**Applicant’s Name, Last name: Zoleikha Mousavi Pak**

Please answer the following questions:

* What is the difference between a *list* and a *set*?

A **List** is a **sorted** datatype, contains collection of ordered data (with sorted index) that can have **duplicate** value, we can access to items of list with **number of index**.  
  
But a **Set** is **non sorted** datatype, contains collection of non ordered data (without index) that have unique values **without duplicate**, we can access to items of set with **name of items**.

Both of them are mutable (we are able to change them).

* What ways of achieving concurrency do you know? What are the limitations of those ways?

1. Multiprocessing: uses in CPU(most) and I/O Gain processes,

> The limitation of this way is limitation of CPU and Power of CPU, it costs so much cpu.

2. Multithreading : uses in I/O Gain processes.  
> The limitation of this way is racecondition problem, because of using a shared memory, for solving this problem we use Mutex Lock.

3. Synchronous shared memory in Queue : without racecondition problem has itself Mutex Lock.

> The limitation of this way is blocking in run some processes.

4. Asynchronous: uses in maximum ability needed case, that is non-blocking.

> The limitation is when a process is in problem, we should handle with seveal type of issues to solve. For example debugging a web-app that is async.

5. Context switching: causes in a single core environment

> The limitation is time and costs so much time

6. Parallelism: causes in a multi core environments

> The limitation is load balancing, extra working and comunication

* What is the worst case time complexity of a quick sort? O(n2)
* What is an *eigenvalue* and an *eigenvector*?

If there exists a square matrix called *A*, a scalar *λ*, and a non-zero vector v, then *λ* is the eigenvalue and v is the eigenvector if the following equation is satisfied:  
 **A***λ = λv*

In other words, if matrix *A* times the vector v is equal to the scalar *λ* times the vector v, then *λ* is the eigenvalue of v, where v is the eigenvector. *(eigen is a German word means “it’s own”)*

We can utilize Eigenvalues and Eigenvectors to reduce the dimension space. To elaborate, one of the key methodologies to improve efficiency in computationally intensive tasks is to reduce the dimensions after ensuring most of the key information is maintained.

* For a given linked list containing the first 100 numbers (0, 1, 1, 2, 3, 5, 8..., 218922995834555200000), please provide a solution that returns a linked list in the following order: 218922995834555200000, ..., 8, 5, 3, 2, 1, 1, 0.

class Node:  
# Constructor to initialize the node object

def \_\_init\_\_(self, data):

self.items = data

self.next = None

class LinkList:

# Function to initialize head

def \_\_init\_\_(self):

self.head = None

# Function to reverse the linked list

def reverse\_linkedlist(self):

prev = None

n = self.head

while n is not None:

next = n.next

n.next = prev

prev = n

n = next

self.head = prev

# Function to insert a new node at the beginning

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

# Utility function to print the linked LinkedList

def printList(self):

temp = self.head

while temp:

print(temp.items)

temp = temp.next

# Driver program to test above functions

llist = LinkList()

data\_list = input('Input the elements in the linked list, separate with space: ').split()

count=0

for data in data\_list:

count += 1

if count <=100:

llist.push(int(data))

else:

llist

print("Given Linked List")

llist.printList()

llist.reverse\_linkedlist()

print("\nReversed Linked List")

llist.printList()