reverse a number using recursion

In [4]:
```
def recursum(number,reverse):
  if number==0:
    return reverse
  remainder = int(number%10)
  reverse = (reverse*10)+remainder
  return recursum(int(number/10),reverse)
num = 123
reverse = 0
print(recursum(num,reverse))
```

```
321
```

perfect number

In [8]:
```
n=28
sum = 0
for i in range(1, n):
    if n % i == 0:
        sum = sum + i
if sum == n:
        print("Perfect number")
else:
    print(" not a Perfect number")
```

```
Perfect number
```

intersection

In [23]:
```
def intersection(a,b):
   return (a&b)
set1={1,2,3,4,5}
set2={4,5,6,7,8}
res=intersection(set1,set2)
print(res)
```

```
{4, 5}
```

array of integers half odd and even

In [2]:
```
nums = [1, 2, 3, 4, 5, 6]
half_odd_even = [i for i in nums if i % 2 == 0] + [i for i in nums if i % 2
print(half_odd_even)
```

```
[2, 4, 6, 1, 3, 5]
```

intersect of two array

In [9]:
```
from collections import Counter
def intersect(nums1, nums2):
    count1, count2 = Counter(nums1), Counter(nums2)
    return list((count1 & count2).elements())
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersect(nums1, nums2))
```

```
[2, 2]
```

non-recursive and recursive

In [10]:
```
def factorial(n, method='recursive'):
    if method == 'recursive':
        if n == 0:
            return 1
        else:
            return n * factorial(n - 1, method)
    elif method == 'non-recursive':
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
    else:
        return "Invalid method. Choose 'recursive' or 'non-recursive'."
print(factorial(5, method='recursive'))
print(factorial(5, method='non-recursive'))
```

```
120
120
```

master theorem

In [11]:
```
def algorithm_analysis(method):
    if method == "master_theorem":
        def master_theorem(a, b, k):
            return f"T(n) = O(n^{k})"
        return master_theorem
    elif method == "substitution_method":
        return "T(n) = O(nlogn)"
    elif method == "iteration_method":
        return "T(n) = O(n^2)"
    else:
        return "Invalid method."
method = "master_theorem"
analysis_function = algorithm_analysis(method)
print(analysis_function(2, 3, 2))
```

```
T(n) = O(n^2)
```

array of asce order

In [12]:
```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
nums = [12, 4, 7, 1, 9, 3]
sorted_nums = merge_sort(nums)
print(sorted_nums)
```

```
[1, 3, 4, 7, 9, 12]
```

10th one

In [17]:
```
def sort(nums):
    odd=[]
    even=[]
    res=[]
    for num in nums:
        if num%2==0:
            even.append(num)
        else:
            odd.append(num)
    for i in range(len(nums)):
        if i%2==0:
            res.append(even.pop())
        else:
            res.append(odd.pop())
    return res
nums=[1,2,6,7]
sorted_num= sort(nums)
print(sorted_num)
```

```
[6, 7, 2, 1]
```

time complixity

In [18]:
```
#o(n)
def linear_search(data, value):
    for index in range(len(data)):
        if data[index] == value:
            return index
    return -1
data = [2, 4, 6, 8, 10]
value = 6
print(linear_search(data,value))

#o(n^2)
def bubble_sort(data):
    n = len(data)
    for i in range(n):
        for j in range(0, n - i - 1):
            if data[j] > data[j + 1]:
                data[j], data[j + 1] = data[j + 1], data[j]
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("the sorted array is.....",data)
```

```
2
the sorted array is..... [11, 12, 22, 25, 34, 64, 90]
```

In [ ]: