EE4371 - Assignment 1

Om Shri Prasath, EE17B113

Q1. (a) Write a Python function that takes a positive integer n, and returns the sum of the squares of all the positive integers smaller than n.

Solution 1 : $O(n)$ Time Complexity

We run a loop to sum the squares of all the positive integers less than n. This algorithm takes $O(n)$ time since we do $n-1$ operations for a given integer n.

In [1]:

```python
"""
Function to find the square sum of positive integers less than n using for loop

Input  : n -> A positive integer n
Output : Sum of squares of all the positive integers less than n
"""


def sq_sum_1(n):

    s = 0  # Variable to hold the sum of squares

    # For loop runs over all positive integers less than n
    for i in range(1, n):
        # For each number we square it and add it to the final answer
        s += i * i

    # Return the sum of squares
    return s


# Testing the code
n = 6
print("Sum of square of numbers less than", n, "=", sq_sum_1(n))
```

```
Sum of square of numbers less than 6 = 55
```

Solution 2 : $O(1)$ Time Complexity

The formula for sum of square of positive integers less than n is given by:

$$S_{n^2} = \frac{(n-1)(n)(2n-1)}{6}$$

We can use this formula to calculate the sum of squares of integers less than n in $O(1)$ time.

In [2]:

```python
"""
Function to find the square sum of positive integers less than n using formula

Input  : n -> A positive integer n
Output : Sum of squares of all the positive integers less than n
"""


def sq_sum_2(n):

    # Using the above formula to calculate the sum
    s = ((n - 1) * (n) * (2 * n - 1)) // 6

    # Return the sum of squares
    return s


# Testing the code
n = 6
print("Sum of square of numbers less than", n, "=", sq_sum_2(n))
```

```
Sum of square of numbers less than 6 = 55
```

Q1. (b) Write a Python function that takes a positive integer n, and returns the sum of the squares of all the *odd* positive integers smaller than n.

Solution 1 : $O(n)$ Time Complexity

We run a loop to sum the squares of all the *odd* integers less than n. This algorithm takes $O(n)$ time since we do at least $n/2$ operations for a given integer n.

In [3]:

```python
"""
Function to find the square sum of positive odd integers less than n using for loop

Input  : n -> A positive integer n
Output : Sum of squares of all the odd positive integers less than n
"""


def sq_odd_sum_1(n):

    s = 0  # Variable to hold the sum of squares

    # For loop runs over all odd positive numbers less than n
    for i in range(1, n, 2):
        # For each number we square it and add it to the final answer
        s += i * i

    # Return the sum of squares
    return s


# Testing the code
n = 6
print("Sum of square of odd numbers less than", n, "=", sq_odd_sum_1(n))
```

```
Sum of square of odd numbers less than 6 = 35
```

Solution 2 : $O(1)$ Time Complexity

The formula for sum of square of *odd* integers less than n is given by:

$$S_{n^2_{odd}} = \frac{(k)(2k+1)(2k-1)}{3}; k = \left\lfloor \frac{n}{2} \right\rfloor$$

We can use this formula to calculate the sum of squares of *odd* integers less than n in $O(1)$ time.

In [4]:

```python
"""
Function to find the square sum of positive odd integers less than n using formula

Input  : n -> A positive integer n
Output : Sum of squares of all the odd positive integers less than n
"""


def sq_odd_sum_2(n):

    k = n // 2  # Calculating the value of k to use in formula

    # Using the above formula to calculate the sum
    s = ((k) * (2 * k + 1) * (2 * k - 1)) // 3

    # Return the sum of squares
    return s


# Testing the code
n = 6
print("Sum of square of odd numbers less than", n, "=", sq_odd_sum_2(n))
```

```
Sum of square of odd numbers less than 6 = 35
```

Q2. What parameter values should be sent to the `range` constructor to produce a range with values:

(a) 60,70,80

Solution : The parameter values are as follows : `range(60,81,10)`

In [5]:

```python
# Testing the parameter range
for i in range(60, 81, 10):
    print(i, end=" ")
```

```
60 70 80
```

(b) 4,2,0,-2,-4

Solution : The parameter values are as follows : `range(4,-5,-2)`

In [6]:

```python
# Testing the parameter range
for i in range(4, -5, -2):
    print(i, end=" ")
```

```
4 2 0 -2 -4
```

Q3. Write a Python function that takes a sequence of integer values and determines if there is a distinct pair of numbers in the sequence whose product is *odd*.

Solution : $O(n)$ Time Complexity

The basic idea is only the product of two odd numbers can result in an odd number. Thus we iterate over the sequence and find if there are atleast two distinct pair of numbers which are odd. The algorithm is $O(n)$, where n is the number of elements in the sequence, since in the worst case we need to iterate over all the elements in the sequence.

In [7]:

```python
"""
Function to determine if there is a distinct pair of numbers in sequence
whose product is odd

Input  : l -> Sequence of numbers
Output : Determines if there is a distinct pair of numbers in sequence
whose product is odd
"""


def distinct_odd_prod(l):

    c = 0  # Counter to keep count of number of odd numbers

    # We iterate over the sequence
    for i in l:

        # We increase the counter if the number is odd
        if i % 2 != 0:
            c += 1

        # Break out of loop if two odd numbers are encountered
        if c >= 2:
            break

    # Determine whether the condition is satisfied
    if c >= 2:
        print(
            "Sequence",
            l,
            "contains distinct odd product numbers",
        )
    else:
        print(
            "Sequence",
            l,
            " does not contain distinct odd product numbers",
        )


# Testing the code
l = [2, 3, 4, 5]
distinct_odd_prod(l)
l = [2, 3, 4]
distinct_odd_prod(l)
```

```
Sequence [2, 3, 4, 5] contains distinct odd product numbers
Sequence [2, 3, 4]  does not contain distinct odd product numbers
```

Q4. Write a Python function that counts the number of vowels in a given character string.

Solution : $O(n)$ Time Complexity

We iterate over the string and check if the character is vowel or not and update a counter accordingly. The algorithm is $O(n)$, where n is the length of string since we need to go over all the characters in the string once to check if it is a vowel or consonant.

In [8]:

```python
"""
Function to count the number of vowels in given string

Input  : st -> Character string
Output : Number of vowels
"""


def vowels_count(st):

    c = 0  # Counter to keep count of number of vowels

    vow = set(["a", "e", "i", "o", "u"])  # Set of vowels

    # We iterate over the characters of string
    for ch in st:

        # We increase the counter if the character is vowel
        if ch.lower() in vow:
            c += 1

    print("Number of vowels in", st, "=", c)


# Testing the code
st = "aeiou"
vowels_count(st)
st = "urals"
vowels_count(st)
```

```
Number of vowels in aeiou = 5
Number of vowels in urals = 2
```

Q5. Write a Python program that takes as input three integers, “a”, “b” and “c”, from the console and determines if they can be used in the following arithmetic formulas: (i) “a+b=c”, (ii) “a=b-c”, (iii) “a*b=c”.

Solution : $O(1)$ Time Complexity

We take the input from the user and check if the inputs satisfy the equations. This can be checked in $O(1)$ time since no iteration is involved.

In [9]:

```python
"""
Python program to check if given input integers
can be used in the above formulas
"""

# Getting input from the user
a = int(input("Enter a = "))
b = int(input("Enter b = "))
c = int(input("Enter c = "))

# Checking if a+b=c
e = a + b
f1 = e == c  # Boolean variable which stores if a+b=c is True or False

# Checking if a=b-c
f = b - c
f2 = f == a  # Boolean variable which stores if a=b-c is True or False

# Checking if a*b=c
g = a * b
f3 = g == c  # Boolean variable which stores if a*b=c is True or False

# Checking which all equations are possible
print()  # Dummy print for spacing
if f1:
    print("a+b=c is True")
if f2:
    print("a=b-c is True")
if f3:
    print("a*b=c is True")
if not (f1 or f2 or f3):
    print("None of the equations are True")
```

```
Enter a = 0
Enter b = 0
Enter c = 0

a+b=c is True
a=b-c is True
a*b=c is True
```

Q6. Write a Python function that takes a sequence of numbers and determines if all the numbers are different from each other (that is, they are distinct).

Solution : $O(n)$ Time Complexity

We use the idea that if all the elements of a sequence are distinct, then the length of the given sequence and the length of a new sequence containing all distinct elements should be the same. This can be achieved in Python by using the `set` function.

- We create a set from the sequence and compare its length with the original sequence.
- If they are the same, then all the elements in the list are distinct and vice-versa.

The creation of the set from the list is of $O(n)$ time complexity where n is the length of the array and since accessing the lengths is $O(1)$, the overall time complexity is $O(n)$.

In [10]:

```python
"""
Function to check if all numbers in sequence are distinct

Input  : l -> Sequence of numbers
Output : Determines if the list contains unique elements or not
"""


def distinct_sequence(l):

    l_set = set(l)  # Creating the set from the list

    # Comparing the lengths
    if len(l) == len(l_set):
        print(l, " - All numbers in the sequence are distinct")
    else:
        print(l, " - The sequence contains duplicate numbers")


# Testing the code
l = [1, 2, 3, 4, 5]
distinct_sequence(l)
l = [1, 1, 2]
distinct_sequence(l)
```

```
[1, 2, 3, 4, 5]  - All numbers in the sequence are distinct
[1, 1, 2]  - The sequence contains duplicate numbers
```

Q7. Design a program that can test the Birthday problem, by a series of experiments, on randomly generated birthdays which test this paradox for n = 5,10,15,20,25,30...200.

Solution : $O(n^2)$ Time Complexity

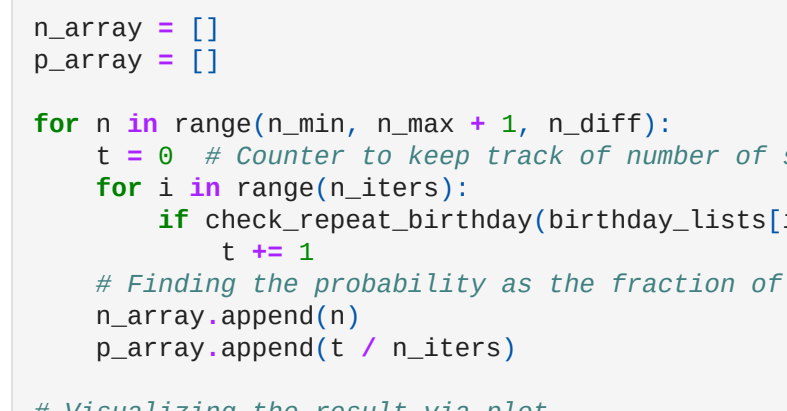
We run multiple experiments for a given n value as follows:

- Generate n birthdays which are numbers between 0-364
- We check if there are any repeating values in the given array which is similar to what we did in the previous question
- We repeat the above process 10,000 times and find the fraction of times where there was a repeating birthday. This fraction gives us the approximate probability

The given solution is $O(n^2)$ time complexity since there are n number of $O(n)$ operations to be done which effectively makes the solution $O(n^2)$. (Here we ignore the 10,000 iterations since its a constant in this case)

In [11]:

```python
"""
Python program to test the birthday problem
"""

import random  # For generating random birthdays
import matplotlib.pyplot as plt  # To display the output

random.seed(2021)

birthday_lists = []

n_min = 5
n_max = 200
n_diff = 5
n_iters = 10000

# Generating random birthdays
for i in range(n_iters):
    l = []
    for i in range(n_max):
        l.append(random.randint(0, 365))
    birthday_lists.append(l)


def check_repeat_birthday(l):

    l_set = set(l)  # Creating a set from the list

    # Return True or False based on whether there is a repeating birthday
    if len(l) == len(l_set):
        return False
    else:
        return True


n_array = []
p_array = []

for n in range(n_min, n_max + 1, n_diff):
    t = 0  # Counter to keep track of number of successes
    for i in range(n_iters):
        if check_repeat_birthday(birthday_lists[i][:n]):
            t += 1
    # Finding the probability as the fraction of times we have overlapping birthdays
    n_array.append(n)
    p_array.append(t / n_iters)

# Visualizing the result via plot
plt.plot(n_array, p_array)
plt.scatter(n_array, p_array)
plt.title("Number of People vs Probability of Shared Birthday")
plt.xlabel("Number of People")
plt.ylabel("Probability of Shared Birthday")
plt.grid()
plt.show()
```



Q8. Write a Python program that outputs all possible strings formed by using the characters ‘c’, ‘a’, ‘t’, ‘d’, ‘o’, and ‘g’ exactly once.

Solution : $O(n!)$ Time Complexity

To find all the permutations using the given characters, we need to use the concept of recursion and backtracking. We define the function to find the permutations as following :

- We start from the beginning of the string, iterate till the end of the string. The function takes in *step* value as input which denotes until what point the recursion has proceeded.
- In each iteration we swap the current and *step* element in the string and we recursively apply the function on the part which has not been swapped.
- When *step* becomes equal to the number of characters, we stop and print the string.

This is of $O(n!)$ time complexity where n is the number of characters, since there are $n!$ permutations of the string.

In [12]:

```python
"""
Python program to find all the strings generated using
‘c’, ‘a’, ‘t’, ‘d’, ‘o’, and ‘g’
"""

c = 0  # Counter to check number of strings generated


def permutation_string(l, step):
    global c  # Defining the global variable inside the function

    # If step is equal to length of l we print the resulting array
    if step == len(l):
        c += 1
        if (c - 1) % 5 == 0:
            print()
        print("{:3} : {}".format(c, "".join(l)),
              end="\t")  # Formatted printing
    else:
        # Loop to find the iterations over step-len(l)
        for i in range(step, len(l)):
            l[i], l[step] = (
                l[step],
                l[i],
            )  # We swap i with step to create new permutation
            permutation_string(
                l, step + 1)  # Recursive call to create further permutations


l = ["c", "a", "t", "d", "o", "g"]
permutation_string(l, 0)
```

```
  1 : catdog	  2 : catdgo	  3 : catgdo	  4 : catgod	  5 : catdog
  6 : catdgo	  7 : cadtgo	  8 : cadtog	  9 : cadotg	 10 : cadogt
 11 : cadtgo	 12 : cadtog	 13 : caotdg	 14 : caotgd	 15 : caogtd
 16 : caogdt	 17 : caotdg	 18 : caotgd	 19 : cadtgo	 20 : cadtog
 21 : cadotg	 22 : cadogt	 23 : cadtgo	 24 : cadtog	 25 : cdatog
 26 : cdatgo	 27 : cdagto	 28 : cdagot	 29 : cdatog	 30 : cdatgo
 31 : cdtago	 32 : cdtaog	 33 : cdtoag	 34 : cdtoga	 35 : cdtago
 36 : cdtaog	 37 : cdoatg	 38 : cdoagt	 39 : cdogat	 40 : cdogta
 41 : cdoatg	 42 : cdoagt	 43 : cdtago	 44 : cdtaog	 45 : cdtoag
 46 : cdtoga	 47 : cdtago	 48 : cdtaog	 49 : catdog	 50 : catdgo
 51 : catgdo	 52 : catgod	 53 : catdog	 54 : catdgo	 55 : cadtgo
 56 : cadtog	 57 : cadotg	 58 : cadogt	 59 : cadtgo	 60 : cadtog
 61 : caotdg	 62 : caotgd	 63 : caogtd	 64 : caogdt	 65 : caotdg
 66 : caotgd	 67 : cadtgo	 68 : cadtog	 69 : cadotg	 70 : cadogt
 71 : cadtgo	 72 : cadtog	 73 : codtag	 74 : codtga	 75 : codgta
 76 : codgat	 77 : codtag	 78 : codtga	 79 : cotdga	 80 : cotdag
 81 : cotadg	 82 : cotagd	 83 : cotdga	 84 : cotdag	 85 : coadtg
 86 : coadgt	 87 : coagdt	 88 : coagtd	 89 : coadtg	 90 : coadgt
 91 : cotdga	 92 : cotdag	 93 : cotadg	 94 : cotagd	 95 : cotdga
 96 : cotdag	 97 : cgtdao	 98 : cgtdoa	 99 : cgtoda	100 : cgtoad
101 : cgtdao	102 : cgtdoa	103 : cgdtoa	104 : cgdtao	105 : cgdato
106 : cgdaot	107 : cgdtoa	108 : cgdtao	109 : cgatdo	110 : cgatod
111 : cgaotd	112 : cgaodt	113 : cgatdo	114 : cgatod	115 : cgdtoa
116 : cgdtao	117 : cgdato	118 : cgdaot	119 : cgdtoa	120 : cgdtao
121 : gcdtao	122 : gcdtoa	123 : gcdota	124 : gcdoat	125 : gcdtao
126 : gcdtoa	127 : gctdoa	128 : gctdao	129 : gctado	130 : gctaod
131 : gctdoa	132 : gctdao	133 : gcadto	134 : gcadot	135 : gcaodt
136 : gcaotd	137 : gcadto	138 : gcadot	139 : gctdoa	140 : gctdao
141 : gctado	142 : gctaod	143 : gctdoa	144 : gctdao	145 : gtcdao
146 : gtcdoa	147 : gtcoda	148 : gtcoad	149 : gtcdao	150 : gtcdoa
151 : gtdcoa	152 : gtdcao	153 : gtdaco	154 : gtdaoc	155 : gtdcoa
156 : gtdcao	157 : gtacdo	158 : gtacod	159 : gtaocd	160 : gtaodc
161 : gtacdo	162 : gtacod	163 : gtdcoa	164 : gtdcao	165 : gtdaco
166 : gtdaoc	167 : gtdcoa	168 : gtdcao	169 : gcdtao	170 : gcdtoa
171 : gcdota	172 : gcdoat	173 : gcdtao	174 : gcdtoa	175 : gctdoa
176 : gctdao	177 : gctado	178 : gctaod	179 : gctdoa	180 : gctdao
181 : gcadto	182 : gcadot	183 : gcaodt	184 : gcaotd	185 : gcadto
186 : gcadot	187 : gctdoa	188 : gctdao	189 : gctado	190 : gctaod
191 : gctdoa	192 : gctdao	193 : gatdco	194 : gatdoc	195 : gatodc
196 : gatocd	197 : gatdco	198 : gatdoc	199 : gadtoc	200 : gadtco
201 : gadcto	202 : gadcot	203 : gadtoc	204 : gadtco	205 : gactdo
206 : gactod	207 : gacotd	208 : gacodt	209 : gactdo	210 : gactod
211 : gadtoc	212 : gadtco	213 : gadcto	214 : gadcot	215 : gadtoc
216 : gadtco	217 : godtca	218 : godtac	219 : godatc	220 : godact
221 : godtca	222 : godtac	223 : gotdac	224 : gotdca	225 : gotcda
226 : gotcad	227 : gotdac	228 : gotdca	229 : gocdta	230 : gocdat
231 : gocadt	232 : gocatd	233 : gocdta	234 : gocdat	235 : gotdac
236 : gotdca	237 : gotcda	238 : gotcad	239 : gotdac	240 : gotdca
241 : togdca	242 : togdac	243 : togadc	244 : togacd	245 : togdca
246 : togdac	247 : todgac	248 : todgca	249 : todcga	250 : todcag
251 : todgac	252 : todgca	253 : tocgda	254 : tocgad	255 : tocagd
256 : tocadg	257 : tocgda	258 : tocgad	259 : todgac	260 : todgca
261 : todcga	262 : todcag	263 : todgac	264 : todgca	265 : tdogca
266 : tdogac	267 : tdoagc	268 : tdoacg	269 : tdogca	270 : tdogac
271 : tdgoac	272 : tdgoca	273 : tdgcoa	274 : tdgcao	275 : tdgoac
276 : tdgoca	277 : tdcoga	278 : tdcoag	279 : tdcaog	280 : tdcago
281 : tdcoga	282 : tdcoag	283 : tdgoac	284 : tdgoca	285 : tdgcoa
286 : tdgcao	287 : tdgoac	288 : tdgoca	289 : togdca	290 : togdac
291 : togadc	292 : togacd	293 : togdca	294 : togdac	295 : todgac
296 : todgca	297 : todcga	298 : todcag	299 : todgac	300 : todgca
301 : tocgda	302 : tocgad	303 : tocagd	304 : tocadg	305 : tocgda
306 : tocgad	307 : todgac	308 : todgca	309 : todcga	310 : todcag
311 : todgac	312 : todgca	313 : tcgdoa	314 : tcgdao	315 : tcdago
316 : tcdaog	317 : tcdgoa	318 : tcdgao	319 : tcgdao	320 : tcgdoa
321 : tcgoda	322 : tcgoad	323 : tcgdao	324 : tcgdoa	325 : tcodga
326 : tcodag	327 : tcoadg	328 : tcoagd	329 : tcodga	330 : tcodag
331 : tcgdao	332 : tcgdoa	333 : tcgoda	334 : tcgoad	335 : tcgdao
336 : tcgdoa	337 : tagdoc	338 : tagdco	339 : tagcdo	340 : tagcod
341 : tagdoc	342 : tagdco	343 : tadgco	344 : tadgoc	345 : tadogc
346 : tadocg	347 : tadgco	348 : tadgoc	349 : taogdc	350 : taogcd
351 : taocgd	352 : taocdg	353 : taogdc	354 : taogcd	355 : tadgco
356 : tadgoc	357 : tadogc	358 : tadocg	359 : tadgco	360 : tadgoc
361 : gadtoc	362 : gadtco	363 : gadcto	364 : gadcot	365 : gadtoc
366 : gadtco	367 : gatdco	368 : gatdoc	369 : gatodc	370 : gatocd
371 : gatdco	372 : gatdoc	373 : gaodtc	374 : gaodct	375 : gaocdt
376 : gaoctd	377 : gaodtc	378 : gaodct	379 : gatdco	380 : gatdoc
381 : gatodc	382 : gatocd	383 : gatdco	384 : gatdoc	385 : gtadoc
386 : gtadco	387 : gtacdo	388 : gtacod	389 : gtadoc	390 : gtadco
391 : gtdaco	392 : gtdaoc	393 : gtdoac	394 : gtdoca	395 : gtdaco
396 : gtdaoc	397 : gtoadc	398 : gtoacd	399 : gtocad	400 : gtocda
401 : gtoadc	402 : gtoacd	403 : gtdaco	404 : gtdaoc	405 : gtdoac
406 : gtdoca	407 : gtdaco	408 : gtdaoc	409 : gadtoc	410 : gadtco
411 : gadcto	412 : gadcot	413 : gadtoc	414 : gadtco	415 : gatdco
416 : gatdoc	417 : gatodc	418 : gatocd	419 : gatdco	420 : gatdoc
421 : gaodtc	422 : gaodct	423 : gaocdt	424 : gaoctd	425 : gaodtc
426 : gaodct	427 : gatdco	428 : gatdoc	429 : gatodc	430 : gatocd
431 : gatdco	432 : gatdoc	433 : gotdac	434 : gotdca	435 : gotcda
436 : gotcad	437 : gotdac	438 : gotdca	439 : godtca	440 : godtac
441 : godatc	442 : godact	443 : godtca	444 : godtac	445 : goatdc
446 : goatcd	447 : goactd	448 : goacdt	449 : goatdc	450 : goatcd
451 : godtca	452 : godtac	453 : godatc	454 : godact	455 : godtca
456 : godtac	457 : gcdtao	458 : gcdtoa	459 : gcdota	460 : gcdoat
461 : gcdtao	462 : gcdtoa	463 : gctdoa	464 : gctdao	465 : gctado
466 : gctaod	467 : gctdoa	468 : gctdao	469 : gcadto	470 : gcadot
471 : gcaodt	472 : gcaotd	473 : gcadto	474 : gcadot	475 : gctdoa
476 : gctdao	477 : gctado	478 : gctaod	479 : gctdoa	480 : gctdao
481 : actdgo	482 : actdog	483 : actodg	484 : actogd	485 : actdgo
486 : actdog	487 : acdtog	488 : acdtgo	489 : acdgto	490 : acdgot
491 : acdtog	492 : acdtgo	493 : acgtdo	494 : acgtod	495 : acgotd
496 : acgodt	497 : acgtdo	498 : acgtod	499 : acdtog	500 : acdtgo
501 : acdgto	502 : acdgot	503 : acdtog	504 : acdtgo	505 : adctgo
506 : adctog	507 : adcotg	508 : adcogt	509 : adctgo	510 : adctog
511 : adtcog	512 : adtcgo	513 : adtgco	514 : adtgoc	515 : adtcog
516 : adtcgo	517 : adgcto	518 : adgcot	519 : adgoct	520 : adgotc
521 : adgcto	522 : adgcot	523 : adtcog	524 : adtcgo	525 : adtgco
526 : adtgoc	527 : adtcog	528 : adtcgo	529 : actdgo	530 : actdog
531 : actodg	532 : actogd	533 : actdgo	534 : actdog	535 : acdtog
536 : acdtgo	537 : acdgto	538 : acdgot	539 : acdtog	540 : acdtgo
541 : acgtdo	542 : acgtod	543 : acgotd	544 : acgodt	545 : acgtdo
546 : acgtod	547 : acdtog	548 : acdtgo	549 : acdgto	550 : acdgot
551 : acdtog	552 : acdtgo	553 : agdtco	554 : agdtoc	555 : agdotc
556 : agdoct	557 : agdtco	558 : agdtoc	559 : agtdoc	560 : agtdco
561 : agtcdo	562 : agtcod	563 : agtdoc	564 : agtdco	565 : agcdto
566 : agcdot	567 : agcodt	568 : agcotd	569 : agcdto	570 : agcdot
571 : agtdoc	572 : agtdco	573 : agtcdo	574 : agtcod	575 : agtdoc
576 : agtdco	577 : aotdcg	578 : aotdgc	579 : aotgdc	580 : aotgcd
581 : aotdcg	582 : aotdgc	583 : aodtgc	584 : aodtcg	585 : aodctg
586 : aodcgt	587 : aodtgc	588 : aodtcg	589 : aoctdg	590 : aoctgd
591 : aocgtd	592 : aocgdt	593 : aoctdg	594 : aoctgd	595 : aodtgc
596 : aodtcg	597 : aodctg	598 : aodcgt	599 : aodtgc	600 : aodtcg
601 : godtca	602 : godtac	603 : godatc	604 : godact	605 : godtca
606 : godtac	607 : gotdac	608 : gotdca	609 : gotcda	610 : gotcad
611 : gotdac	612 : gotdca	613 : gocdta	614 : gocdat	615 : gocadt
616 : gocatd	617 : gocdta	618 : gocdat	619 : gotdac	620 : gotdca
621 : gotcda	622 : gotcad	623 : gotdac	624 : gotdca	625 : gtodca
626 : gtodac	627 : gtoadc	628 : gtoacd	629 : gtodca	630 : gtodac
631 : gtdoac	632 : gtdoca	633 : gtdcoa	634 : gtdcao	635 : gtdoac
636 : gtdoca	637 : gtcoda	638 : gtcoad	639 : gtcaod	640 : gtcado
641 : gtcoda	642 : gtcoad	643 : gtdoac	644 : gtdoca	645 : gtdcoa
646 : gtdcao	647 : gtdoac	648 : gtdoca	649 : godtca	650 : godtac
651 : godatc	652 : godact	653 : godtca	654 : godtac	655 : gotdac
656 : gotdca	657 : gotcda	658 : gotcad	659 : gotdac	660 : gotdca
661 : gocdta	662 : gocdat	663 : gocadt	664 : gocatd	665 : gocdta
666 : gocdat	667 : gotdac	668 : gotdca	669 : gotcda	670 : gotcad
671 : gotdac	672 : gotdca	673 : gctdoa	674 : gctdao	675 : gctado
676 : gctaod	677 : gctdoa	678 : gctdao	679 : gcdtao	680 : gcdtoa
681 : gcdota	682 : gcdoat	683 : gcdtao	684 : gcdtoa	685 : gcotda
686 : gcotad	687 : gcoatd	688 : gcoadt	689 : gcotda	690 : gcotad
691 : gcdtao	692 : gcdtoa	693 : gcdota	694 : gcdoat	695 : gcdtao
696 : gcdtoa	697 : gadtoc	698 : gadtco	699 : gadcto	700 : gadcot
701 : gadtoc	702 : gadtco	703 : gatdco	704 : gatdoc	705 : gatodc
706 : gatocd	707 : gatdco	708 : gatdoc	709 : gaodtc	710 : gaodct
711 : gaocdt	712 : gaoctd	713 : gaodtc	714 : gaodct	715 : gatdco
716 : gatdoc	717 : gatodc	718 : gatocd	719 : gatdco	720 : gatdoc
```

Q9. Write a Python program that can take a positive integer greater than 2 as input and write out the number of times one must repeatedly divide this number by 2 before getting a value less than 2.

Solution 1 : $O(log(n))$ Time Complexity

We keep dividing the number by 2 until the result becomes less than 2. This will take $O(log(n))$ time since we need to do at least log(n) divisions

In [13]:

```python
"""
Python program to find number of times to divide a
given number by 2 to get a value less than 2
"""

n = int(input("Enter the number : "))  # Getting the number as input

c = 0  # Counter to count number of divisions done

k = n  # Placeholder for repeated division

# Run the loop until the number becomes less than 2
while k >= 2:
    k = k / 2  # Dividing by 2
    c += 1  # Increasing the counter

print(
    "\nWe need to divide",
    n,
    "by 2 atleast\033[1m",
    c,
    "\033[0;0mtimes to get",
    k,
    "< 2",
)  # Printing the answer
```

```
Enter the number : 6

We need to divide 6 by 2 atleast 2 times to get 1.5 < 2
```

Solution 2 : $O(1)$ Time Complexity

We can get the same answer above by taking the $\lfloor log_2(n) \rfloor$

In [14]:

```python
"""
Python program to find number of times to divide a
given number by 2 to get a value less than 2
"""

from math import log2, floor

n = int(input("Enter the number : "))  # Getting the number as input

c = floor(log2(n))

k = n / 2**(c)
print(
    "\nWe need to divide",
    n,
    "by 2 atleast\033[1m",
    c,
    "\033[0;0mtimes to get",
    k,
    "< 2",
)  # Printing the answer
```

```
Enter the number : 6

We need to divide 6 by 2 atleast 2 times to get 1.5 < 2
```