Write a program to implement binary search and find the time complexity of the program

```
import time
import numpy as np
import matplotlib.pyplot as plt
def binary_search_basic(arr, target):
    low,high =0,len(arr)
    while low < high:
        mid =(low+high)
        if target < arr[mid]:
            high=mid
        elif target>arr[mid]:
            low=mid+1
        else:
            return mid
    return -1
elements=np.array([i*1000 for i in range(1,40)])
plt.xlabel('list length')
plt.ylabel('Time complexity')
times=list()
for i in range(1, 40):
    start=time.time()
    a=np.random.randint(1000, size=i*1000)
    end=time.time()
    times.append(end-start)
    print("time taken for binary search in",i*1000,"elementsis",end-start)
plt.plot(elements,times,label="binary search")
plt.grid()
```

```
plt.legend()

plt.show()
```

output

```
time taken for binary search in 1000 elementsis 0.0 s

time taken for binary search in 2000 elementsis 0.0 s

time taken for binary search in 3000 elementsis 0.0 s

time taken for binary search in 4000 elementsis 0.0 s

time taken for binary search in 5000 elementsis 0.0 s

time taken for binary search in 6000 elementsis 0.0 s

time taken for binary search in 7000 elementsis 0.0 s

time taken for binary search in 8000 elementsis 0.0 s

time taken for binary search in 9000 elementsis 0.0 s

time taken for binary search in 10000 elementsis 0.0 s

time taken for binary search in 11000 elementsis 0.0 s

time taken for binary search in 12000 elementsis 0.0 s

time taken for binary search in 13000 elementsis 0.0 s

time taken for binary search in 14000 elementsis 0.0 s

time taken for binary search in 15000 elementsis 0.0 s

time taken for binary search in 16000 elementsis 0.0 s

time taken for binary search in 17000 elementsis 0.0 s

time taken for binary search in 18000 elementsis 0.0 s

time taken for binary search in 19000 elementsis 0.0 s

time taken for binary search in 20000 elementsis 0.0 s

time taken for binary search in 21000 elementsis 0.0 s

time taken for binary search in 22000 elementsis 0.0 s

time taken for binary search in 23000 elementsis 0.0 s
```

time taken for binary search in 24000 elementsis 0.0 s

time taken for binary search in 25000 elementsis 0.0 s

time taken for binary search in 26000 elementsis 0.0 s

time taken for binary search in 27000 elementsis 0.0 s

time taken for binary search in 28000 elementsis 0.0 s

time taken for binary search in 29000 elementsis 0.0 s

time taken for binary search in 30000 elementsis 0.0 s

time taken for binary search in 31000 elementsis 0.0 s

time taken for binary search in 32000 elementsis 0.0 s

time taken for binary search in 33000 elementsis 0.010033845901489258 s

time taken for binary search in 34000 elementsis 0.0 s

time taken for binary search in 35000 elementsis 0.0 s

time taken for binary search in 36000 elementsis 0.0 s

time taken for binary search in 37000 elementsis 0.0 s

time taken for binary search in 38000 elementsis 0.0 s

time taken for binary search in 39000 elementsis 0.0 s

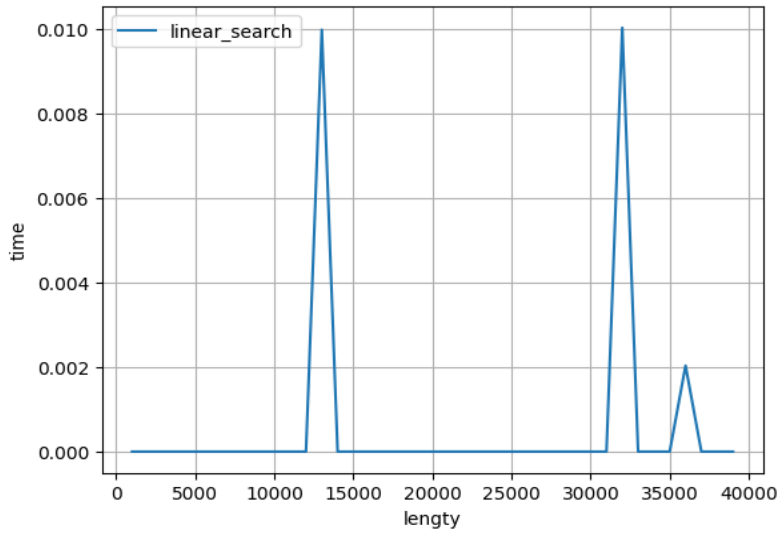
Write a program for linear search compute time and space complexity plot graph using asymptomatic notations

```
import time
import numpy as np
import matplotlib.pyplot as plt
def linear_search(A,x):
    for i in range(0,len(A)):
        if A[i]==x:
            print('search is success at',i)
            return
    print('search is not success')
elements = np.array([i*1000 for i in range(1, 40)])
plt.xlabel('lengty')
plt.ylabel('time')
times = list()
for i in range (1,40):
        start=time.time()
        a=np.random.randint(1000, size=i*1000)
        linear_search(a,1)
        end=time.time()
        times.append(end-start)
        print('time taken', i*1000, 'element',end-start)
plt.plot(elements, times, label="linear_search")
plt.grid()
plt.legend()
plt.show()
```

Output

```
search is success at 121
time taken 1000 element 0.0
search is success at 285
time taken 2000 element 0.0
search is success at 541
time taken 3000 element 0.0
search is success at 771
time taken 4000 element 0.0
search is success at 214
time taken 5000 element 0.0
search is success at 1364
time taken 6000 element 0.0
search is success at 2406
time taken 7000 element 0.0
search is success at 972
time taken 8000 element 0.0
search is success at 3323
time taken 9000 element 0.0
search is success at 620
time taken 10000 element 0.0
search is success at 112
time taken 11000 element 0.0
search is success at 1759
time taken 12000 element 0.0
search is success at 1634
time taken 13000 element 0.009980916976928711
search is success at 71
time taken 14000 element 0.0
search is success at 25
time taken 15000 element 0.0
search is success at 133
time taken 16000 element 0.0
search is success at 1124
time taken 17000 element 0.0
search is success at 1873
time taken 18000 element 0.0
search is success at 894
time taken 19000 element 0.0
search is success at 461
time taken 20000 element 0.0
search is success at 326
time taken 21000 element 0.0
search is success at 1993
time taken 22000 element 0.0
search is success at 2854
time taken 23000 element 0.0
```

```
search is success at 128
time taken 24000 element 0.0
search is success at 102
time taken 25000 element 0.0
search is success at 4926
time taken 26000 element 0.0
search is success at 4284
time taken 27000 element 0.0
search is success at 254
time taken 28000 element 0.0
search is success at 113
time taken 29000 element 0.0
search is success at 4344
time taken 30000 element 0.0
search is success at 2224
time taken 31000 element 0.0
search is success at 2283
time taken 32000 element 0.010028839111328125
search is success at 477
time taken 33000 element 0.0
search is success at 2429
time taken 34000 element 0.0
search is success at 1538
time taken 35000 element 0.0
search is success at 2781
time taken 36000 element 0.0020341873168945312
search is success at 60
time taken 37000 element 0.0
search is success at 646
time taken 38000 element 0.0
search is success at 442
time taken 39000 element 0.0
```



Write a program for bubble sort and find time complexity of the program

```
import time
start = time.time()

def bubleSort(a):
    b = True
    while b:
        b = False
        for i in range(len(a)-1):
            if a[i] > a[i+1]:
                a[i], a[i+1] = a[i+1], a[i]
                print("Swapped: {} with {}".format(a[i+1], a[i]))
                b = True
    return a
a = [70,30,20,50,60,10,40]
print("\nUnsorted array is:",a)
bubleSort(a)
print("\nSorted array is:",a)
end = time.time()
print("The time complexity of program is :", end-start)
```

Output

```
Unsorted array is: [70, 30, 20, 50, 60, 10, 40]
Swapped: 70 with 30
Swapped: 70 with 20
Swapped: 70 with 50
Swapped: 70 with 60
Swapped: 70 with 10
Swapped: 70 with 40
Swapped: 30 with 20
Swapped: 60 with 10
Swapped: 60 with 40
Swapped: 50 with 10
Swapped: 50 with 40
Swapped: 30 with 10
Swapped: 20 with 10

Sorted array is: [10, 20, 30, 40, 50, 60, 70]
The time complexity of program is : 0.0
```

Write a program to implement merge sort

```
def merge_sort(unsorted_list):
  if len(unsorted_list) <= 1:
    return unsorted_list
# Find the middle point and devide it
  middle = len(unsorted_list) // 2
  left_list = unsorted_list[:middle]
  right_list = unsorted_list[middle:]

  left_list = merge_sort(left_list)
  right_list = merge_sort(right_list)
  return list(merge(left_list, right_list))

# Merge the sorted halves
def merge(left_half,right_half):
  res = []
  while len(left_half) != 0 and len(right_half) != 0:
    if left_half[0] < right_half[0]:
      res.append(left_half[0])
      left_half.remove(left_half[0])
    else:
      res.append(right_half[0])
      right_half.remove(right_half[0])
  if len(left_half) == 0:
    res = res + right_half
  else:
    res = res + left_half
  return res
unsorted_list = [64, 34, 25, 12, 22, 11, 90]
print(merge_sort(unsorted_list))
```

Output

```
[11, 12, 22, 25, 34, 64, 90]
```

Write a program to implement a selection sort

```
def selectionSort(array, size):

    for ind in range(size):
        min_index = ind

        for j in range(ind + 1, size):
            # select the minimum element in every iteration
            if array[j] < array[min_index]:
                min_index = j
         # swapping the elements to sort the array
        (array[ind], array[min_index]) = (array[min_index], array[ind])

arr = [-2, 45, 0, 11, -9,88,-97,-202,747]
size = len(arr)
selectionSort(arr, size)
print('The array after sorting in Ascending Order by selection sort is:')
print(arr)
```

Output

```
The array after sorting in Ascending Order by selection sort is:
[-202, -97, -9, -2, 0, 11, 45, 88, 747]
```

Write a program to implement single linked list

```
import time

start=time.time()


import collections

single_linked_list=collections.deque()

single_linked_list.append('30')

single_linked_list.append('23')

single_linked_list.append('56')

single_linked_list.append('68')


print(" elements of single linked list is :\n",single_linked_list)


# insert new element at  beginning

single_linked_list.insert(0,'65')

print(" after adding an elements at begining:\n",single_linked_list)


# insert new element at middle

single_linked_list.insert(2,'75')

print(" after adding an elements at midle :\n",single_linked_list)


# insert new element at  end

single_linked_list.insert(7,'15')

print(" after adding an elements at end :\n",single_linked_list)


# delete an element at beginning

single_linked_list.remove('65')
```

```
print(" after removing  an elements at begining :\n",single_linked_list)


# delete an element at beginning

single_linked_list.remove('75')

print(" after removing  an elements at middle:\n",single_linked_list)


# delete an element at beginning

single_linked_list.remove('15')

print(" after removing  an elements at end:\n",single_linked_list)
```

Output

```
elements of single linked list is :
 deque(['30', '23', '56', '68'])
 after adding an elements at begining:
 deque(['65', '30', '23', '56', '68'])
 after adding an elements at midle :
 deque(['65', '30', '75', '23', '56', '68'])
 after adding an elements at end :
 deque(['65', '30', '75', '23', '56', '68', '15'])
 after removing  an elements at begining :
 deque(['30', '75', '23', '56', '68', '15'])
 after removing  an elements at middle:
 deque(['30', '23', '56', '68', '15'])
 after removing  an elements at end:
 deque(['30', '23', '56', '68'])
```

Write a program to implement double linked list

```
import time

start=time.time()


import collections

double_linked_list=collections.deque()

double_linked_list.append('10')

double_linked_list.append('20')

double_linked_list.append('30')

double_linked_list.append('40')


print(" elements of double linked list is :\n",double_linked_list)


# insert new element at  beginning

double_linked_list.insert(0,'5')

print(" after adding an elements at begining:\n",double_linked_list)


# insert new element at middle

double_linked_list.insert(2,'15')

print(" after adding an elements at midle :\n",double_linked_list)


# insert new element at  end

double_linked_list.insert(7,'25')

print(" after adding an elements at end :\n",double_linked_list)


# delete an element at beginning

double_linked_list.remove('5')
```

```
print(" after removing  an elements at begining :\n",double_linked_list)


# delete an element at beginning

double_linked_list.remove('15')

print(" after removing  an elements at middle:\n",double_linked_list)


# delete an element at beginning

double_linked_list.remove('25')

print(" after removing:\n",double_linked_list)
```

Output

```
elements of double linked list is :
 deque(['10', '20', '30', '40'])
 after adding an elements at begining:
 deque(['5', '10', '20', '30', '40'])
 after adding an elements at midle :
 deque(['5', '10', '15', '20', '30', '40'])
 after adding an elements at end :
 deque(['5', '10', '15', '20', '30', '40', '25'])
 after removing  an elements at begining :
 deque(['10', '15', '20', '30', '40', '25'])
 after removing  an elements at middle:
 deque(['10', '20', '30', '40', '25'])
 after removing:
 deque(['10', '20', '30', '40'])
```

Write a program to implement circular_linked_list

```
import time
start=time.time()

import collections
circular_linked_list=collections.deque()
circular_linked_list.append('1')
circular_linked_list.append('2')
circular_linked_list.append('3')
circular_linked_list.append('4')

print(" elements of circular linked list is :\n",circular_linked_list)

# insert new element at  beginning
circular_linked_list.insert(0,'5')
print(" after adding an elements at begining:\n",circular_linked_list)

# insert new element at middle
circular_linked_list.insert(2,'15')
print(" after adding an elements at midle :\n",circular_linked_list)

# insert new element at  end
circular_linked_list.insert(7,'25')
print(" after adding an elements at end :\n",circular_linked_list)

# delete an element at beginning
circular_linked_list.remove('5')
print(" after removing  an elements at begining :\n",circular_linked_list)

# delete an element at beginning
circular_linked_list.remove('15')
print(" after removing  an elements at middle:\n",circular_linked_list)

# delete an element at beginning
circular_linked_list.remove('25')
print(" after removing  an elements at end:\n",circular_linked_list)
```

Output

```
elements of circular linked list is :
 deque(['1', '2', '3', '4'])
 after adding an elements at begining:
 deque(['5', '1', '2', '3', '4'])
 after adding an elements at midle :
 deque(['5', '1', '15', '2', '3', '4'])
 after adding an elements at end :
 deque(['5', '1', '15', '2', '3', '4', '25'])
 after removing  an elements at begining :
 deque(['1', '15', '2', '3', '4', '25'])
 after removing  an elements at middle:
 deque(['1', '2', '3', '4', '25'])
 after removing  an elements at end:
 deque(['1', '2', '3', '4'])
```

Write a program to implement circular_double_linked_list

```
import time

start=time.time()


import collections

circular_double_linked_list=collections.deque()

circular_double_linked_list.append('100')

circular_double_linked_list.append('200')

circular_double_linked_list.append('300')

circular_double_linked_list.append('400')


print(" elements of circular_double linked list is :\n",circular_double_linked_list)


# insert new element at  beginning

circular_double_linked_list.insert(0,'105')

print(" after adding an elements at begining:\n",circular_double_linked_list)


# insert new element at middle

circular_double_linked_list.insert(2,'205')

print(" after adding an elements at midle :\n",circular_double_linked_list)


# insert new element at  end

circular_double_linked_list.insert(7,'305')

print(" after adding an elements at end :\n",circular_double_linked_list)


# delete an element at beginning

circular_double_linked_list.remove('105')
```

```
print(" after removing  an elements at begining :\n",circular_double_linked_list)


# delete an element at beginning

circular_double_linked_list.remove('205')

print(" after removing  an elements at middle:\n",circular_double_linked_list)


# delete an element at beginning

circular_double_linked_list.remove('305')

print(" after removing  an elements at end:\n",circular_double_linked_list)
```

Output

```
elements of circular_double linked list is :
 deque(['100', '200', '300', '400'])
 after adding an elements at begining:
 deque(['105', '100', '200', '300', '400'])
 after adding an elements at midle :
 deque(['105', '100', '205', '200', '300', '400'])
 after adding an elements at end :
 deque(['105', '100', '205', '200', '300', '400', '305'])
 after removing  an elements at begining :
 deque(['100', '205', '200', '300', '400', '305'])
 after removing  an elements at middle:
 deque(['100', '200', '300', '400', '305'])
 after removing  an elements at end:
 deque(['100', '200', '300', '400'])
```

program to demonstrate stack implementation using a list

```
stack = []

# append() function to push  element in the stack

stack.append('a')

stack.append('b')

stack.append('c')


print('Initial stack')

print(stack)


# pop() function to pop  element from stack in

# LIFO order

print('\nElements popped from stack:')

print(stack.pop())

print(stack.pop())

print(stack.pop())


print('\nStack after elements are popped:')

print(stack)


# uncommenting print(stack.pop())

# the stack is now empty
```

Output

```
Initial stack
['a', 'b', 'c']

Elements popped from stack:
c
b
a

Stack after elements are popped:
[]
```

Write a program to implement Fibonacci series using recursive method

```
def fib(n):


    if n==0 or n==1:
        return n
    else:
        return fib(n-1)+fib(n-2)
n=int(input("enter number"))
print(" fibonocci series is :")
for i in range(0,n):
    print(fib(i), end=" ")
```

output

```
enter number 15
 fibonocci series :
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Write a program to implement Factorial of the given number using recursive method

```
# Python 3 program to find  factorial of given number
def factorial(n):

    # Checking the number is 1 or 0 then
    # return 1
    # other wise return factorial
    if (n==1 or n==0):

        return 1

    else:

        return (n * factorial(n - 1))


num =int(input('enter the number to find factorial'))
print("number : ",num)
print("Factorial : ",factorial(num))
```

output

```
enter the number to find factorial5
number :  5
Factorial :  120
```

Write a program to implement in_order tree traversal for given graph

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def inorderTraversal(root):
    answer = []

    inorderTraversalUtil(root, answer)
    return answer

def inorderTraversalUtil(root, answer):

    if root is None:
        return

    inorderTraversalUtil(root.left, answer)
    answer.append(root.val)
    inorderTraversalUtil(root.right, answer)
    return

root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
```

```
root.left.right = TreeNode(5)

print(inorderTraversal(root))
```

Output

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

Write a program to implement pre_order tree traversal for given graph

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key


# A function to do preorder tree traversal
def printPreorder(root):

    if root:

        # First print the data of node
        print(root.val),

        # Then recur on left child
        printPreorder(root.left)

        # Finally recur on right child
        printPreorder(root.right)


# Driver code
if __name__ == "__main__":
  root = Node(1)
  root.left = Node(2)
```

```
root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)


# Function call

print("Preorder traversal of binary tree is")

printPreorder(root)
```

Output

```
Preorder traversal of binary tree is
1
2
4
5
3
```

Write a program to implement post _order tree traversal for given graph

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

# A function to do postorder tree traversal
def printPostorder(root):

    if root:

        # First recur on left child
        printPostorder(root.left)

        # the recur on right child
        printPostorder(root.right)

        # now print the data of node
        print(root.val),


# Driver code
if __name__ == "__main__":
  root = Node(1)
  root.left = Node(2)
  root.right = Node(3)
```

```
root.left.left = Node(4)

root.left.right = Node(5)


# Function call

print("\nPostorder traversal of binary tree is")

printPostorder(root)
```

Output

```
Postorder traversal of binary tree is
4
5
2
3
1
```

Write a program to find DFS for a given graph

```
def dfs(node, graph, visited, component):
    component.append(node)    # Store answer
    visited[node] = True     # Mark visited


       # Traverse to each adjacent node of a node
    for child in graph[node]:
         # Check whether the node is visited or not
        if not visited[child]:
            # Call the dfs recursively


            dfs(child, graph, visited, component)


if __name__ == "__main__":


    # Graph of nodes
    graph = {
        0: [2],
        1: [2, 3],
        2: [0, 1, 4],
        3: [1, 4],
        4: [2, 3]
    }
     # Starting node
    node = 0
    visited = [False]*len(graph)
    component = []
```

```
 # Traverse to each node of a graph
dfs(node, graph, visited, component)
print(f"Following is the Depth-first search: {component}")
```

Output

```
Following is the Depth-first search: [0, 2, 1, 3, 4]
```

Write a program to find BFS for a given graph

```
# BFS algorithm in Python



import collections


# BFS algorithm
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)



if __name__ == '__main__':
```

```
graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}

print("Following is Breadth First Traversal: ")

bfs(graph, 0)
```

Output

```
Following is Breadth First Traversal:
0 1 2 3
```

Write a program to implement Hash Table

```
# initializing objects

int_val = 4

str_val = 'GeeksforGeeks'

flt_val = 24.56


# Printing the hash values.

# Notice Integer value doesn't change

#Floating and String value is changed in Hashing

print("The integer hash value is : " + str(hash(int_val)))

print("The string hash value is : " + str(hash(str_val)))

print("The float hash value is : " + str(hash(flt_val)))
```

Output

```
The integer hash value is : 4
The string hash value is : -8948358945277372056
The float hash value is : 1291272085159665688
```