1. Find kth even integers.

In [15]:

```
start=int(input("Enter the starting values of range: "))
end=int(input("Enter the endign values of range:"))
list1=list(range(start,end+1))
print("Your list is:",list1)
even=(list(filter(lambda x:(x%2==0),list1)))
print("Even number in the list:",even)
```

Enter the starting values of range: 1
Enter the endign values of range:20
Your list is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Even number in the list: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

2.Sum the series 1+2+3+…+n.

In [23]:

```
n=int(input("Enter the range of number:"))
sum=0
for i in range(1,n+1):
   sum+=i
print("The sum of the series = ",sum)
```

Enter the range of number:11
The sum of the series = 66

3.Find kth power of an integer number n.

In [7]:

```
number = int(input(" Please Enter any Positive Integer : "))
exponent = int(input(" Please Enter Exponent Value : "))

power = 1
i = 1

while(i <= exponent):
   power = power * number
   i = i + 1

print("The Result of {0} Power {1} = {2}".format(number, exponent, power))
```

Please Enter any Positive Integer : 2
 Please Enter Exponent Value : 2
The Result of 2 Power 2 = 4

4.Find Minimum and maximum of the given array.

In [8]:

```
def maxminposition(A, n):
  # inbuilt function to find the position of minimum
  minposition = A.index(min(A))
  # inbuilt function to find the position of maximum
  maxposition = A.index(max(A))
  print ("The maximum is at position::", maxposition + 1)
  print ("The minimum is at position::", minposition + 1)

A=list()
n=int(input("Enter the size of the List ::"))
print("Enter the Element ::")
for i in range(int(n)):
  k=int(input(""))
  A.append(k)
maxminposition(A,n)
```

```
Enter the size of the List ::5
Enter the Element ::
1
2
3
4
5
The maximum is at position:: 5
The minimum is at position:: 1
```

5.Display the Fibonacci series upto given integer number.

In [9]:

```
nterms = int(input("How many terms? "))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
  print("Please enter a positive integer")
elif nterms == 1:
  print("Fibonacci sequence upto",nterms,":")
  print(n1)
else:
  print("Fibonacci sequence:")
  while count < nterms:
      print(n1)
      nth = n1 + n2
      # update values
      n1 = n2
      n2 = nth
```

```
        count += 1
```

How many terms? 9
Fibonacci sequence:
0
1
1
2
3
5
8
13
21

6.Search an element into given array using following methods:

- (Explain method also)
  Sequential / Linear Search
- This is the simplest searching technique.
- It sequentially checks each element of the list for the target searching value until a match is found or until all the elements have been searched.
- This searching technique can be performed on both type of list, either the list is sorted or unsorted.
  Implement Linear search following the below steps:
- Start the search from the first element and Check key = n with each element of list x.
- If element is found, return the index position of the key.
- If element is not found, return element is not present.
  Binary Search
- Binary Search is applied on the sorted array or list of large size. It's time complexity of O(log n) makes it very fast as compared to other sorting algorithms. To use binary search on a collection, the collection must first be sorted.
  Implement binary search following the below steps:
- Start with the middle element of the given list:
- If the target value is equal to the middle element of the array, then return the index of the middle element.
- Otherwise, compare the middle element with the target value,
- If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
- If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
- If a match is found, return the index of the element matched.
- Otherwise, return -1.

Linear Search

In [21]:

```
pos = -1
def search(list,n):
   i = 0
   while i<len(list):
     if list[i] == n:
       globals()['pos'] = i
       return True
     i = i + 1
   return False
list = [5,8,4,6,9,2]
n = 6
if search(list,n):
  print("Found at ",pos+1)
else:
  print("Not Found")
```

Found at 4

Binary Search

In [18]:

```
def binary_search(list1, n):
   low = 0
   high = len(list1) - 1
   mid = 0
    while low <= high:
       # for get integer result
       mid = (high + low) // 2
        # Check if n is present at mid
       if list1[mid] < n:
           low = mid + 1
        # If n is greater, compare to the right of mid
       elif list1[mid] > n:
           high = mid - 1
        # If n is smaller, compared to the left of mid
       else:
           return mid
            # element was not present in the list, return -1
   return -1

# Initial list1
list1 = [12, 24, 32, 39, 45, 50, 54]
n = 45
 # Function call
result = binary_search(list1, n)
 if result != -1:
```

```
    print("Element is present at index", str(result))
else:
    print("Element is not present in list1")
```

Element is present at index 4

7. Sort the elements of an array using following methods:

- (Explain method also)
- Selection Sort
- Merge Sort
- Quick Sort

Selection sort

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
- The subarray which is already sorted.
- Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

In [25]:

```
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j
    A[i], A[min_idx] = A[min_idx], A[i]
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i]),
```

Sorted array
11
12
22
25
64

MergeSort

- Merge Sort is a Divide and Conquer algorithm.
- It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
- The merge() function is used for merging two halves.
- The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

In [29]:

```
# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r- m
    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

# Copy data to temp arrays L[] and R[]
    for i in range(0 , n1):
        L[i] = arr[l + i]

    for j in range(0 , n2):
        R[j] = arr[m + 1 + j]

# Merge the temp arrays back into arr[l..r]
    i = 0 # Initial index of first subarray
    j = 0 # Initial index of second subarray
    k = l # Initial index of merged subarray

    while i < n1 and j < n2 :
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

# Copy the remaining elements of L[], if there
# are any
    while i < n1:
        arr[k] = L[i]
```

```
        i += 1
        k += 1

# Copy the remaining elements of R[], if there
# are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr,l,r):
    if l < r:

        # Same as (l+r)//2, but avoids overflow for
        # large l and h
        m = (l+(r-1))//2

# Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)


# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print ("Given array is")
for i in range(n):
    print ("%d" %arr[i]),

mergeSort(arr,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
    print ("%d" %arr[i])
```

```
Given array is
12
11
13
5
6
7
```

Sorted array is
5
6
7
11
12
13

Quick Sort

- Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.
- Always pick first element as pivot.
- Always pick last element as pivot (implemented below)
- Pick a random element as pivot.
- Pick median as pivot.

In [30]:

```
def partition(arr, low, high):
    i = (low-1)# index of smaller element
    pivot = arr[high] # pivot

    for j in range(low, high):

# If current element is smaller than or
# equal to pivot
        if arr[j] <= pivot:

# increment index of smaller element
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort


def quickSort(arr, low, high):
```

```
    if len(arr) == 1:
        return arr
    if low < high:

# pi is partitioning index, arr[p] is now
# at right place
        pi = partition(arr, low, high)

# Separately sort elements before
# partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)


# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted array is:")
for i in range(n):
    print("%d" % arr[i]),
```

```
Sorted array is:
1
5
7
8
9
10
```

In [ ]:

8. a

Q.8) a) $T(n) = T(n-1) + c$ if $n > 0$ otherwise 1 for $n = 0$

$T(n) = T(n-1) + 1$

Substitute $T(n-1) = T(n-1) + 1$

$T(n) = [T(n-2) + 1] + 1$

$T(n) = T(n-2) + 2$

$T(n) = [T(n-3) + 1] + 2$

$T(n) = T(n-3) + 3$

Continue for k times

$T(n) = T(n-k) + k$

Assume $n - k = 0$

$n = k$

$T(n) = T(n-n) + n$

$T(n) = T(0) + n$

$T(n) = 1 + n$

we can call it as $Q(n)$

$\therefore T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

```
void Test (int n)
{
 if (n>0)
 {
   Print ('%d', n);
   Test(n-1);
 }
}
```

Test (3)

-3 Test(2)

-2 Test(1)

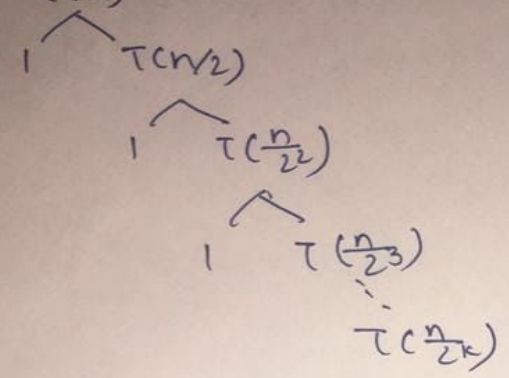
-1 Test(0)

$f(n) = n + 1$ calls.

$\therefore Q(n)$

b) $T(n) = T(n/2) + c$, if $(n > 1)$ otherwise 1 for $n = 1$

$T(n) = T(n/2) + c$



$T(n) = T(n/2) + 1$

$T(n) = [T(\frac{n}{2^2}) + 1] + 1$

$T(n) = T(\frac{n}{2^2}) + 2$

$T(n) = T(\frac{n}{2^3}) + 3$

$T(n) = T(\frac{n}{2^k}) + k$

k steps

$\because \frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2^n$ $[a^b = c \quad b = \log_a^c]$

$O(\log_2^n)$

Assume $\frac{n}{2^k} = 1, n = 2^k$

$k = \log n$

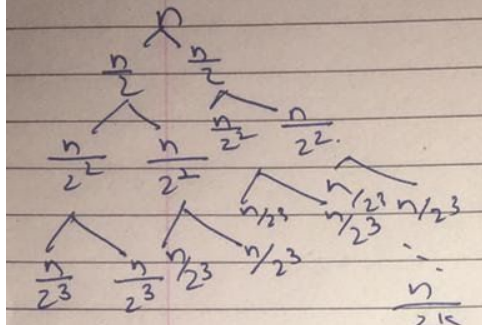
$T(n) = T(1) + k$

$T(n) = 1 + \log n$

$O(\log n)$

```
Void Test(int n)
{
  if (n>1)
  {
  Print("%d ", n);
     Test (n/2)
   }
  }
```

c) $T(n) = 2T(n/2) + cn$, if $n>1$ otherwise a for $n=1$



The process will continue k times.

we, assume $\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log n$

Therefore this is $n \log n$

The time for it is $O(n \log n)$

```
void Test(int n){
  if (n>1){
    for(i=1, i<n, i++){
      Stmt;
    } test (n/2);
      Test (n/2);
  }
}
```

$T(n) = 2T(n/2) + n$

$= 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n.$

$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + n + n.$

$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + 2^n$

$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n.$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$

Assume $T\left(\frac{n}{2^k}\right) = T(1)$

$\frac{n}{2^k} = 1 \quad n = 2^k$

$k = \log n$

$T(n) = 2^k T(1) + kn$

$T(n) = n \times 1 + n \log n$

$O(n \log n)$