pk   
   
 ri ==> Value of R with probability pi

Expected value is basically sum of product of following two terms (for all possible events)

a) Probability of an event.

b) Value of R at that even

**Example 1:**  
In above example of coin toss,  
Expected value of profit = 50 \* (1/2) +   
 (-50) \* (1/2)  
 = 0

**Example 2:**  
Expected value of six faced dice throw is   
 = 1\*(1/6) + 2\*(1/6) + .... + 6\*(1/6)  
 = 3.5

**Linearity of Expectation:**

Let R1 and R2 be two discrete random variables on some probability space, then

E[R1 + R2] = E[R1] + E[R2]

For example, expected value of sum for 3 dice throws is = 3 \* 7/2 = 7

Refer [this](https://www.geeksforgeeks.org/linearity-of-expectation/) for more detailed explanation and examples.

**Expected Number of Trials until Success**

If probability of success is p in every trial, then expected number of trials until success is 1/p. For example, consider 6 faced fair dice is thrown until a ‘5’ is seen as result of dice throw. The expected number of throws before seeing a 5 is 6. Note that 1/6 is probability of getting a 5 in every trial. So number of trials is 1/(1/6) = 6.

As another example, consider a QuickSort version that keeps on looking for pivots until one of the middle n/2 elements is picked. The expected time number of trials for finding middle pivot would be 2 as probability of picking one of the middle n/2 elements is 1/2. This example is discussed in more detail in [Set 1](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/).

Refer [this](https://www.geeksforgeeks.org/expected-number-of-trials-before-success/) for more detailed explanation and examples.

**More on Randomized Algorithms:**

1. [Randomized Algorithms | Set 1 (Introduction and Analysis)](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/)
2. [Randomized Algorithms | Set 2 (Classification and Applications)](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/)
3. [Randomized Algorithms | Set 3 (1/2 Approximate Median)](https://www.geeksforgeeks.org/randomized-algorithms-set-3-12-approximate-median/)

[All Randomized Algorithm Topics](https://www.geeksforgeeks.org/category/algorithm/randomized/)

**Randomized Algorithms | Set 1 (Introduction and Analysis)**

**What is a Randomized Algorithm?**

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called a Randomized Algorithm. For example, in Randomized Quick Sort, we use a random number to pick the next pivot (or we randomly shuffle the array). And in [Karger’s algorithm](https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/), we randomly pick an edge.

**How to analyse Randomized Algorithms?**

Some randomized algorithms have deterministic time complexity. For example, [this](https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/) implementation of Karger’s algorithm has time complexity is O(E). Such algorithms are called [Monte Carlo Algorithms](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/) and are easier to analyse for worst case.

On the other hand, time complexity of other randomized algorithms (other than Las Vegas) is dependent on value of random variable. Such Randomized algorithms are called [Las Vegas Algorithms](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/). These algorithms are typically analysed for expected worst case. To compute expected time taken in worst case, all possible values of the used random variable needs to be considered in worst case and time taken by every possible value needs to be evaluated. Average of all evaluated times is the expected worst case time complexity. Below facts are generally helpful in analysis os such algorithms.

[Linearity of Expectation](https://www.geeksforgeeks.org/linearity-of-expectation/)

[Expected Number of Trials until Success.](https://www.geeksforgeeks.org/expected-number-of-trials-before-success/)

For example consider below a randomized version of QuickSort.

A **Central Pivot** is a pivot that divides the array in such a way that one side has at-least 1/4 elements.

// Sorts an array arr[low..high]  
**randQuickSort(arr[], low, high)**

**1.** If low >= high, then EXIT.

**2.** While pivot 'x' is not a Central Pivot.  
 (i) Choose uniformly at random a number from [low..high].   
 Let the randomly picked number number be **x**.  
 (ii) Count elements in arr[low..high] that are smaller   
 than arr[x]. Let this count be **sc**.  
 (iii) Count elements in arr[low..high] that are greater   
 than arr[x]. Let this count be **gc**.  
 (iv) Let **n** = (high-low+1). If sc >= n/4 and  
 gc >= n/4, then x is a central pivot.

**3.** Partition arr[low..high] around the pivot x.

**4.** // Recur for smaller elements  
 randQuickSort(arr, low, sc-1)

**5.** // Recur for greater elements  
 randQuickSort(arr, high-gc+1, high)

The important thing in our analysis is, time taken by step 2 is O(n).

**How many times while loop runs before finding a central pivot?**

The probability that the randomly chosen element is central pivot is 1/n.

Therefore, expected number of times the while loop runs is n (See [this](https://www.geeksforgeeks.org/expected-number-of-trials-before-success/) for details)

Thus, the expected time complexity of step 2 is O(n).

**What is overall Time Complexity in Worst Case?**

In worst case, each partition divides array such that one side has n/4 elements and other side has 3n/4 elements. The worst case height of recursion tree is Log 3/4 n which is O(Log n).

T(n) < T(n/4) + T(3n/4) + O(n)  
T(n) < 2T(3n/4) + O(n)

Solution of above recurrence is O(n Log n)

Note that the above randomized algorithm is not the best way to implement randomized Quick Sort. The idea here is to simplify the analysis as it is simple to analyse.

Typically, randomized Quick Sort is implemented by randomly picking a pivot (no loop). Or by shuffling array elements. Expected worst case time complexity of this algorithm is also O(n Log n), but analysis is complex, the MIT prof himself mentions same in his lecture [here](https://www.youtube.com/watch?v=852wJdsgl2I).

Example :

**import** random

**import** time

**def** find\_solution(n):

  # seed the random number generator with the current time

  random.seed(time.time())

  # randomly select a number between 1 and n and return it as the solution

**return** random.randint(1, n)

**def** main():

  n **=** 10  # the range of possible solutions is 1 to n

  print("Solution:", find\_solution(n))

**if** \_\_name\_\_ **==** '\_\_main\_\_':

  main()

**Output**

Solution: 10

[Randomized Algorithms | Set 2 (Classification and Applications)](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/)

**References:**

<http://www.tcs.tifr.res.in/~workshop/nitrkl_igga/randomized-lecture.pdf>

**Randomized Algorithms | Set 2 (Classification and Applications)**

We strongly recommend to refer below post as a prerequisite of this. [Randomized Algorithms | Set 1 (Introduction and Analysis)](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/)

**Classification**

Randomized algorithms are classified in two categories.

**Las Vegas:**

A Las Vegas algorithm were introduced by Laszio babai in 188.

A Las Vegas algorithm is an algorithm which uses randomness, but does not gives guarantees that the solution obtained for given problem is corroct or not. It takes the risk with resources used. A quick-sort algorithm is a simple example of Las-Vegas algorithm. To sort the given array of n numbers quickly we use the quick sort algorithm. For that we find out central element which is also called as pivot element and each element is compared with this pivot element. Sorting is done in less time or it requires more time is dependent on how we select the pivot element. To pick the pivot element randomly we can use Las-Vegas algorithm.

**Definition:**

A randomized algorithm that always produce correct result with only variation from one aun to another being its running time is known as Las-Vegas algorithm.

OR

A randomized algorithm which always produces a correct result or it informs about the failure is known as Las-Vegas algorithm.

OR

A Las-Vegas algorithm take the risk with the resources used for computation but it does ot take risk with the result i.e. it gives correct and expected output for the given problem.

 Let us consider the above example of quick sort algorithm. In this algorithm we choose the pivot element randomly. But the result of this problem is always a sortcd array. A Las-Vcgas agoritim is having one restriction i.e. the solution for the given problem can be found out in finite time. In this algorithm the numbers of possible solutions arc limited. Thc actual solution is complex in nature or complicated to calculate but it is casy to verify the corectness of candidate solution.

These algorithms always produce correct or optimum result. Time complexity of these algorithms is based on a random value and time complexity is evaluated as expected value. For example, Randomized QuickSort always sorts an input array and [expected worst case time complexity of QuickSort is O(nLogn)](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/).

**Relation with the Monte-Carlo Algorithms:**

1. The Las-vegas algorithm can be differentiated with the Monte-carlo algorithms in which the resources used to find out the solution are bounded but it does not give guarantee  that the solution obtained is accurate.
2. In some applications by making early termination a Las-vegas algorithm can be convertcd into Monte-carlo algorithm.

**Complexity Analysis:**

  The complexity class of given problem which is solved by using a Las-vegas algorithms is expect that the given problem is solved with zero error probability and in polynomial time.

This zero error probability polynomial time is also called as ZPP which is obtained as follows,

ZPP = RP ∩ CO-RP

 Where, RP = Randomized polynomial time.

Randomized polynomial time algorithm always provide correct output when the comect output is no, but with a certain probability bounded away from one when the answer is yes. These kinds of decision problem can be included in class RP i.e. randomized where polynomial time.

That is how we can solve given problem in expected polynomial time by using Las-vegas algorithm. Generally there is no upper bound for Las-vegas algorithn related to worst case run time.

**Monte Carlo:**

The computational algorithms which rely on repeated random sampling to compute their resulls such algorithm are called as **Monte-carlo algorithms.**

**OR**

The random algorithm is Monte-carlo algorithms if it can give the wrong answer sometimes.

Whenever the existing deterministic algorithm is fail or it is impossible to compute the solution for given problem then Monte-carlo algorithms or methods are used. Monte-carlo methods are best repeated computation of the random numbers, and that’s why these algorithms are used for solving physical simulation system and mathematical system.

This Monte-carlo algorithms are specially useful for disordered materials, fluids, cellular structures. In case of mathematics these method are used to calculate the definite integrals, these integrals are provided with the complicated boundary conditions for multidimensional integrals. This method is successive one with consideration of risk analysis when compared to other methods.

There is no single Monte carlo methods other than the term describes a large and widely used class approaches and these approach use the following pattern.

1. Possible inputs of domain is defined.

2. By using a certain specified probability distribution generate the inputs randomly from the domain.

3. By using these inputs perform a deterministic computation.

4.The final result can be computed by aggregating the results of the individual computation.

Produce correct or optimum result with some probability. These algorithms have deterministic running time and it is generally easier to find out worst case time complexity. For example [this implementation of Karger’s Algorithm](https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/) produces minimum cut with probability greater than or equal to 1/n2 (n is number of vertices) and has worst case time complexity as O(E). Another example is [Fermat Method for Primality Testing](https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/). **Example to Understand Classification:**

Consider a binary array where exactly half elements are 0  
and half are 1. The task is to find index of any 1.

A Las Vegas algorithm for this task is to keep picking a random element until we find a 1. A Monte Carlo algorithm for the same is to keep picking a random element until we either find 1 or we have tried maximum allowed times say k. The Las Vegas algorithm always finds an index of 1, but time complexity is determined as expect value. The [expected number of trials before success](https://www.geeksforgeeks.org/expected-number-of-trials-before-success/) is 2, therefore expected time complexity is O(1). The Monte Carlo Algorithm finds a 1 with probability [1 – (1/2)k]. Time complexity of Monte Carlo is O(k) which is deterministic

**Optimization of Monte-Carlo Algorithms:**

1. In general the Monte-carlo optimization techniques are dependent on the random walks. The program for Monte carlo algorithms move in multidimensional space around the generated marker or handle. It wanted to move to the lower function but sometimes moves against the gradient.
2. In numerical optimization the numerical simulation is used which effective and efficientand popular application for the random numbers. The travelling salesman problem is an optimization problem which is one of the best examples of optimizations.
3. There are various optimization techniques available for Monte-carlo algorithms such as Evolution strategy, Genetic algorithms, parallel tempering etc.

**Applications and Scope:**

The Monte-carlo methods has wider range of applications. It uses in various areas like physical science, Design and visuals, Finance and business, Telecommunication etc. In general Monte carlo methods are used in mathematics. By generating random numbers we can solve the various problem. The problems which are complex in nature or difficult to solve are solved by using Monte-carlo algorithms. Monte carlo integration is the most common application of Monte-carlo algorithm.

The deterministic algorithm provides a correct solution but it takes long time or its runtime is large. This run-time can be improved by using the Monte carlo integration algorithms. There are various methods used for integration by using Monte-carlo methods such as,

i) Direct sampling methods which includes the stratified sampling, recursive

stratified sampling, importance sampling.

ii) Random walk Monte-carlo algorithm which is used to find out the integration for

given problem.

iii) Gibbs sampling.

1. Consider a tool that basically does sorting. Let the tool be used by many users and there are few users who always use tool for already sorted array. If the tool uses simple (not randomized) QuickSort, then those few users are always going to face worst case situation. On the other hand if the tool uses Randomized QuickSort, then there is no user that always gets worst case. Everybody gets expected O(n Log n) time.
2. Randomized algorithms have huge applications in Cryptography.
3. [Load Balancing](https://www.geeksforgeeks.org/load-balancing-on-servers-random-algorithm/).
4. Number-Theoretic Applications: [Primality Testing](https://en.wikipedia.org/wiki/Solovay%E2%80%93Strassen_primality_test)
5. Data Structures: Hashing, Sorting, Searching, [Order Statistics](https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-2-expected-linear-time/) and Computational Geometry.
6. Algebraic identities: Polynomial and [matrix identity verification](https://en.wikipedia.org/wiki/Randomized_algorithm#Verifying_matrix_multiplication). Interactive proof systems.
7. Mathematical programming: Faster algorithms for linear programming, Rounding linear program solutions to integer program solutions
8. Graph algorithms: Minimum spanning trees, shortest paths, [minimum cuts](https://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/).
9. Counting and enumeration: Matrix permanent Counting combinatorial structures.
10. Parallel and distributed computing: Deadlock avoidance distributed consensus.
11. Probabilistic existence proofs: Show that a combinatorial object arises with non-zero probability among objects drawn from a suitable probability space.
12. Derandomization: First devise a randomized algorithm then argue that it can be derandomized to yield a deterministic algorithm.

**Randomized algorithm**s are algorithms that use randomness as a key component in their operation. They can be used to solve a wide variety of problems, including optimization, search, and decision-making. Some examples of applications of randomized algorithms include:

1. Monte Carlo methods: These are a class of randomized algorithms that use random sampling to solve problems that may be deterministic in principle, but are too complex to solve exactly. Examples include estimating pi, simulating physical systems, and solving optimization problems.
2. Randomized search algorithms: These are algorithms that use randomness to search for solutions to problems. Examples include genetic algorithms and simulated annealing.
3. Randomized data structures: These are data structures that use randomness to improve their performance. Examples include skip lists and hash tables.
4. Randomized load balancing: These are algorithms used to distribute load across a network of computers, using randomness to avoid overloading any one computer.
5. Randomized encryption: These are algorithms used to encrypt and decrypt data, using randomness to make it difficult for an attacker to decrypt the data without the correct key.

**Example :**

**import** random

# Generates a random permutation of the given array

**def** random\_permutation(array):

    # Shuffle the array using the random number generator

    random.shuffle(array)

array **=** [1, 2, 3, 4, 5]

# Generate a random permutation of the array

random\_permutation(array)

# Print the shuffled array

print(array)

**Output**

5 1 4 2 3

**from** random **import** shuffle

**def** find\_median(numbers):

    n **=** len(numbers)

**if** n **==** 0:

**return** None

**if** n **==** 1:

**return** numbers[0]

    # Shuffle the list to ensure a random ordering

    shuffle(numbers)

    # Find the median by selecting the middle element

**return** numbers[n **//** 2]

# Example usage

print(find\_median([1, 2, 3, 4, 5]))  # Output: 3

**print**(find\_median([1, 2, 3, 4, 5, 6]))  # Output: 3 or 4 (randomly chosen)

**print**(find\_median([]))  # Output: None

**print**(find\_median([7]))  # Output: 7

**Output**

4  
6  
None  
7

**Sources:** <http://theory.stanford.edu/people/pragh/amstalk.pdf> <https://en.wikipedia.org/wiki/Randomized_algorithm> This article is contributed by **Ashish Sharma**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

**Randomized Algorithms | Set 3 (1/2 Approximate Median)**

We strongly recommend to refer below articles as a prerequisite of this.

[Randomized Algorithms | Set 1 (Introduction and Analysis)](https://www.geeksforgeeks.org/randomized-algorithms-set-1-introduction-and-analysis/)

[Randomized Algorithms | Set 2 (Classification and Applications)](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/)

In this post, a Monte Carlo algorithm is discussed.

**Problem Statement :** Given an unsorted array A[] of n numbers and ε > 0, compute an element whose rank (position in sorted A[]) is in the range [(1 – ε)n/2, (1 + ε)n/2].

For ½ Approximate Median Algorithm &epsilom; is 1/2 => rank should be in the range [n/4, 3n/4]

We can find k’th smallest element in [O(n) expected time](https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-2-expected-linear-time/) and [O(n) worst case](https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-3-worst-case-linear-time/) time.

**What if we want in less than O(n) time with low probable error allowed?**

Following steps represent an algorithm that is O((Log n) x (Log Log n)) time and produces incorrect result with probability less than or equal to 2/n2.

1. Randomly choose k elements from the array where k=c log n (c is some constant)
2. Insert then into a set.
3. Sort elements of the set.
4. Return median of the set i.e. (k/2)th element from the set

/\* C++ program to find Approximate Median using

   1/2 Approximate Algorithm \*/

#include<bits/stdc++.h>

**using namespace** std;

// This function returns the Approximate Median

**int** randApproxMedian(**int** arr[],**int** n)

{

    // Declaration for the random number generator

    random\_device rand\_dev;

    mt19937 generator(rand\_dev());

    // Random number generated will be in the range [0,n-1]

    uniform\_int\_distribution<**int**> distribution(0, n-1);

**if** (n==0)

**return** 0;

**int** k = 10\*log2(n); // Taking c as 10

    // A set stores unique elements in sorted order

    set<**int**> s;

**for** (**int** i=0; i<k; i++)

    {

        // Generating a random index

**int** index = distribution(generator);

        //Inserting into the set

        s.insert(arr[index]);

    }

    set<**int**> ::iterator itr = s.begin();

    // Report the median of the set at k/2 position

    // Move the itr to k/2th position

    advance(itr, (s.size()/2) - 1);

    // Return the median

**return** \*itr;

}

// Driver method to test above method

**int** main()

{

**int** arr[] = {1, 3, 2, 4, 5, 6, 8, 7};

**int** n = **sizeof**(arr)/**sizeof**(**int**);

**printf**("Approximate Median is %d\n",randApproxMedian(arr,n));

**return** 0

}

**Time Complexity:**

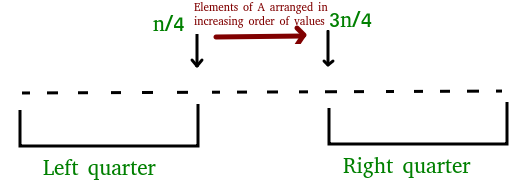
We use a set provided by the STL in C++. In STL Set, insertion for each element takes O(log k). So for k insertions, time taken is O (k log k).

Now replacing k with c log n

=>O(c log n (log (clog n))) =>O (log n (log log n))

**How is probability of error less than 2/n2?**

Algorithm makes an error if the set S has at least k/2 elements are from the Left Quarter or Right Quarter.

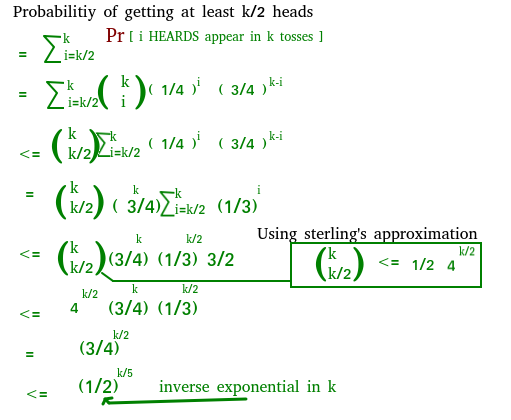


It is quite easy to visualize this statement since the median which we report will be (k/2)th element and if we take k/2 elements from the left quarter(or right quarter) the median will be from the left quarter (or the right quarter).

An array can be divided into 4 quarters each of size n/4. So P(selecting left quarter) is 1/4. So what is the probability that at least k/2 elements are from the Left Quarter or Right Quarter? This probability problem is same as below :

Given a coin which gives HEADS with probability 1/4 and TAILS with 3/4. The coin is tossed k times. What is the probability that we get at least k/2 HEADS is less than or equal to?

**Explanation:**



If we put k = c log n for c = 10, we get   
P <= (1/2)2log n  
P <= (1/2)log n2  
P <= n-2

Probability of selecting at least k/2 elements from the left quarter) <= 1/n2

Probability of selecting at least k/2 elements from the left or right quarter) <= 2/n2

Therefore algorithm produces incorrect result with probability less that or equal to 2/n2.

**References:**[www.cse.iitk.ac.in/users/sbaswana/CS648/Lecture-2-CS648.pptx](http://www.cse.iitk.ac.in/users/sbaswana/CS648/Lecture-2-CS648.pptx)

This article is contributed by **Chirag Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

**Easy Questions:**

**Write a function that generates one of 3 numbers according to given probabilities**

You are given a function rand(a, b) which generates equiprobable random numbers between [a, b] inclusive. Generate 3 numbers x, y, z with probability P(x), P(y), P(z) such that P(x) + P(y) + P(z) = 1 using the given rand(a,b) function.

The idea is to utilize the equiprobable feature of the rand(a,b) provided. ***Let the given probabilities be in percentage form, for example P(x)=40%, P(y)=25%, P(z)=35%.***.

Following are the detailed steps.

**1)** Generate a random number between 1 and 100. Since they are equiprobable, the probability of each number appearing is 1/100.

**2)** Following are some important points to note about generated random number ‘r’.

a) ‘r’ is smaller than or equal to P(x) with probability P(x)/100.

b) ‘r’ is greater than P(x) and smaller than or equal P(x) + P(y) with P(y)/100.

c) ‘r’ is greater than P(x) + P(y) and smaller than or equal 100 (or P(x) + P(y) + P(z)) with probability P(z)/100.

**import** random

# This function generates 'x' with probability px/100, 'y' with

# probability py/100  and 'z' with probability pz/100:

# Assumption: px + py + pz = 100 where px, py and pz lie

# between 0 to 100

**def** random(x, y, z, px, py, pz):

    # Generate a number from 1 to 100

    r **=** random.randint(1, 100)

    #  r is smaller than px with probability px/100

**if** (r <**=** px):

**return** x

    # r is greater than px and smaller than

    # or equal to px+py with probability py/100

**if** (r <**=** (px**+**py)):

**return** y

    # r is greater than px+py and smaller than

    # or equal to 100 with probability pz/100

**else**:

**return** z

# This code is contributed by rohan07

Time Complexity: O(1)

Auxiliary Space: O(1)

*From <*[*https://www.geeksforgeeks.org/write-a-function-to-generate-3-numbers-according-to-given-probabilities/*](https://www.geeksforgeeks.org/write-a-function-to-generate-3-numbers-according-to-given-probabilities/)*>*

**Generate 0 and 1 with 25% and 75% probability**

Given a function rand50() that returns 0 or 1 with equal probability, write a function that returns 1 with 75% probability and 0 with 25% probability using rand50() only. Minimize the number of calls to the rand50() method. Also, the use of any other library function and floating-point arithmetic are not allowed.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is to use **Bitwise OR**. A bitwise OR takes two bits and returns 0 if both bits are 0, while otherwise, the result is 1. So it has 75% probability that it will return 1.

Below is the implementation of the above idea :

# Program to print 1 with 75% probability and 0

# with 25% probability

**from** random **import** randrange

# Random Function to that returns 0 or 1 with

# equal probability

**def** rand50():

        # The randrange function will generate integer

    # between the half closed interval at end

    # Here by passing parameter as 0,2

    # the function will generate integer between 0 and 1

**return** (int)(randrange(0, 2)) & 1

# Random Function to that returns 1 with 75%

# probability and 0 with 25% probability using

# Bitwise OR

**def** rand75():

**return** rand50() | rand50()

# Driver code to test above functions

**for** i **in** range(0, 50):

    print(rand75(), end**=**"")

    # This code is contributed by meetgor.

**Output**

11101010110101011010000101011110100010111110101111

**Time Complexity:**O(1)

**Auxiliary Space:**O(1)

On similar lines, we can also use **Bitwise AND**. Since it returns 0 with 75% probability, we have to invert the result.

// Random Function to that returns 1 with 75%   
// probability and 0 with 25% probability using  
// Bitwise AND  
bool rand75()   
{  
 return !(rand50() & rand50());  
}

Below is the implementation of the above idea :

**from** random **import** randrange

# Random Function to that returns 0 or 1 with

# equal probability

**def** rand50():

**return** ((int)(randrange(0, 2)) & 1)

**def** rand75():

**return** (rand50() & rand50())^1

**for** i **in** range(0, 50):

    print(rand75(), end**=**"")

**Output**

11111111000111101111110011111110011110111111010111

We can replace Bitwise OR and Bitwise AND operators with **OR and AND operators** as well –

// Random Function to that returns 1 with 75%  
// probability and 0 with 25% probability using   
// OR or AND operator  
int rand75()  
{  
 return !(rand50() && rand50());  
 // return rand50() || rand50()  
}

We can also achieve the result using the **left shift operator and Bitwise XOR** –

# Program to print 1 with 75% probability and 0

# with 25% probability

**from** random **import** randrange

# Random Function to that returns 0 or 1 with

# equal probability

**def** rand50():

        # rand range function generates a integer between

    # the provided ranges which is half closed interval

    # It will generate integer 0 or 1 if passed 0,2 as parameter

**return** (int)(randrange(0, 2)) & 1

# Random Function to that returns 1 with 75%

# probability and 0 with 25% probability using

# left shift and Bitwise XOR

**def** rand75():

    # x is one of {0, 1}

    x **=** rand50()

    x **=** x << 1

    # x is now one of {00, 10}

    x **=** x ^ rand50()

    # x is now one of {00, 01, 10, 11}

**return** 1 **if** (x > 0) **else** 0

# Driver code to test above functions

**for** i **in** range(0, 50):

    print(rand75(), end**=**"")

    # This code is contributed by meetgor.

**Output**

10110100111011011110111100101111110111100001111111

**Time Complexity:** O(1)

**Auxiliary Space:**O(1)

*From <*[*https://www.geeksforgeeks.org/generate-0-1-25-75-probability/*](https://www.geeksforgeeks.org/generate-0-1-25-75-probability/)*>*

**Implement rand3() using rand2()**

Given a function rand2() that returns 0 or 1 with equal probability, implement rand3() using rand2() that returns 0, 1 or 2 with equal probability. Minimize the number of calls to rand2() method. Also, use of any other library function and floating point arithmetic are not allowed.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is to use expression **2 \* rand2() + rand2()**. It returns 0, 1, 2, 3 with equal probability. To make it return 0, 1, 2 with equal probability, we eliminate the undesired event 3.

Below is the implementation of above idea –

# Python3 Program to print 0, 1 or 2 with equal

# Probability

**import** random

# Random Function to that returns 0 or 1 with

# equal probability

**def** rand2():

    # randint(0,100) function will generate odd or even

    # number [1,100] with equal probability. If rand()

    # generates odd number, the function will

    # return 1 else it will return 0

    tmp**=**random.randint(1,100)

**return** tmp**%**2

# Random Function to that returns 0, 1 or 2 with

# equal probability 1 with 75%

**def** rand3():

    # returns 0, 1, 2 or 3 with 25% probability

    r **=** 2 **\*** rand2() **+** rand2()

**if** r<3:

**return** r

**return** rand3()

# Driver code to test above functions

**if** \_\_name\_\_**==**'\_\_main\_\_':

**for** i **in** range(100):

        print(rand3(),end**=**"")

#This code is contributed by sahilshelangia

**Output :**

2111011101112002111002020210112022022022211100100121202021102100010200121121210122011022111020

**Another Solution –**

If x = rand2() and y = rand2(), x + y will return 0 and 2 with 25% probability and 1 with 50% probability. To make probability of 1 equal to that of 0 and 2 i.e. 25%, we eliminate one undesired event that’s resulting in x + y = 1 i.e. either (x = 1, y = 0) or (x = 0, y = 1).

int rand3()  
{  
 int x, y;

do {  
 x = rand2();  
 y = rand2();  
 } while (x == 0 && y == 1);

return x + y;  
}

*From <*[*https://www.geeksforgeeks.org/implement-rand3-using-rand2/*](https://www.geeksforgeeks.org/implement-rand3-using-rand2/)*>*

**Birthday Paradox**

*How many people must be there in a room to make the probability 100% that at-least two people in the room have same birthday?*

Answer: 367 (since there are 366 possible birthdays, including February 29).

The above question was simple. Try the below question yourself.

***How many people must be there in a room to make the probability 50% that at-least two people in the room have same birthday?***

Answer: 23

The number is surprisingly very low. In fact, we need only 70 people to make the probability 99.9 %.

Let us discuss the generalized formula.

**What is the probability that two persons among n have same birthday?**

Let the probability that two people in a room with n have same birthday be P(same). P(Same) can be easily evaluated in terms of P(different) where P(different) is the probability that all of them have different birthday.

P(same) = 1 – P(different)

P(different) can be written as 1 x (364/365) x (363/365) x (362/365) x …. x (1 – (n-1)/365)

*How did we get the above expression?*

Persons from first to last can get birthdays in following order for all birthdays to be distinct:

The first person can have any birthday among 365

The second person should have a birthday which is not same as first person

The third person should have a birthday which is not same as first two persons.

…………….

……………

The n’th person should have a birthday which is not same as any of the earlier considered (n-1) persons.

**Approximation of above expression**

The above expression can be approximated using Taylor’s Series.

provides a first-order approximation for ex for x << 1:

To apply this approximation to the first expression derived for p(different), set x = -a / 365. Thus,

The above expression derived for p(different) can be written as

1 x (1 – 1/365) x (1 – 2/365) x (1 – 3/365) x …. x (1 – (n-1)/365)

By putting the value of 1 – a/365 as e-a/365, we get following.

Therefore,

p(same) = 1- p(different)

An even coarser approximation is given by

p(same)

By taking Log on both sides, we get the reverse formula.

Using the above approximate formula, we can approximate number of people for a given probability. For example the following C++ function find() returns the smallest n for which the probability is greater than the given p.

**Implementation of approximate formula.**

The following is program to approximate number of people for a given probability.

# Python3 code to approximate number

# of people in Birthday Paradox problem

**import** math

# Returns approximate number of

# people for a given probability

**def** find( p ):

**return** math.ceil(math.sqrt(2 **\*** 365 **\***

                     math.log(1**/**(1**-**p))));

# Driver Code

print(find(0.70))

# This code is contributed by "Sharad\_Bhardwaj".

**Output :**

30

Time Complexity: O(log n)

Auxiliary Space: O(1)

**Source:**

<http://en.wikipedia.org/wiki/Birthday_problem>

**Applications:**

1) Birthday Paradox is generally discussed with hashing to show importance of collision handling even for a small set of keys.

2) [Birthday Attack](http://en.wikipedia.org/wiki/Birthday_attack)

Below is an alternate implementation in C language :

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Assuming non-leap year

    num **=** 365;

    denom **=** 365;

    pr **=** 0.7;

    n **=** 0;

    p **=** 1;

**while** (p > pr):

        p **\*=** (num **/** denom);

        num **-=** 1;

        n **+=** 1;

**print**("Total no. of people out of which there is ", end**=**"");

    print ("{0:.1f}".format(p), end**=**"")

    print(" probability that two of them " **+** "have same birthdays is ", n);

# This code is contributed by Rajput-Ji

Time Complexity: O(log n)

Auxiliary Space: O(1)

*From <*[*https://www.geeksforgeeks.org/birthday-paradox/*](https://www.geeksforgeeks.org/birthday-paradox/)*>*

**Shuffle a deck of cards**

Given a deck of cards, the task is to shuffle them. Asked in [Amazon Interview](https://www.geeksforgeeks.org/amazon-interview-experience-set-145-campus/)

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Prerequisite : [Shuffle a given array](https://www.geeksforgeeks.org/shuffle-a-given-array/) **Algorithm:**

1. First, fill the array with the values in order.  
2. Go through the array and exchange each element   
 with the randomly chosen element in the range   
 from itself to the end.

// It is possible that an element will be swap  
// with itself, but there is no problem with that.

# Python3 program for shuffling desk of cards

# Function which shuffle and print the array

**import** random

**def** shuffle(card,n) :

    # Initialize seed randomly

**for** i **in** range(n):

        # Random for remaining positions.

        r **=** i **+** (random.randint(0,55) **%** (52 **-**i))

        tmp**=**card[i]

        card[i]**=**card[r]

        card[r]**=**tmp

#Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    a**=**[0, 1, 2, 3, 4, 5, 6, 7, 8,

       9, 10, 11, 12, 13, 14, 15,

       16, 17, 18, 19, 20, 21, 22,

       23, 24, 25, 26, 27, 28, 29,

       30, 31, 32, 33, 34, 35, 36,

       37, 38, 39, 40, 41, 42, 43,

       44, 45, 46, 47, 48, 49, 50,

       51]

    shuffle(a,52)

    print(a)

#this code is contributed by sahilshelangia

Output:

29 27 20 23 26 21 35 51 15 18 46 32 33 19   
24 30 3 45 40 34 16 11 36 50 17 10 7 5 4   
39 6 47 38 28 13 44 49 1 8 42 43 48 0 12   
37 41 25 2 31 14 22

***Time Complexity:*** O(n)

***Space Complexity:*** O(1)

*From <*[*https://www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/*](https://www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/)*>*

**Program to generate CAPTCHA and verify user**

A [CAPTCHA](https://en.wikipedia.org/wiki/CAPTCHA) (Completely Automated Public Turing test to tell Computers and Humans Apart) is a test to determine whether the user is human or not.

So, the task is to generate unique CAPTCHA every time and to tell whether the user is human or not by asking user to enter the same CAPTCHA as generated automatically and checking the user input with the generated CAPTCHA.

**Examples:**

CAPTCHA: x9Pm72se  
Input: x9Pm62es  
Output: CAPTCHA Not Matched

CAPTCHA: cF3yl9T4  
Input: cF3yl9T4  
Output: CAPTCHA Matched

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The set of characters to generate CAPTCHA are stored in a character array chrs[] which contains (a-z, A-Z, 0-9), therefore size of chrs[] is 62.

To generate a unique CAPTCHA every time, a random number is generated using rand() function (rand()%62) which generates a random number between 0 to 61 and the generated random number is taken as index to the character array chrs[] thus generates a new character of captcha[] and this loop runs n (length of CAPTCHA) times to generate CAPTCHA of given length.

# Python program to automatically generate CAPTCHA and

# verify user

**import** random

# Returns true if given two strings are same

**def** checkCaptcha(captcha, user\_captcha):

**if** captcha **==** user\_captcha:

**return** True

**return** False

# Generates a CAPTCHA of given length

**def** generateCaptcha(n):

    # Characters to be included

    chrs **=** "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

    # Generate n characters from above set and

    # add these characters to captcha.

    captcha **=** ""

**while** (n):

        captcha **+=** chrs[random.randint(1, 1000) **%** 62]

        n **-=** 1

**return** captcha

# Driver code

# Generate a random CAPTCHA

captcha **=** generateCaptcha(9)

print(captcha)

# Ask user to enter a CAPTCHA

print("Enter above CAPTCHA:")

usr\_captcha **=** input()

# Notify user about matching status

**if** (checkCaptcha(captcha, usr\_captcha)):

**print**("CAPTCHA Matched")

**else**:

    print("CAPTCHA Not Matched")

# This code is contributed by shubhamsingh10

Output:

CAPTCHA: cF3yl9T4  
Enter CAPTCHA: cF3yl9T4  
CAPTCHA Matched

**Time Complexity:** O(n)

**Space Complexity:** O(1)

*From <*[*https://www.geeksforgeeks.org/program-generate-captcha-verify-user/*](https://www.geeksforgeeks.org/program-generate-captcha-verify-user/)*>*

**Expectation or expected value of an array**

[Expectation or expected value](https://www.geeksforgeeks.org/linearity-of-expectation/)of any group of numbers in probability is the long-run average value of repetitions of the experiment it represents. For example, the expected value in rolling a six-sided die is 3.5, because the average of all the numbers that come up in an extremely large number of rolls is close to 3.5. Less roughly, the law of large numbers states that the arithmetic mean of the values almost surely converges to the expected value as the number of repetitions approaches infinity. The expected value is also known as the **expectation**, **mathematical expectation**, EV, or first moment.

Given an array, the task is to calculate the **expected value** of the array.

**Examples :**

***Input:****[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]*

***Output:****3.5*

***Input:****[1.0, 9.0, 6.0, 7.0, 8.0, 12.0]*

***Output:****7.16*

Recommended Practice

[Please try your approach on IDE first, before](https://ide.geeksforgeeks.org/)

[moving on to the solution.](https://ide.geeksforgeeks.org/)

[Try It!](https://ide.geeksforgeeks.org/)

Below is the implementation :

# python code to calculate expected   
# value of an array

# Function to calculate expectation  
def calc\_Expectation(a, n):  
   
 # variable prb is for probability   
 # of each element which is same for  
 # each element   
 prb = 1 / n  
   
 # calculating expectation overall  
 sum = 0  
 for i in range(0, n):  
 sum += (a[i] \* prb)   
   
 # returning expectation as sum  
 return float(sum)

# Driver program  
n = 6;  
a = [ 1.0, 2.0, 3.0,4.0, 5.0, 6.0 ]

# Function for calculating expectation  
expect = calc\_Expectation(a, n)

# Display expectation of given array  
print( "Expectation of array E(X) is : ",  
 expect )

# This code is contributed by Sam007

**Output**

Expectation of array E(X) is : 3.5

**Time complexity:** O(n)

**Auxiliary Space:** O(1)

**Find an index of maximum occurring element with equal probability**

Given an array of integers, find the most occurring element of the array and return any one of its indexes randomly with equal probability.

Examples:

**Input:**   
arr[] = [-1, 4, 9, 7, 7, 2, 7, 3, 0, 9, 6, 5, 7, 8, 9]

**Output:**    
Element with maximum frequency present at index 6  
OR  
Element with maximum frequency present at Index 3  
OR  
Element with maximum frequency present at index 4  
OR  
Element with maximum frequency present at index 12

All outputs above have equal probability.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is to iterate through the array once and find out the maximum occurring element and its frequency n. Then we generate a random number r between 1 and n and finally return the r’th occurrence of maximum occurring element in the array.

Below are implementation of above idea –

# Python3 program to return index of most occurring element

# of the array randomly with equal probability

**import** random

# Function to return index of most occurring element

# of the array randomly with equal probability

**def** findRandomIndexOfMax(arr, n):

    # freq store frequency of each element in the array

    mp **=** dict()

**for** i **in** range(n) :

**if**(arr[i] **in** mp):

            mp[arr[i]] **=** mp[arr[i]] **+** 1

**else**:

            mp[arr[i]] **=** 1

    max\_element **= -**323567

    # stores max occurring element

    # stores count of max occurring element

    max\_so\_far **= -**323567

    # traverse each pair in map and find maximum

    # occurring element and its frequency

**for** p **in** mp :

**if** (mp[p] > max\_so\_far):

            max\_so\_far **=** mp[p]

            max\_element **=** p

    # generate a random number between [1, max\_so\_far]

    r **=** int( ((random.randrange(1, max\_so\_far, 2) **%** max\_so\_far) **+** 1))

    i **=** 0

    count **=** 0

    # traverse array again and return index of rth

    # occurrence of max element

**while** ( i < n ):

**if** (arr[i] **==** max\_element):

            count **=** count **+** 1

        # Print index of rth occurrence of max element

**if** (count **==** r):

**print**("Element with maximum frequency present at index " , i )

**break**

        i **=** i **+** 1

# Driver code

# input array

arr **=** [**-**1, 4, 9, 7, 7, 2, 7, 3, 0, 9, 6, 5, 7, 8, 9]

n **=** len(arr)

findRandomIndexOfMax(arr, n)

# This code is contributed by Arnab Kundu

**Output:**

Element with maximum frequency present at index 4

**Time complexity** of above solution is O(n).

**Auxiliary space** used by the program is O(n).

*From <*[*https://www.geeksforgeeks.org/find-index-maximum-occurring-element-equal-probability/*](https://www.geeksforgeeks.org/find-index-maximum-occurring-element-equal-probability/)*>*

**Randomized Binary Search Algorithm**

We are given a sorted array A[] of n elements. We need to find if x is present in A or not.In binary search we always used middle element, here we will randomly pick one element in given range.

In [Binary Search](https://www.geeksforgeeks.org/binary-search/) we had

middle = (start + end)/2

In Randomized binary search we do following

Generate a random number t  
Since range of number in which we want a random  
number is [start, end]  
Hence we do, t = t % (end-start+1)  
Then, t = start + t;  
Hence t is a random number between start and end

It is a [Las Vegas randomized algorithm](https://www.geeksforgeeks.org/randomized-algorithms-set-2-classification-and-applications/) as it always finds the correct result.

**Expected Time complexity of Randomized Binary Search Algorithm**

For n elements let say expected time required be T(n), After we choose one random pivot, array size reduces to say k. Since pivot is chosen with equal probability for all possible pivots, hence p = 1/n.

T(n) is sum of time of all possible sizes after choosing pivot multiplied by probability of choosing that pivot plus time take to generate random pivot index.Hence

T(n) = p\*T(1) + p\*T(2) + ..... + p\*T(n) + 1  
putting p = 1/n  
T(n) = ( T(1) + T(2) + ..... + T(n) ) / n + 1  
n\*T(n) = T(1) + T(2) + .... + T(n) + n .... eq(1)  
Similarly for n-1  
(n-1)\*T(n-1) = T(1) + T(2) + ..... + T(n-1) + n-1 .... eq(2)  
Subtract eq(1) - eq(2)  
n\*T(n) - (n-1)\*T(n-1) = T(n) + 1  
(n-1)\*T(n) - (n-1)\*T(n-1) = 1  
(n-1)\*T(n) = (n-1)\*T(n-1) + 1  
T(n) = 1/(n-1) + T(n-1)  
T(n) = 1/(n-1) + 1/(n-2) + T(n-2)  
T(n) = 1/(n-1) + 1/(n-2) + 1/(n-3) + T(n-3)  
Similarly,  
T(n) = 1 + 1/2 + 1/3 + ... + 1/(n-1)  
Hence T(n) is equal to (n-1)th Harmonic number,   
n-th harmonic number is O(log n)  
Hence T(n) is O(log n)

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/binary-search/1)

**Recursive implementation of Randomized Binary Search**

# Python3 program to implement recursive

# randomized algorithm.

# To generate random number

# between x and y ie.. [x, y]

**import** random

**def** getRandom(x,y):

    tmp**=**(x **+** random.randint(0,100000) **%** (y**-**x**+**1))

**return** tmp

# A recursive randomized binary search function.

# It returns location of x in

# given array arr[l..r] is present, otherwise -1

**def** randomizedBinarySearch(arr,l,r,x) :

**if** r>**=**l:

        # Here we have defined middle as

        # random index between l and r ie.. [l, r]

        mid**=**getRandom(l,r)

        # If the element is present at the

        # middle itself

**if** arr[mid] **==** x:

**return** mid

        # If element is smaller than mid, then

        # it can only be present in left subarray

**if** arr[mid]>x:

**return** randomizedBinarySearch(arr, l, mid**-**1, x)

        # Else the element can only be present

        # in right subarray

**return** randomizedBinarySearch(arr, mid**+**1,r, x)

    # We reach here when element is not present

    # in array

**return -**1

# Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    arr **=** [2, 3, 4, 10, 40]

    n**=**len(arr)

    x**=**10

    result **=** randomizedBinarySearch(arr, 0, n**-**1, x)

**if** result**==-**1:

        print('Element is not present in array')

**else**:

        print('Element is present at index ', result)

# This code is contributes by sahilshelangia

**Output:**

Element is present at index 3

**Iterative implementation of Randomized Binary Search**

# Python program to implement iterative

# randomized algorithm.

# To generate random number

# between x and y ie.. [x, y]

**from** random **import** randint

**def** getRandom(x, y):

**return** randint(x,y)

# A iterative randomized binary search function.

# It returns location of x in

# given array arr[l..r] if present, otherwise -1

**def** randomizedBinarySearch(arr, l, r, x):

**while** (l <**=** r):

        # Here we have defined middle as

        # random index between l and r ie.. [l, r]

        m **=** getRandom(l, r)

        # Check if x is present at mid

**if** (arr[m] **==** x):

**return** m

        # If x greater, ignore left half

**if** (arr[m] < x):

            l **=** m **+** 1

        # If x is smaller, ignore right half

**else**:

            r **=** m **-** 1

    # if we reach here, then element was

    # not present

**return -**1

# Driver code

arr **=** [2, 3, 4, 10, 40]

n **=** len(arr)

x **=** 10

result **=** randomizedBinarySearch(arr, 0, n**-**1, x)

**if** result **==** 1:

    print("Element is not present in array")

**else**:

**print**("Element is present at index", result)

# This code is contributed by ankush\_953

**Output:**

Element is present at index 3