Week 3

Sunday, August 24, 2025 3:37 PM

Quick sort → Uses divide and conquer algo. chose a pivot. Correct place of pivot is such that, left side elements are smaller than pivot; right side elements are greater than pivot. Repeatative process, until the entire list gets sorted!

Partition function

```python
def partition(L,lower,upper):
  # Select first element as a pivot
  pivot = L[lower]
  i = lower
  for j in range(lower+1,upper+1):
    if L[j] <= pivot:
      i += 1
      L[i],L[j] = L[j],L[i]
  L[lower],L[i]= L[i],L[lower]
  # Return the position of pivot
  return i
```

→ 1st element as pivot

→ for j in (lower+1 to last element)

move i

swap i & j element

place pivot at correct place

Return location of pivot!

Implementing Quick sort with the help of partition function

```python
def quicksort(L,lower,upper):
  if(lower < upper):
    pivot_pos = partition(L,lower,upper);
    # Call the quick sort on leftside part of pivot
    quicksort(L,lower,pivot_pos-1)
    # Call the quick sort on rightside part of pivot
    quicksort(L,pivot_pos+1,upper)
  return L
```

Finding pos of pivot with the help of Partition func

→ Quick sort on left half

→ Quick sort on right half

↳ Sorted list!

Recurrance Relation → for Best case : $T(n) = 2T(n/2) + O(n)$

→ for Worst case : $T(n) = T(n-1) + O(n)$

Complexity → for Best case : $O(n \log n)$ (In each call, pivot is at middle and divides the list)

→ for avg case : $O(n \log n)$

→ for worst case : $O(n^2)$ (Already sorted list)

→ for worst case : $O(n^2)$ (Already sorted list)

→ Not Stable → Inplace sorting

Master Theorem for decreasing func → General form: $T(n) = a\,T(n-b) + O(n^k)$

for $a=1 \Rightarrow O(n \times n^k)$

for $a>1 \Rightarrow O(n^k a^{n/b})$

for worst case of Quick sort;

$\Rightarrow T(n) = T(n-1) + O(n)$

$\Rightarrow a=1$ & $k=1$

$\Rightarrow$ Ans : $O(n \times n^1) \Rightarrow O(n^2)$

STACK → Last in, First out. Like a stack of Plates, the one on the top was placed last, but will be picked 1st.

Operations include push and pop → Removing the element at top.

push ↓ Placing new element at top.

```
class Stack:
    def __init__(self):
        self.stack = []
    def isempty(self):
        return(self.stack == [])
    def Push(self,v):
        self.stack.append(v)
    def Pop(self):
        v = None
        if not self.isempty():
            v = self.stack.pop()
        return v
    def __str__(self):
        return(str(self.stack))
```

→ Initialization

→ Checking empty stack

→ Adding/placing element at top

→ Removing element at top

↳ Returning the stack. This could be traversed!

Queue → First in, First out. Much like the line you stand in at a

Queue → First in, First out. Much like the line you stand in at a BANK. The earlier you come, the earlier you get free.

operations include enqueue and dequeue → Removing from the queue.

enqueue → Adding into the Queue

```
class Queue:
    def __init__(self):
        self.queue = []
    def isempty(self):
        return(self.queue == [])
    def enqueue(self,v):
        self.queue.append(v)
    def dequeue(self):
        v = None
        if not self.isempty():
            v = self.queue[0]
            self.queue = self.queue[1:]
        return v
    def __str__(self):
        return(str(self.queue))
```

- `__init__` → Initializing the queue
- `isempty` → Checking for empty queue
- `enqueue` → entering into the queue
- `dequeue` → Removing from the queue
- `__str__` → Printing the Queue. This could be traversed!

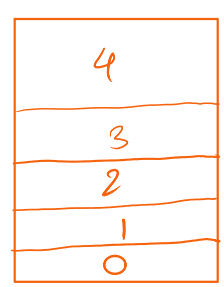
QUIZ May 23

Q

```
for i in range(5):
    S.push(i)
    Q.Enqueue(i)
while not Q.isEmpty():
    S.push(Q.Dequeue())
while not S.isEmpty():
    Q.Enqueue(S.pop())
while not Q.isEmpty()
    print(Q.Dequeue(), end=" ")
```

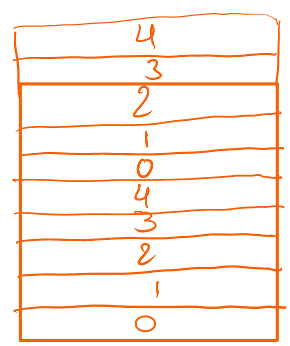
output?

S =



Q = [0, 1, 2, 3, 4]

S =



Q = [ ]

Q = [4, 3, 2, 1, 0, 4, 3, 2, 1, 0]

⇒ 4 3 2 1 0 4 3 2 1 0 Ans.

QUIZ Jan 203

Q def fun(Q):

Queue Stack

Add [Dequeue() pop()

Stack

Ans.

QUIZ Jan 2023

Q.

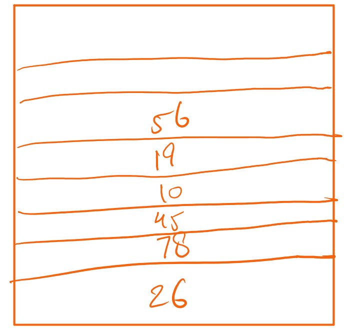
```
def fun(Q):
    while (!Q.isEmpty()):
        temp = Q.Dequeue()
        S.push(temp)

    while (!S.isEmpty()):
        temp = S.pop()
        Q.Enqueue(temp)
```

Queue	Stack
Add Dequeue()	pop()
Remove Enqueue()	push()

Given Q = [26, 78, 45, 10, 19, 56] and stack S is empty. What will be content of Q after fun finishes it execution?

Stack



Ans.

Q = [56, 19, 10, 45, 78, 26]

Linear Hashing → A data structure that allows efficient insertion, deletion

Q. Consider inserting the key 24, 36, 58, 65, 79 into a hash table of size m = 11 using linear probing, the primary hash function is h(k) = k mod m. What will be the hash table after insterting all keys in given order?

0	1	2	3	4	5	6	7	8	9	10
		24	36	58	79					65

Ans.

QUIZ 1 SEPT 2023

Q. The key values are integers and hash function used is key mod 13. Key values 14, 55, 144, 83, 122, 131 are inserted into the table in what index would key 131 be inserted?

0	1	2	3	4	5	6	7	8	9	10	11	12
	14	144	55	131	83	122						

Q. h(key) = key mod 5, use linear probing for solving collisions. Key values are 45, 51, 60, 18, 34. Find order of insertion of values in the hash table?

0	1	2	3	4
45	51	60	18	34

Linked List → Basic element is a node!

A node :

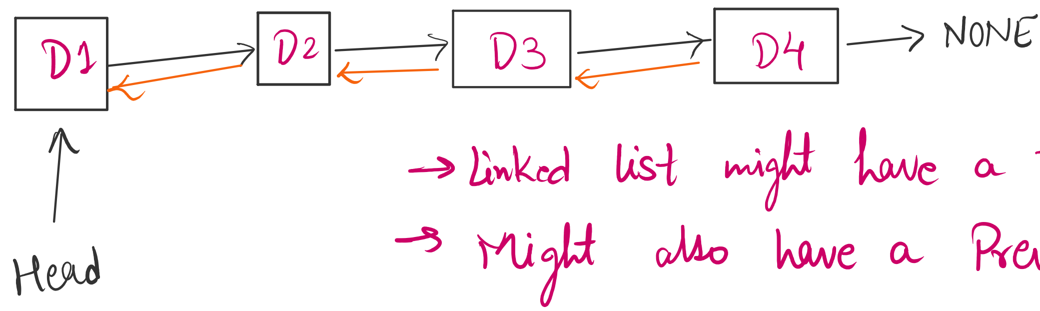
Data	Info

⇒ Data.next points to the next node.

Nodes combine together to create a linked list.

singly linked list

Doubly linked list



→ Linked list might have a tail.

→ Might also have a Previous arrow as well.

→ Head is used to traverse the list.

→ Insertion, Deletion and traversal are important things related to linked list.

Advantage

- Insertion and deletion operations are easy
- Many complex applications can be easily carried out with linked list concepts like tree, graph, etc.

Disadvantage

- More memory required to store data
- Random access is not possible

Application

- Implementation stack, queue, deque
- Representation of graph.
- Representation of sparse matrix
- Manipulation of the polynomial expression