Practice Problems Solutions for Python (T2) (Chapter 3,4 and 5) with

1. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings. Example:

Input: ['abc', 'xyz', 'aba', '1221']

Output: 2

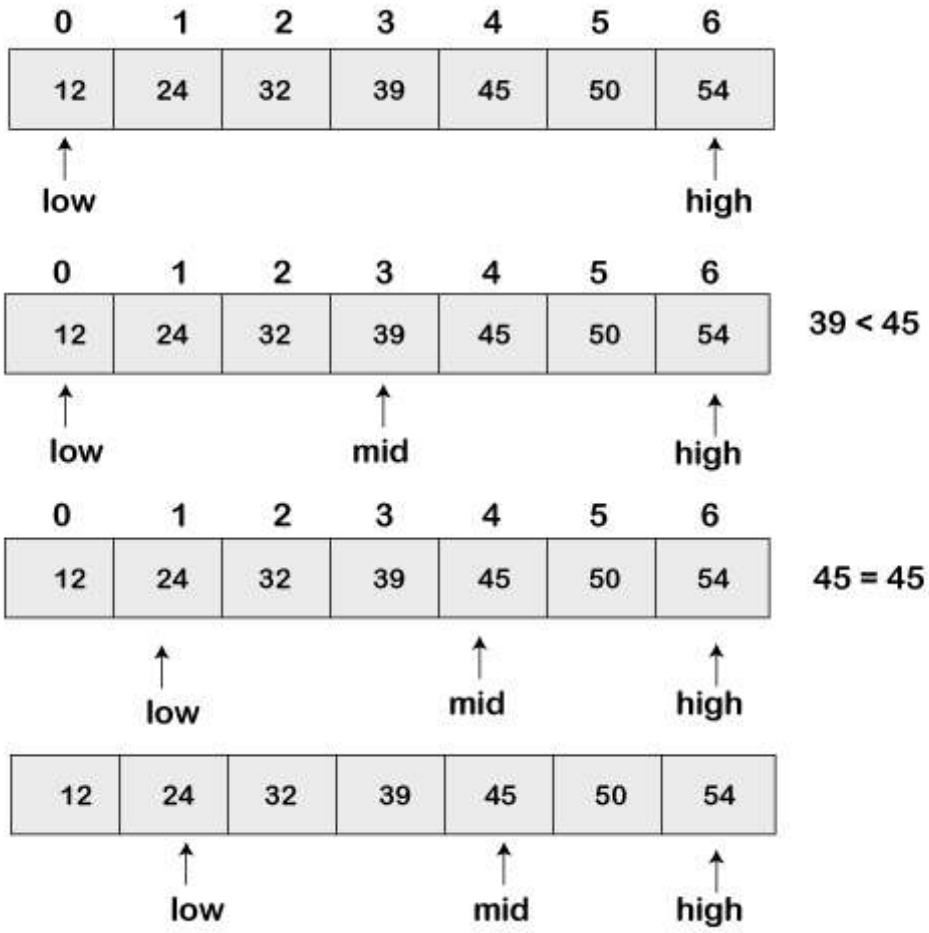
In [ ]:
```
def match_words(words):
  ctr = 0

  for word in words:
    if len(word) > 1 and word[0] == word[-1]:
      ctr += 1
  return ctr

print(match_words(['abc', 'xyz', 'aba', '1221']))
```

```
2
```

2. Write a python program for binary search.

Binary Search : In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

Step by step example :



Output: Element is present at index 3

In [ ]:
```
def binary_search(arr, low, high, x):

    # Check base case
    if high >= low:

        mid = (high + low) // 2

        # If 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
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)

        # Else the element can only be present in right subarray
```

```
        else:

            return binary_search(arr, mid + 1, high, x)


    else:

        # Element is not present in the array

        return -1

# Test array

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

x = 10

# Function call

result = binary_search(arr, 0, len(arr)-1, x)


if result != -1:

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

else:

    print("Element is not present")
```

```
Element is present at index 3
```

3. Write a python program to find the indices of all occurrences of target in the uneven matrix.

An irregular/uneven matrix, or ragged matrix, is a matrix that has a different number of elements in each row. Ragged matrices are not used in linear algebra, since standard matrix transformations cannot be performed on them, but they are useful as arrays in computing.

Input: [([1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19]), 19]

Output: [[0, 4], [1, 0], [1, 3], [4, 1]]

Input: [([1, 2, 3, 2], [], [7, 9, 2, 1, 4]),2]

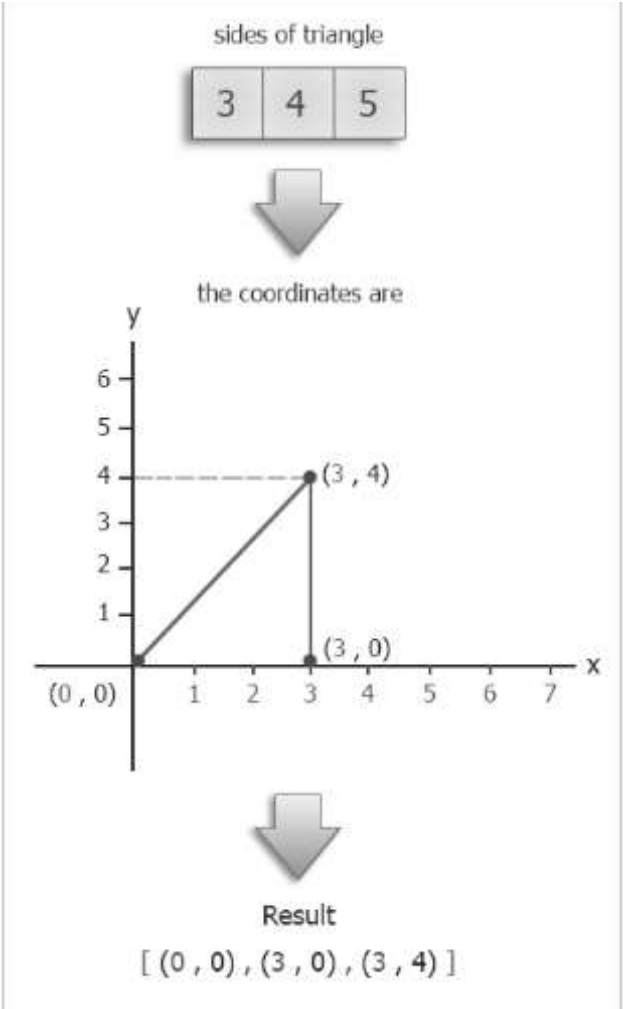
Output: [[0, 1], [0, 3], [2, 2]]

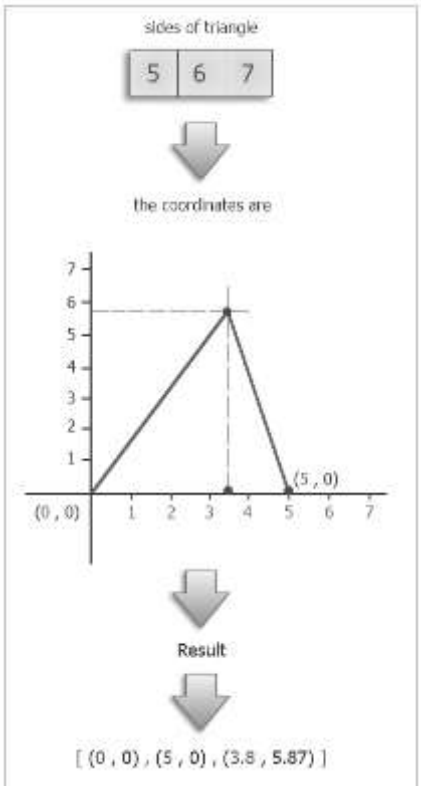
In [42]:
```
def test(M, T):
    l=[]
    for i,row in enumerate(M):
        for j,n in enumerate(row):
            if n==T:
                l.append([i,j])
    print("Indices of all occurrences of the target value in the said uneven matrix
M = [[1, 2, 3, 2], [], [7, 9, 2, 1, 4]]
print("Matrix:",M)
T = 2
```

```
print("Target value:",T)
test(M,T)
M = [[1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19]]
T = 19
print("Matrix:")
print(M)
print("Target value:")
print(T)
test(M,T)
```

```
Matrix: [[1, 2, 3, 2], [], [7, 9, 2, 1, 4]]
Target value: 2
Indices of all occurrences of the target value in the said uneven matrix: [[0, 1],
[0, 3], [2, 2]]
Matrix:
[[1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19]]
Target value:
19
Indices of all occurrences of the target value in the said uneven matrix: [[0, 4],
[1, 0], [1, 3], [4, 1]]
```

4. Write a python program to find the coordinates of a triangle with the given side lengths.





Heron's formula to find area (s * (s - a) * (s - b) * (s - c)) ** 0.5 where s=semi perimeter a,b,c are the lengths of sides

Input: [3, 4, 5]

Output: [[0.0, 0.0], [3, 0.0], [3.0, 4.0]]

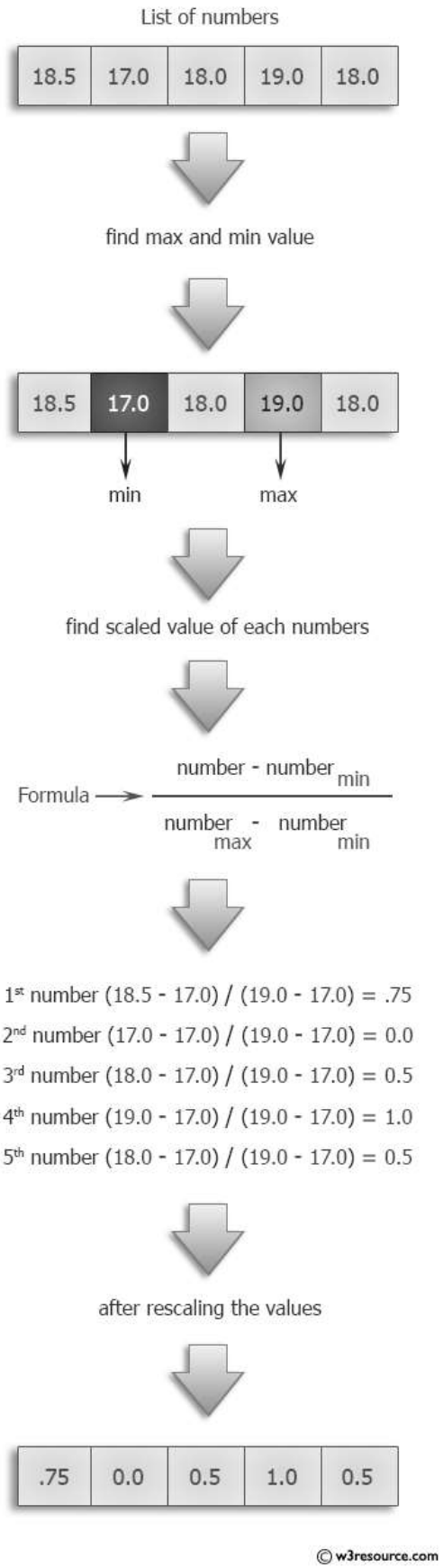
Input: [5, 6, 7]

Output: [[0.0, 0.0], [5, 0.0], [3.8, 5.878775382679628]]

In [18]:
```
def test(sides):
    a, b, c = sorted(sides)
    s = sum(sides) / 2 # semi-perimeter
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5 # Heron's formula
    y = 2 * area / a # height
    x = (c ** 2 - y ** 2) ** 0.5
    return [[0.0, 0.0], [a, 0.0], [x, y]]
sides = [3, 4, 5]
print("Sides of the triangle:",sides)
print("Coordinates of a triangle with the said side lengths:")
print(test(sides))
sides = [5, 6, 7]
print("\nSides of the triangle:",sides)
print("Coordinates of a triangle with the said side lengths:")
print(test(sides))
```

```
Sides of the triangle: [3, 4, 5]
Coordinates of a triangle with the said side lengths:
[[0.0, 0.0], [3, 0.0], [3.0, 4.0]]

Sides of the triangle: [5, 6, 7]
Coordinates of a triangle with the said side lengths:
[[0.0, 0.0], [5, 0.0], [3.8, 5.878775382679628]]
```

5.Write a python program to rescale and shift numbers of a given list, so that they cover the range [0, 1].



Input: [18.5, 17.0, 18.0, 19.0, 18.0] Output: [0.75, 0.0, 0.5, 1.0, 0.5]

Input: [13.0, 17.0, 17.0, 15.5, 2.94] Output: [0.7155049786628734, 1.0, 1.0, 0.8933143669985776, 0.0]

In [ ]:
```
def test(nums):
    a = min(nums)
    b = max(nums)
    if b - a == 0:
        return [0.0] + [1.0] * (len(nums) - 1)
    for i in range(len(nums)):
        nums[i] = (nums[i] - a) / (b - a)
    return nums

nums = [18.5, 17.0, 18.0, 19.0, 18.0]
print("Original list:")
print(nums)
print("Rescale and shift the numbers of the said list so that they cover the range
print(test(nums))
nums = [13.0, 17.0, 17.0, 15.5, 2.94]
print("\nOriginal list:")
print(nums)
print("Rescale and shift the numbers of the said list so that they cover the range
print(test(nums))
```

```
Original list:
[18.5, 17.0, 18.0, 19.0, 18.0]
Rescale and shift the numbers of the said list so that they cover the range [0,
1]:
[0.75, 0.0, 0.5, 1.0, 0.5]

Original list:
[13.0, 17.0, 17.0, 15.5, 2.94]
Rescale and shift the numbers of the said list so that they cover the range [0,
1]:
[0.7155049786628734, 1.0, 1.0, 0.8933143669985776, 0.0]
```

6. Write a python program to find the numbers that are greater than 10 and have odd first and last digits.

Input: [1, 3, 79, 10, 4, 1, 39, 62]

Output: [79, 39]

Input: [11, 31, 77, 93, 48, 1, 57]

Output: [11, 31, 77, 93, 57]

In [22]:
```
def test(nums):
    l=[]
    for x in nums:
        if x>10 and x%10%2 and int(str(x)[0])%2:
            l.append(x)
    print(l)
nums = [1, 3, 79, 10, 4, 1, 39]
print("Original list of numbers:")
print(nums)
print("Numbers of the said array that are greater than 10 and have odd first and la
```

```
(test(nums))
nums = [11, 31, 77, 93, 48, 1, 57]
print("\nOriginal list of numbers:")
print(nums)
print("Numbers of the said array that are greater than 10 and have odd first and la
(test(nums))
```

```
Original list of numbers:
[1, 3, 79, 10, 4, 1, 39]
Numbers of the said array that are greater than 10 and have odd first and last dig
its:
[79, 39]

Original list of numbers:
[11, 31, 77, 93, 48, 1, 57]
Numbers of the said array that are greater than 10 and have odd first and last dig
its:
[11, 31, 77, 93, 57]
```

7. Write a python program to shift the decimal digits n places to the left, wrapping the extra digits around. If shift > the number of digits of n, reverse the string.

Input: n = 12345 and shift = 1

Output: Result = 23451

Input: n = 12345 and shift = 2

Output: Result = 34512

Input: n = 12345 and shift = 3

Output: Result = 45123

Input: n = 12345 and shift = 5

Output: Result = 12345

Input: n = 12345 and shift = 6

Output: Result = 54321

In [ ]:
```
def test(n, shift):
    s = str(n)
    if shift > len(s):
        return s[::-1]
    return s[shift:] + s[:shift]

print("Shift the decimal digits n places to the left. If shift > the number of digi

n = 12345
shift = 1
print("\nn =",n," and shift =",shift)
print("Result = ",test(n, shift))
n = 12345
```

```
shift = 2
print("\nn =",n," and shift =",shift)
print("Result = ",test(n, shift))
n = 12345
shift = 3
print("\nn =",n," and shift =",shift)
print("Result = ",test(n, shift))
n = 12345
shift = 5
print("\nn =",n," and shift =",shift)
print("Result = ",test(n, shift))
n = 12345
shift = 6
print("\nn =",n," and shift =",shift)
print("Result = ",test(n, shift))
```

```
Shift the decimal digits n places to the left. If shift > the number of digits of
n, reverse the string.:

n = 12345  and shift = 1
Result =  23451

n = 12345  and shift = 2
Result =  34512

n = 12345  and shift = 3
Result =  45123

n = 12345  and shift = 5
Result =  12345

n = 12345  and shift = 6
Result =  54321
```

8. Write a python program to create a list containing that number in between each pair of adjacent numbers.

Given a list of numbers and a number to inject,

Input: [12, -7, 3, -89, 14, 88, -78, -1, 2, 7] Separator: 6

Output: [12, 6, -7, 6, 3, 6, -89, 6, 14, 6, 88, 6, -78, 6, -1, 6, 2, 6, 7]

Input: [1, 2, 3, 4, 5, 6] Separator: 9

Output: [1, 9, 2, 9, 3, 9, 4, 9, 5, 9, 6]

In [ ]:
```
def test(nums, sep):
    ans = [sep] * (2 * len(nums) - 1)
    ans[::2] = nums
    return ans
nums = [12, -7, 3, -89, 14, 88, -78, -1, 2, 7]
separator = 6
print("List of numbers:",nums)
print("Separator:",separator)
print("Inject the separator in between each pair of adjacent numbers of the said li
print(test(nums,separator))
```

```
nums = [1, 2, 3, 4, 5, 6]
separator = 9
print("\nList of numbers:",nums)
print("Separator:",separator)
print("Inject the separator in between each pair of adjacent numbers of the said li
print(test(nums,separator))
```

```
List of numbers: [12, -7, 3, -89, 14, 88, -78, -1, 2, 7]
Separator: 6
Inject the separator in between each pair of adjacent numbers of the said list:
[12, 6, -7, 6, 3, 6, -89, 6, 14, 6, 88, 6, -78, 6, -1, 6, 2, 6, 7]

List of numbers: [1, 2, 3, 4, 5, 6]
Separator: 9
Inject the separator in between each pair of adjacent numbers of the said list:
[1, 9, 2, 9, 3, 9, 4, 9, 5, 9, 6]
```

9. Write a python program to find two indices making a given string unhappy.

A string is happy if every three consecutive characters are distinct.

Input: Python

Output: None

Input: Unhappy

Output: [4, 5]

Input: Find

Output: None

Input: Street

Output: [3, 4]

In [ ]:
```
def test(s):
    for i in range(len(s) - 2):
        if s[i] == s[i + 1]:
            return [i, i + 1]
        if s[i] == s[i + 2]:
            return [i, i + 2]

strs = "Python"
print("Original string:",strs)
print("Find two indices making the said string unhappy!")
print(test(strs))
strs = "Unhappy"
print("\nOriginal string:",strs)
print("Find two indices making the said string unhappy!")
print(test(strs))
strs = "Find"
print("\nOriginal string:",strs)
print("Find two indices making the said string unhappy!")
print(test(strs))
strs = "Street"
```

```
print("\nOriginal string:",strs)
print("Find two indices making the said string unhappy!")
print(test(strs))
```

```
Original string: Python
Find two indices making the said string unhappy!
None

Original string: Unhappy
Find two indices making the said string unhappy!
[4, 5]

Original string: Find
Find two indices making the said string unhappy!
None

Original string: Street
Find two indices making the said string unhappy!
[3, 4]
```

10. Write a python program to start with a list of integers, keep every other element in place and otherwise sort the list.

Input: [2, 5, 6, 3, 1, 4, 34]

Output: [1, 5, 2, 3, 6, 4, 34]

Input: [8, 0, 7, 2, 9, 4, 1, 2, 8, 3]

Output: [1, 0, 7, 2, 8, 4, 8, 2, 9, 3]

In [ ]:
```
def test(nums):
    li = nums.copy()
    for i in range(len(li)):
        if i % 2 == 0:
            for j in range(i+2, len(li), 2):
                if li[j] < li[i]:
                    swap(li, i, j)

    return li
def swap(li, i, j):
    temp = li[i]
    li[i] = li[j]
    li[j] = temp
nums = [2, 5, 6, 3, 1, 4, 34]
print("Original list (triple) of lists:")
print(nums)
print("In the said list, keep every other element in place and otherwise sort the l
print(test(nums))
nums = [8, 0, 7, 2, 9, 4, 1, 2, 8, 3]
print("\nOriginal list (triple) of lists:")
print(nums)
print("In the said list, keep every other element in place and otherwise sort the l
print(test(nums))
```

```
Original list (triple) of lists:
[2, 5, 6, 3, 1, 4, 34]
In the said list, keep every other element in place and otherwise sort the list.:
[1, 5, 2, 3, 6, 4, 34]

Original list (triple) of lists:
[8, 0, 7, 2, 9, 4, 1, 2, 8, 3]
In the said list, keep every other element in place and otherwise sort the list.:
[1, 0, 7, 2, 8, 4, 8, 2, 9, 3]
```

11.Write a python program to get the single digits in numbers sorted backwards and converted to English words.

Input: [1, 3, 4, 5, 11]

Output: ['five', 'four', 'three', 'one']

Input: [27, 3, 8, 5, 1, 31]

Output: ['eight', 'five', 'three', 'one']

In [ ]:
```
def test(nums):
    digits = {"zero": None,
              "one": 1,
              "two": 2,
              "three": 3,
              "four": 4,
              "five": 5,
              "six": 6,
              "seven": 7,
              "eight": 8,
              "nine": 9}
    digits_backwards = {digits[k]: k for k in digits}
    digits = [digits[s] for s in digits]
    li = [digits[n] for n in nums if n in digits]
    return [digits_backwards[n] for n in sorted(li, reverse=True)]

nums = [1, 3, 4, 5, 11]
print("Original list of numbers:")
print(nums)
print("Return the single digits in nums sorted backwards and converted to English w
print(test(nums))
nums = [27, 3, 8, 5, 1, 31]
print("\nOriginal list of numbers:")
print(nums)
print("Return the single digits in nums sorted backwards and converted to English w
print(test(nums))
```

```
Original list of numbers:
[1, 3, 4, 5, 11]
Return the single digits in nums sorted backwards and converted to English words:
['five', 'four', 'three', 'one']

Original list of numbers:
[27, 3, 8, 5, 1, 31]
Return the single digits in nums sorted backwards and converted to English words:
['eight', 'five', 'three', 'one']
```

12. Write a Python program to find the following strange sort of list of numbers: the first element is the smallest, the second is the largest of the remaining, the third is the smallest of the remaining, the fourth is the smallest of the remaining, etc.

Input: [1, 3, 4, 5, 11]

Output: [1, 11, 3, 5, 4]

Input: [27, 3, 8, 5, 1, 31]

Output: [1, 31, 3, 27, 5, 8]

Input: [1, 2, 7, 3, 4, 5, 6] Output: [1, 7, 2, 6, 3, 5, 4]

In [ ]:
```
def test(nums):
    if len(nums) < 2:
        return nums
    result = []
    for i in range(len(nums)//2):
        result.append(min(nums))
        nums.remove(min(nums))
        result.append(max(nums))
        nums.remove(max(nums))
    if len(nums) > 0:
        result.append(nums[0])
    if len(result) < 2*len(nums):
        result.extend(nums[len(result) // 2 + 1:len(result) // 2 + 1 + len(nums) -
    return result

nums = [1, 3, 4, 5, 11]
print("Original list of numbers:")
print(nums)
print("Strange sort of list of said numbers:")
print(test(nums))
nums = [27, 3, 8, 5, 1, 31]
print("\nOriginal list of numbers:")
print(nums)
print("Strange sort of list of said numbers:")
print(test(nums))
nums = [1, 2, 7, 3, 4, 5, 6]
print("\nOriginal list of numbers:")
print(nums)
print("Strange sort of list of said numbers:")
print(test(nums))
```

```
Original list of numbers:
[1, 3, 4, 5, 11]
Strange sort of list of said numbers:
[1, 11, 3, 5, 4]

Original list of numbers:
[27, 3, 8, 5, 1, 31]
Strange sort of list of said numbers:
[1, 31, 3, 27, 5, 8]

Original list of numbers:
[1, 2, 7, 3, 4, 5, 6]
Strange sort of list of said numbers:
[1, 7, 2, 6, 3, 5, 4]
```

13.Write a python program to find four positive even integers whose sum is a given integer.

Input: n = 100

Output: [94, 2, 2, 2]

Input: n = 1000

Output: [994, 2, 2, 2]

Input: n = 10000

Output: [9994, 2, 2, 2]

Input: n = 1234567890

Output: [1234567884, 2, 2, 2]

In [ ]:
```
def test(n):
    for a in range(n, 0, -1):
        if not a % 2 == 0:
            continue
        for b in range(n - a, 0, -1):
            if not b % 2 == 0:
                continue
            for c in range(n - b - a, 0, -1):
                if not c % 2 == 0:
                    continue
                for d in range(n - b - c - a, 0, -1):
                    if not d % 2 == 0:
                        continue
                    if a + b + c + d == n:
                        return [a, b, c, d]

n = 100
print("Four positive even integers whose sum is",n)
print(test(n))
n = 1000
print("\nFour positive even integers whose sum is",n)
print(test(n))
n = 10000
print("\nFour positive even integers whose sum is",n)
```

```
print(test(n))
n = 1234567890
print("\nFour positive even integers whose sum is",n)
print(test(n))
```

```
Four positive even integers whose sum is 100
[94, 2, 2, 2]

Four positive even integers whose sum is 1000
[994, 2, 2, 2]

Four positive even integers whose sum is 10000
[9994, 2, 2, 2]

Four positive even integers whose sum is 1234567890
[1234567884, 2, 2, 2]
```

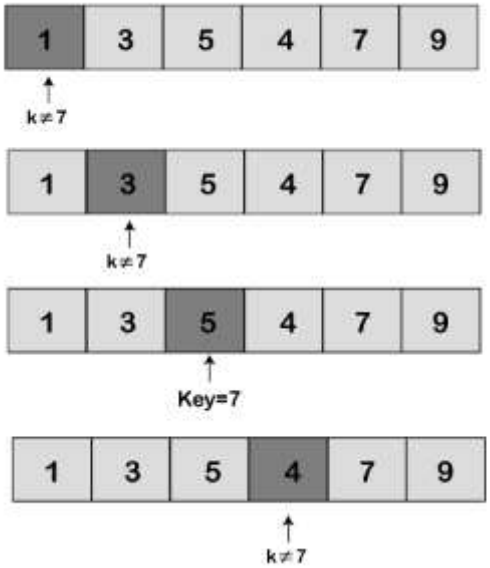
14.Write a python program to implement linear search.

Linear search is a method of finding elements within a list. It is also called a sequential search. It is the simplest searching algorithm because it searches the desired element in a sequential manner. It compares each and every element with the value that we are searching for. If both are matched, the element is found, and the algorithm returns the key's index position.

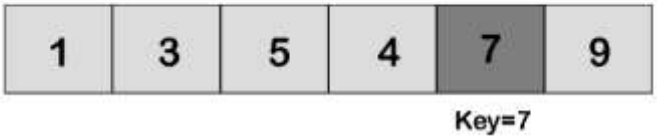
Step - 1: Start the search from the first element and Check key = 7 with each element of list x.

List to be Searched for

Step - 2: If element is found, return the index position of the key.



Step - 3: If element is not found, return element is not present.



Input: Enter the list of numbers: 5 4 3 2 1 10 11 2 The number to search for: 1

Output: 1 was found at index 4.

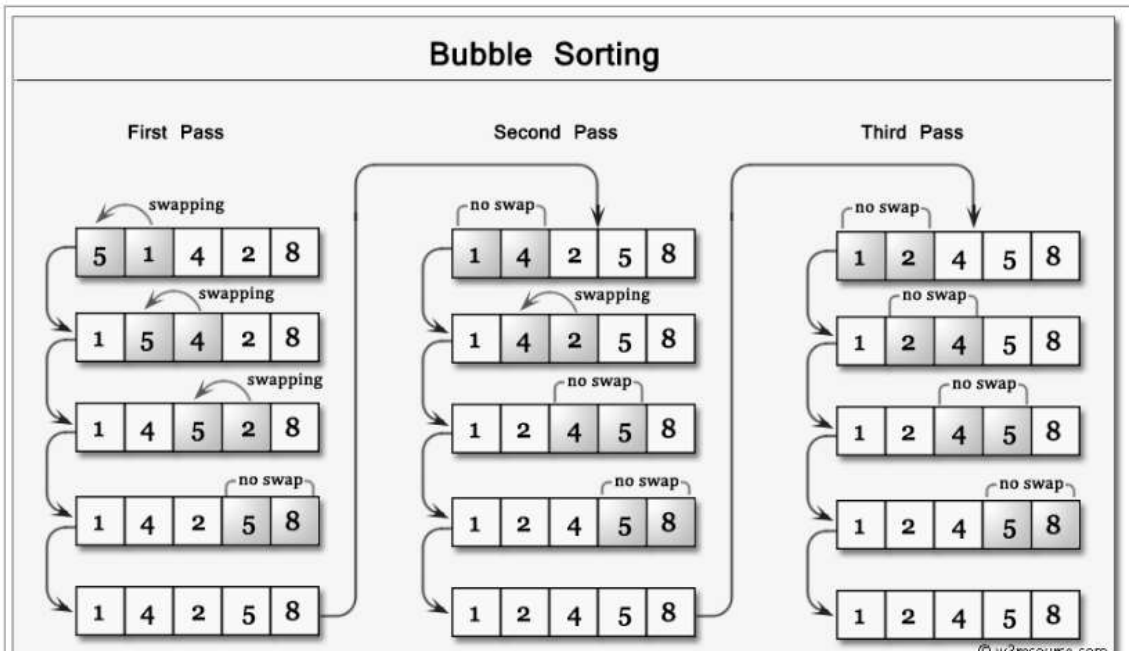
In [37]:
```
def linear_search(alist, key):
    """Return index of key in alist. Return -1 if key not present."""
    for i in range(len(alist)):
        if alist[i] == key:
            return i
    return -1


alist = input('Enter the list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = linear_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key, index))
```

```
Enter the list of numbers: 5 4 3 2 1 10 11 2
The number to search for: 1
1 was found at index 4.
```

15.Write a python program to sort a list of elements using the bubble sort algorithm.

Note : According to Wikipedia "Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort. It can be practical if the input is usually in sort order but may occasionally have some out-of-order elements nearly in position.

Step by step pictorial presentation :



Input:nlist = [14,46,43,27,57,41,45,21,70]

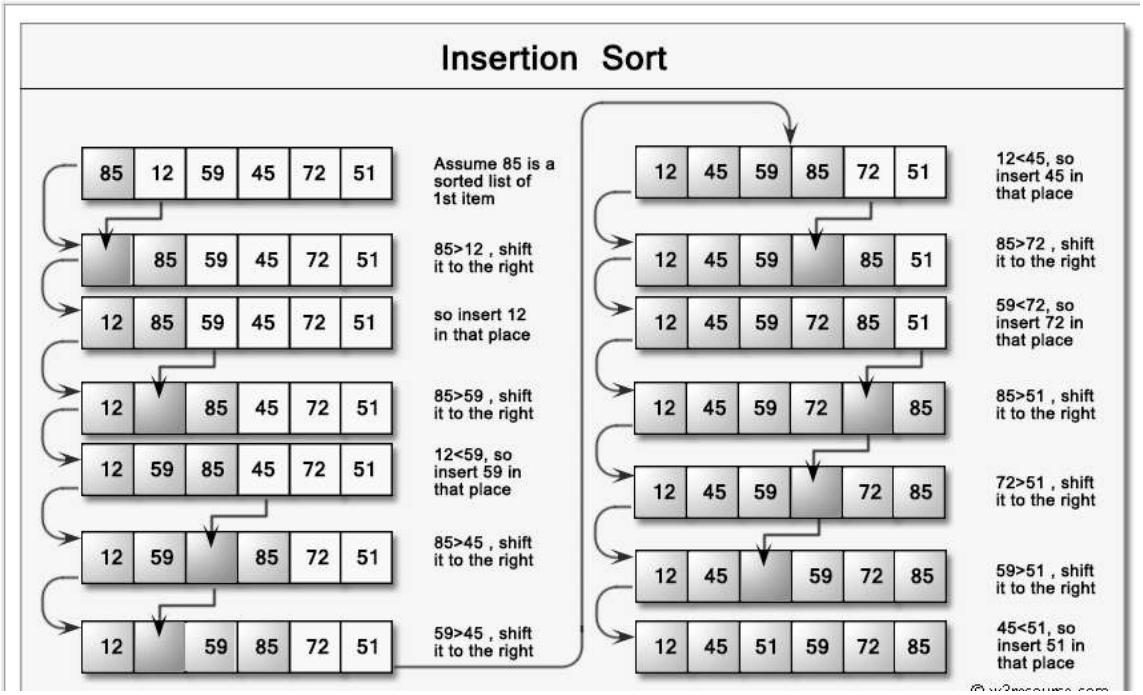
Output: [14, 21, 27, 41, 43, 45, 46, 57, 70]

In [ ]:
```
def bubbleSort(nlist):
    for passnum in range(len(nlist)-1,0,-1):
        for i in range(passnum):
            if nlist[i]>nlist[i+1]:
                temp = nlist[i]
                nlist[i] = nlist[i+1]
                nlist[i+1] = temp

nlist = [14,46,43,27,57,41,45,21,70]
bubbleSort(nlist)
print(nlist)
```

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

16.Write a python program to sort a list of elements using the insertion sort algorithm.

Note: According to Wikipedia "Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort."



Input: nlist = [14,46,43,27,57,41,45,21,70]

Output: [14, 21, 27, 41, 43, 45, 46, 57, 70]

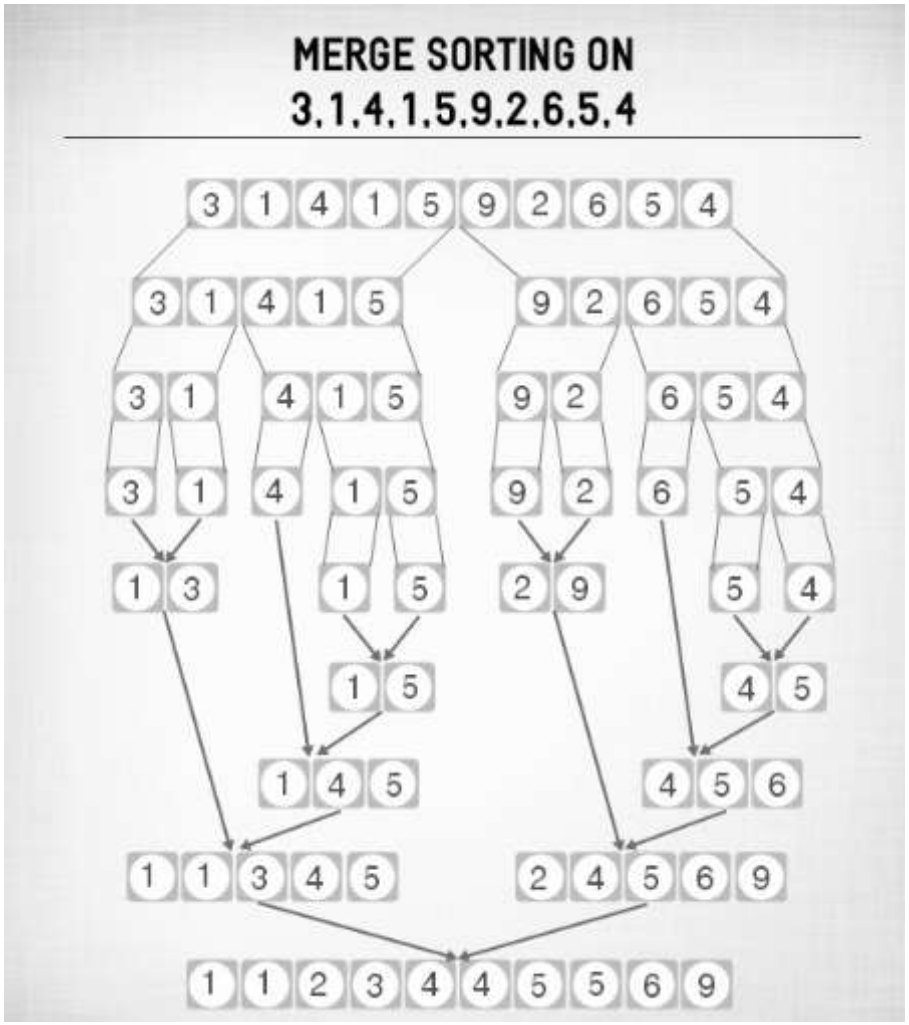
In [ ]:
```
def insertionSort(nlist):
   for index in range(1,len(nlist)):

     currentvalue = nlist[index]
     position = index

     while position>0 and nlist[position-1]>currentvalue:
         nlist[position]=nlist[position-1]
         position = position-1

     nlist[position]=currentvalue

nlist = [14,46,43,27,57,41,45,21,70]
insertionSort(nlist)
print(nlist)
```

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

17.Write a python program to sort a list of elements using the merge sort algorithm.

Note: According to Wikipedia "Merge sort (also commonly spelled mergesort) is an O (n log n) comparison-based sorting algorithm. Most implementations produce a stable sort, which

means that the implementation preserves the input order of equal elements in the sorted output." Algorithm: Conceptually, a merge sort works as follows : Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.



Input: nlist = [14,46,43,27,57,41,45,21,70]

Output:

Splitting [14, 46, 43, 27, 57, 41, 45, 21, 70] Splitting [14, 46, 43, 27] Splitting [14, 46] Splitting [14] Merging [14] Splitting [46] Merging [46] Merging [14, 46] Splitting [43, 27] Splitting [43] Merging [43] Splitting [27] Merging [27] Merging [27, 43] Merging [14, 27, 43, 46] Splitting [57, 41, 45, 21, 70] Splitting [57, 41] Splitting [57] Merging [57] Splitting [41] Merging [41] Merging [41, 57] Splitting [45, 21, 70] Splitting [45] Merging [45] Splitting [21, 70] Splitting [21] Merging [21] Splitting [70] Merging [70] Merging [21, 70] Merging [21, 45, 70] Merging [21, 41, 45, 57, 70] Merging [14, 21, 27, 41, 43, 45, 46, 57, 70] [14, 21, 27, 41, 43, 45, 46, 57, 70]

In [ ]:
```
def mergeSort(nlist):
    print("Splitting ",nlist)
    if len(nlist)>1:
        mid = len(nlist)//2
        lefthalf = nlist[:mid]
        righthalf = nlist[mid:]
```

```
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=j=k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                nlist[k]=lefthalf[i]
                i=i+1
            else:
                nlist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            nlist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            nlist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",nlist)

nlist = [14,46,43,27,57,41,45,21,70]
mergeSort(nlist)
print(nlist)
```

```
Splitting  [14, 46, 43, 27, 57, 41, 45, 21, 70]
Splitting  [14, 46, 43, 27]
Splitting  [14, 46]
Splitting  [14]
Merging  [14]
Splitting  [46]
Merging  [46]
Merging  [14, 46]
Splitting  [43, 27]
Splitting  [43]
Merging  [43]
Splitting  [27]
Merging  [27]
Merging  [27, 43]
Merging  [14, 27, 43, 46]
Splitting  [57, 41, 45, 21, 70]
Splitting  [57, 41]
Splitting  [57]
Merging  [57]
Splitting  [41]
Merging  [41]
Merging  [41, 57]
Splitting  [45, 21, 70]
Splitting  [45]
Merging  [45]
Splitting  [21, 70]
Splitting  [21]
Merging  [21]
Splitting  [70]
Merging  [70]
Merging  [21, 70]
Merging  [21, 45, 70]
Merging  [21, 41, 45, 57, 70]
Merging  [14, 21, 27, 41, 43, 45, 46, 57, 70]
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

18. Write a python program for backtracking example

Backtracking is a form of recursion. But it involves choosing only option out of any possibilities. We begin by choosing an option and backtrack from it, if we reach a state where we conclude that this specific option does not give the required solution. We repeat these steps by going across each available option until we get the desired solution. Below is an example of finding all possible order of arrangements of a given set of letters. When we choose a pair we apply backtracking to verify if that exact pair has already been created or not. If not already created, the pair is added to the answer list else it is ignored.

Write a python program for backtracking example

Input: permute(1, ["a","b","c"])

Output: ['a', 'b', 'c']

Input: permute(2, ["a","b","c"])

Output: ['aa', 'ab', 'ac', 'ba', 'bb', 'bc', 'ca', 'cb', 'cc']

In [ ]:
```
def permute(list, s):
   if list == 1:
      return s
   else:
      return [
         y + x
         for y in permute(1, s)
         for x in permute(list - 1, s)
      ]
print(permute(1, ["a","b","c"]))
print(permute(2, ["a","b","c"]))
```

```
['a', 'b', 'c']
['aa', 'ab', 'ac', 'ba', 'bb', 'bc', 'ca', 'cb', 'cc']
```

19. Write a python program for tower of hanoi game.

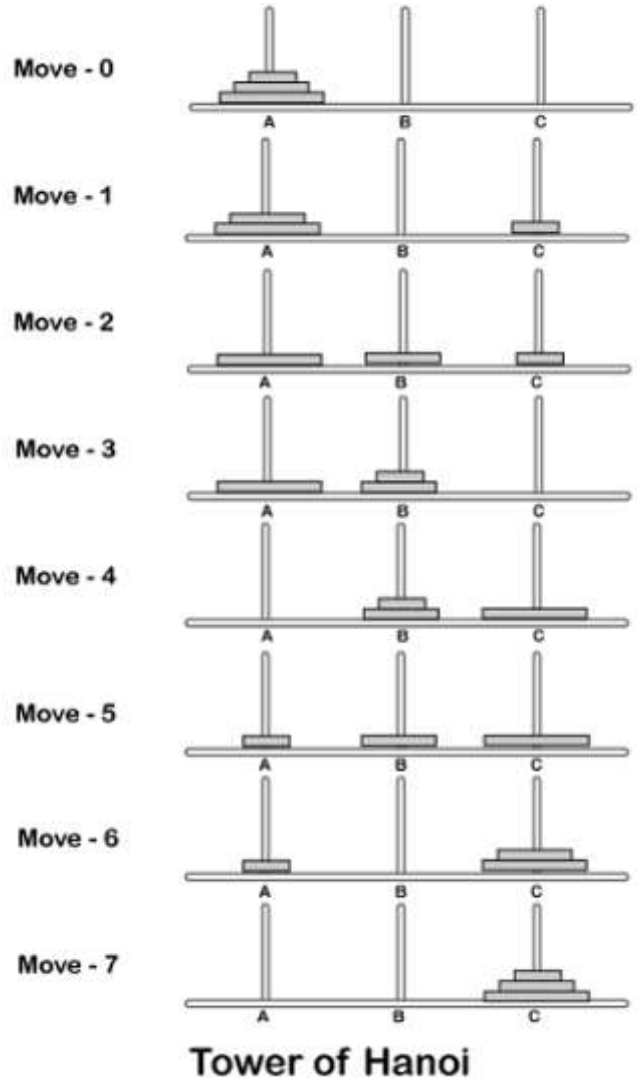
The program prompts the user for the number of disks n and the program prints the procedure to move n disks from peg A to peg C using peg B.

The rules of "Tower of Hanoi" are quite simple, but the solution is slightly hard. There are three rods. The disks are stacked in the descending order; the largest disk stacked at the bottom and the smallest one on top.

The task is to transfer the disks from one source rod to another target rod.

The following rule must not be violated

Only one disk can be moved at a time. The most upper disk from one of the rod can be stimulated in move. The smaller disk cannot be placed at the lower of the largest disk. The number of moves can be calculated as 2n - 1.



Tower of Hanoi

Input: Enter number of disks: 4

Output: Move disk 1 from peg A to peg B. Move disk 2 from peg A to peg C. Move disk 1 from peg B to peg C. Move disk 3 from peg A to peg B. Move disk 1 from peg C to peg A. Move disk 2 from peg C to peg B. Move disk 1 from peg A to peg B. Move disk 4 from peg A to peg C. Move disk 1 from peg B to peg C. Move disk 2 from peg B to peg A. Move disk 1 from peg C to peg A. Move disk 3 from peg B to peg C. Move disk 1 from peg A to peg B. Move disk 2 from peg A to peg C. Move disk 1 from peg B to peg C.

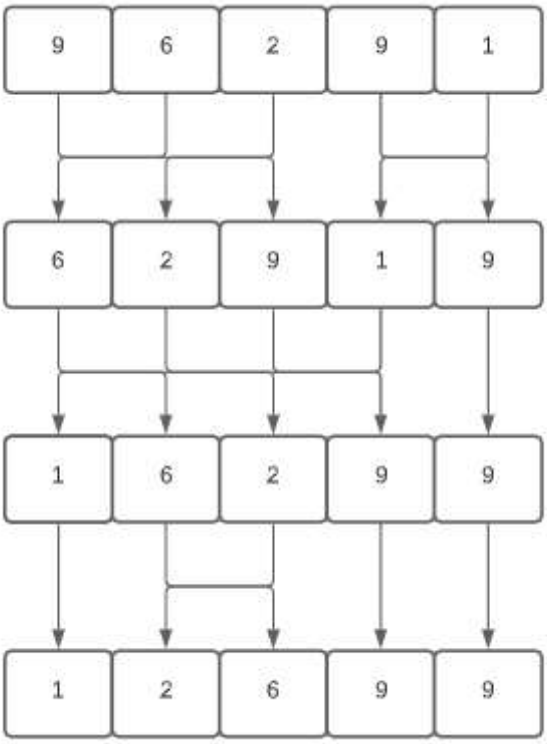
In [ ]:
```
def hanoi(disks, source, auxiliary, target):
    if disks == 1:
        print('Move disk 1 from peg {} to peg {}.'.format(source, target))
        return

    hanoi(disks - 1, source, target, auxiliary)
    print('Move disk {} from peg {} to peg {}.'.format(disks, source, target))
    hanoi(disks - 1, auxiliary, source, target)


disks = int(input('Enter number of disks: '))
hanoi(disks, 'A', 'B', 'C')
```

```
Enter number of disks: 4
Move disk 1 from peg A to peg B.
Move disk 2 from peg A to peg C.
Move disk 1 from peg B to peg C.
Move disk 3 from peg A to peg B.
Move disk 1 from peg C to peg A.
Move disk 2 from peg C to peg B.
Move disk 1 from peg A to peg B.
Move disk 4 from peg A to peg C.
Move disk 1 from peg B to peg C.
Move disk 2 from peg B to peg A.
Move disk 1 from peg C to peg A.
Move disk 3 from peg B to peg C.
Move disk 1 from peg A to peg B.
Move disk 2 from peg A to peg C.
Move disk 1 from peg B to peg C.
```

20.Write a python program to Implement CockTail Sort

Firstly array is traversed from left to right. During traversal, adjacent items are compared and based on the conditions, the values are swapped. By this, the largest number will be at the end of the array.

Now the array is traversed in the opposite direction and based on the conditions, elements are swapped. By this, the smallest number will be at the start.



Input: Enter the list of numbers: 3 18 5 2 10 0 7 4

Output: Sorted list: [0, 2, 3, 4, 5, 7, 10, 18]

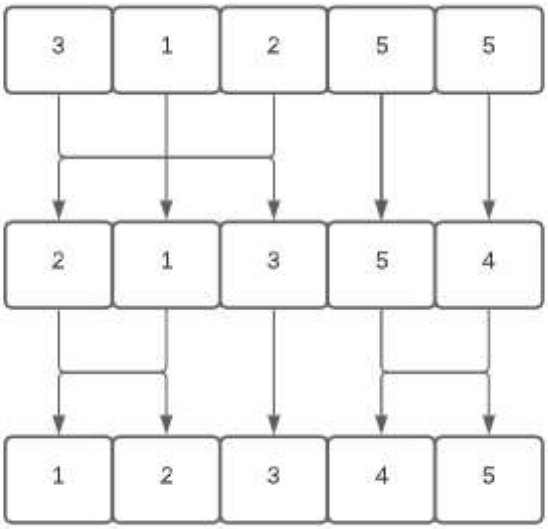
In [ ]:
```
def cocktail_shaker_sort(alist):
    def swap(i, j):
        alist[i], alist[j] = alist[j], alist[i]
```

```
    upper = len(alist) - 1
    lower = 0

    no_swap = False
    while (not no_swap and upper - lower > 1):
        no_swap = True
        for j in range(lower, upper):
            if alist[j + 1] < alist[j]:
                swap(j + 1, j)
                no_swap = False
        upper = upper - 1

        for j in range(upper, lower, -1):
            if alist[j - 1] > alist[j]:
                swap(j - 1, j)
                no_swap = False
        lower = lower + 1


alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
cocktail_shaker_sort(alist)
print('Sorted list: ', end='')
print(alist)
```

```
Enter the list of numbers: 3 18 5 2 10 0 7 4
Sorted list: [0, 2, 3, 4, 5, 7, 10, 18]
```

21. Write a python program to implement Comb sort.

The Comb Sort is a variant of the Bubble Sort. Like the Shell sort, the Comb Sort increases the gap used in comparisons and exchanges. Some implementations use the insertion sort once the gap is less than a certain amount. The basic idea is to eliminate turtles, or small values near the end of the list, since in a bubble sort these slow the sorting down tremendously. Rabbits, large values around the beginning of the list do not pose a problem in bubble sort. In bubble sort, when any two elements are compared, they always have a gap of 1. The basic idea of comb sort is that the gap can be much more than 1.



Input: Enter the list of numbers: 2 8 4 3 7 10 23 4 5

Output: Sorted list: [2, 3, 4, 4, 5, 7, 8, 10, 23]

In [ ]:
```
def comb_sort(alist):
    def swap(i, j):
        alist[i], alist[j] = alist[j], alist[i]

    gap = len(alist)
    shrink = 1.3

    no_swap = False
    while not no_swap:
        gap = int(gap/shrink)

        if gap < 1:
            gap = 1
            no_swap = True
        else:
            no_swap = False

        i = 0
        while i + gap < len(alist):
            if alist[i] > alist[i + gap]:
                swap(i, i + gap)
                no_swap = False
            i = i + 1


alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
comb_sort(alist)
print('Sorted list: ', end='')
print(alist)
```

```
Enter the list of numbers: 2 8 4 3 7 10 23 4 5
Sorted list: [2, 3, 4, 4, 5, 7, 8, 10, 23]
```

22. Write a python program to Implement Bucket Sort

According to Wikipedia "Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort. It can be practical if the input is usually in sort order but may occasionally have some out-of-order elements nearly in position."

Input: Enter the list of (nonnegative) numbers: 2 1 5 10 3 5 7

Output: Sorted list: [1, 2, 3, 5, 5, 7, 10]

In [38]:
```
def bucket_sort(alist):
    largest = max(alist)
    length = len(alist)
    size = largest/length

    buckets = [[] for _ in range(length)]
    for i in range(length):
        j = int(alist[i]/size)
        if j != length:
            buckets[j].append(alist[i])
        else:
            buckets[length - 1].append(alist[i])

    for i in range(length):
        insertion_sort(buckets[i])

    result = []
    for i in range(length):
        result = result + buckets[i]

    return result

def insertion_sort(alist):
    for i in range(1, len(alist)):
        temp = alist[i]
        j = i - 1
        while (j >= 0 and temp < alist[j]):
            alist[j + 1] = alist[j]
            j = j - 1
        alist[j + 1] = temp


alist = input('Enter the list of (nonnegative) numbers: ').split()
alist = [int(x) for x in alist]
sorted_list = bucket_sort(alist)
print('Sorted list: ', end='')
print(sorted_list)
```

```
Enter the list of (nonnegative) numbers: 2 1 5 10 3 5 7
Sorted list: [1, 2, 3, 5, 5, 7, 10]
```

23.Write a python program to Minimize Lateness using Greedy Algorithm

We are given n requests numbered 0 to n – 1. Each request i has a time that it takes to complete t(i) and a deadline d(i). If a request i starts at time s(i), then its finish time is f(i) = s(i) + t(i). The lateness of request i is L(i) = f(i) – d(i) if f(i) > d(i). The maximum value of L(i) over all i is called the maximum lateness. That is, maximum lateness = max(L(0), L(1), ..., L(n – 1)). The problem is to schedule all the requests such that the value of maximum lateness is minimized.

Problem Solution

1. The function minimize_lateness defined.
2. It takes two lists ttimes and dtimes as arguments.
3. ttimes[i] is the time taken to complete request i.

4. dtimes[i] is the deadline of request i.
5. The algorithm works by sorting the requests in order of earliest deadline.
6. This ordering is an optimal way to schedule the requests to minimize maximum lateness.
7. The minimum maximum lateness is then calculated.
8. The minimum maximum lateness and the optimal schedule ordering is returned.

Input: Enter number of requests: 5 Enter the time taken to complete the 5 request(s) in order: 2 4 1 3 5 Enter the deadlines of the 5 request(s) in order: 3 1 5 6 3

Output: The minimum maximum lateness: 9 The order in which the requests should be scheduled: [1, 0, 4, 2, 3]

In [ ]:
```
def minimize_lateness(ttimes, dtimes):
    """Return minimum max lateness and the schedule to obtain it.

    (min_lateness, schedule) is returned.

    Lateness of a request i is L(i) = finish time of i - deadline of if
    request i finishes after its deadline.
    The maximum lateness is the maximum value of L(i) over all i.
    min_lateness is the minimum value of the maximum lateness that can be
    achieved by optimally scheduling the requests.

    schedule is a list that contains the indexes of the requests ordered such
    that minimum maximum lateness is achieved.

    ttime[i] is the time taken to complete request i.
    dtime[i] is the deadline of request i.
    """
    # index = [0, 1, 2, ..., n - 1] for n requests
    index = list(range(len(dtimes)))
    # sort according to deadlines
    index.sort(key=lambda i: dtimes[i])

    min_lateness = 0
    start_time = 0
    for i in index:
        min_lateness = max(min_lateness,
                           (ttimes[i] + start_time) - dtimes[i])
        start_time += ttimes[i]

    return min_lateness, index


n = int(input('Enter number of requests: '))
ttimes = input('Enter the time taken to complete the {} request(s) in order: '
               .format(n)).split()
ttimes = [int(tt) for tt in ttimes]
dtimes = input('Enter the deadlines of the {} request(s) in order: '
               .format(n)).split()
dtimes = [int(dt) for dt in dtimes]

min_lateness, schedule = minimize_lateness(ttimes, dtimes)
print('The minimum maximum lateness:', min_lateness)
print('The order in which the requests should be scheduled:', schedule)
```

```
Enter number of requests: 5
Enter the time taken to complete the 5 request(s) in order: 2 4 1 3 5
Enter the deadlines of the 5 request(s) in order: 3 1 5 6 3
The minimum maximum lateness: 9
The order in which the requests should be scheduled: [1, 0, 4, 2, 3]
```

24.Write a python program to find the starting number, under ten thousand will create the longest chain.

From Wikipedia - The Collatz conjecture is a conjecture in mathematics that concerns a sequence defined as follows: start with any positive integer n. Then each term is obtained from the previous term as follows: if the previous term is even, the next term is one half the previous term. If the previous term is odd, the next term is 3 times the previous term plus 1. The conjecture is that no matter what value of n, the sequence will always reach 1.

Output: 6171

In [39]:
```
def create_sequence_num(n):
    terms = 1
    while n > 1:
        if n % 2 == 0:
            n = n / 2
        else:
            n = 3 * n + 1
        terms += 1
    return terms


def Collatz_starting_num():
    temp = 0
    i = 1
    while i < 10000:
        s = create_sequence_num(i)
        if s > temp:
            temp = s
            value = i
        i += 1

    return value

print(Collatz_starting_num())
```

```
6171
```

25. Write a python program to find three numbers from an array such that the sum of three numbers equal to zero.

Input: nums1=[-1,0,1,2,-1,-4]

Output: [[-1, -1, 2], [-1, 0, 1]]

Input: nums2 = [-25,-10,-7,-3,2,4,8,10

Output: [[-10, 2, 8], [-7, -3, 10]]

In [40]:
```
# return a list of lists of length 3
def three_Sum(num):
    if len(num)<3: return []
    num.sort()
    result=[]
    for i in range(len(num)-2):
        left=i+1
        right=len(num)-1
        if i!=0 and num[i]==num[i-1]:continue
        while left<right:
            if num[left]+num[right]==-num[i]:
                result.append([num[i],num[left],num[right]])
                left=left+1
                right=right-1
                while num[left]==num[left-1] and left<right:left=left+1
                while num[right]==num[right+1] and left<right: right=right-1
            elif num[left]+num[right]<-num[i]:
                left=left+1
            else:
                right=right-1
    return result

nums1=[-1,0,1,2,-1,-4]
nums2 = [-25,-10,-7,-3,2,4,8,10]
print(three_Sum(nums1))
print(three_Sum(nums2))
```

```
[[-1, -1, 2], [-1, 0, 1]]
[[-10, 2, 8], [-7, -3, 10]]
```

26. Write a python program to check a sequence of numbers is a geometric progression or not.

In mathematics, a geometric progression or geometric sequence is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the common ratio. For example, the sequence 2, 6, 18, 54, ... is a geometric progression with common ratio 3. Similarly, 10, 5, 2.5, 1.25, ... is a geometric sequence with common ratio 1/2.

Input: [2, 6, 18, 54]

Output: True

Input: [5, 8, 9, 11]

Output: False

In [41]:
```
def is_geometric(li):
    if len(li) <= 1:
        return True
    # Calculate ratio
    ratio = li[1]/float(li[0])
    # Check the ratio of the remaining
    for i in range(1, len(li)):
        if li[i]/float(li[i-1]) != ratio:
```

```
            return False
    return True

print(is_geometric([2, 6, 18, 54]))

print(is_geometric([10, 5, 2.5, 1.25]))

print(is_geometric([5, 8, 9, 11]))
```

```
True
True
False
```

In [ ]: