Algorithms

1. Linear search
2. Binary Search
3. Bubble sort
4. Insertion Sort

1. Linear Search

In [ ]:
```python
#Python program for Linear Search
#create array to store all the numbers
myList = [4, 2, 8, 17, 9, 3, 7, 12, 34, 21]
#enter item to search for
item = int(input("Please enter item to be found "))
found = False
for index in range(len(myList)):
    if(myList[index] == item):
        found = True
if(found):
    print("Item found")
else:
    print("Item not found")
```

In [ ]:
```python
len(myList)
```

In [ ]:
```python
# my code here
lowerBound = 0
upperBound = len(myList) - 1
item = input("Enter the item to be searched: ")
found = False
index = lowerBound
while index<upperBound or found != True:
    if item == myList[index]:
        found = True
        break
    elif found:
        print("found")
        break
    else:
        print("Item not found")
        break
    index= index+ 1

```

In [ ]:
```
# 2. Binary search
```

In [ ]:
```
myList = [2,3,4,7,8,9,12,17,21,34]
```

In [ ]:
```
lowerBound = 0
upperBound = len(myList)-1
item = int(input("Enter the item to be searched: "))


while lowerBound <= upperBound:
    index = (lowerBound + upperBound)//2
    print(index)
    #break
    if item == myList[index]:
        print("Found at", index)
    if item < upperBound:
        lowerBound = index + 1
    if item > myList[index]:
        upperBound = index -1
```

In [ ]:

```python

# Python3 code to implement iterative Binary
# Search.

# It returns location of x in given array arr
# if present, else returns -1
def binarySearch(arr, l, r, x):

    while l <= r:

        mid = l + (r - l) // 2;

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1

    # If we reach here, then the element
    # was not present
    return -1

# Driver Code
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print ("Element is present at index % d" % result)
else:
    print ("Element is not present in array")
```

In [ ]:

```
myList = [2,3,4,7,8,9,12,17,21,34]
lowerBound = 0
UpperBound = 9
value = int(input("Enter the item you are searching for "))
found= False
while lowerBound<=UpperBound:

    index = (lowerBound + UpperBound)//2
    #print(index)
    if myList[index]==value:
        print("Found at ", index)
        found = True
        break
    if myList[index]>value:
        UpperBound = index - 1
    if myList[index]<value:
        lowerBound = index + 1
    lowerBound = lowerBound + 1
if not found:
    print("Not found")

```

3. Bubble Sort

In [ ]:

```
%%time
#Python program for Bubble Sort
myList = [70,46,43,27,57,41,45,21,14]
top    = len(myList)
swap = True
#Pre-condition loop
while (swap) or (top > 0):
    swap = False
    for index in range(top - 1):

        if myList[index] > myList[index + 1]:
            temp = myList[index]
            myList[index] = myList[index + 1]
            myList[index + 1] = temp
            swap = True
    top = top - 1
#output the sorted array
print(myList)
```

In [ ]:

```
len(myList)
```

In [ ]:
```
%%time
l = 0
u = 8
top = u
swap=True
while swap or l<top:
    swap = False
    for index in range(0,top-1):
        if myList[index]>myList[index+1]:
            temp= myList[index]
            myList[index]=myList[index+1]
            myList[index+1] = temp
            swap = True
    top = top - 1
print(myList)


```

Bubble sort - Decending order

In [ ]:
```
%%time
l = 0
u = 8
top = u
swap=True
while swap or l<top:
    swap = False
    for index in range(0,top-1):
        if myList[index]<myList[index+1]:
            temp= myList[index]
            myList[index]=myList[index+1]
            myList[index+1] = temp
            swap = True
    top = top - 1
print(myList)

```

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

In [ ]:
```
bubbleSort(myList)
```

In [ ]:
```
# Q 2 9618 MJ 42
def linearSearch(searchValue):

    for x in range(0, 10):
        if arrayData[x] == searchValue:
            return True
    return False
```

In [ ]:
```
arrayData = [10, 5, 6, 7, 1, 12, 13, 15, 21, 8]
linearSearch(4)
```

Insertion Sort

In [ ]:
```
myList = [5,8,7,2,4,3,9,10]
lowerBound = 0
upperBound = 7
for i in range(1, len(myList)):
    print(i)
    key = myList[i]

    place = i - 1
    if myList[place] > key:
        while place >= lowerBound and myList[place] > key:
            temp = myList[place+1]
            myList[place + 1] = myList[place]
            myList[place] = temp
            place = place - 1
        myList[place+1] = key
        print("key", key, "place", place, "temp", temp)
print(myList)
```

In [ ]:
```
def insertion_sort(myList):
    myList = [5,8,7,2,4,3,9,10]
    lowerBound = 0
    upperBound = 7
    for i in range(1, len(myList)):
        key = myList[i]
        place = i - 1
        if myList[place] > key:
            while place >= lowerBound and myList[place] > key:
                temp = myList[place+1]
                myList[place + 1] = myList[place]
                myList[place] = temp
                place = place - 1
            myList[place+1] = key
    return myList

```

In [ ]:
```
import tkinter

```

In [ ]:
```
%%time
insertion_sort(myList)
```

19. 1 Cont.... Abstract Data types (ADT)s

1. Stacks
2. Queues
3. Linked Lists
4. Binary Trees
5. Graphs
6. Dictionary
7. Big O notation

Stacks

In [ ]:
```
# creating a stack with the size of ten items
stack = [None for index in range(0,10)]
basePointer = 0
topPointer = -1
stackFull = 10
item = None
```

In [ ]:
```
# size of the stack
size  = len(stack)
print(size)
```

In [ ]:
```
stackFull
```

In [ ]:
```
# push
def push(item):
    global topPointer
    if topPointer < stackFull - 1:
        topPointer = topPointer + 1
        stack[topPointer] = item
    else:
        print("Stack is full, cannot push")
```

In [ ]:
```python
# pop
def pop():
    global topPointer, basePointer, item
    if topPointer == basePointer -1:
        print("Stack is empty,cannot pop")
    else:
        topPointer = topPointer -1
        item = stack[topPointer]
    return stack
```

In [ ]:
```python
def pop():
    global topPointer, basePointer, item
    if topPointer == basePointer -1:
        print("Stack is empty,cannot pop")
    else:
        item = stack[topPointer]
        topPointer = topPointer -1
```

Queues

In [ ]:
```python
# creating a new queue
queue = [None for index in range(0,10)]
frontPointer = 0
rearPointer = -1
queueFull = 10
queueLength = 0
```

In [ ]:
```python
#enQueue- adding stuff to the queue
def enQueue(item):
    global queueLength, rearPointer
    if queueLength < queueFull:
        if rearPointer < len(queue) - 1:
            rearPointer = rearPointer + 1
        else:
            rearPointer = 0
        queueLength = queueLength + 1
        queue[rearPointer] = item
    else:
        print("Queue is full, cannot enqueue")
```

In [ ]:
```
def deQueue():
    global queueLength, frontPointer, item
    if queueLength == 0:
        print("Queue is empty,cannot dequeue")
    else:
        item = queue[frontPointer]
        if frontPointer == len(queue) - 1:
            frontPointer = 0
        else:
            frontPointer = frontPointer + 1
    queueLength = queueLength -1

```

Linked Lists

In [ ]:
```
# Setup Linked list and search items
```

In [2]:
```
#Python program for finding an item in a linked list
myLinkedList = [27, 19, 36, 42, 16, None, None, None, None, None, None, N
myLinkedListPointers = [-1, 0, 1, 2, 3 ,6 ,7 ,8 ,9 ,10 ,11, -1]
startPointer = 4
nullPointer = -1
heapStartPointer = 5

def find(itemSearch):
    found = False
    itemPointer = startPointer
    while itemPointer != nullPointer and not found:
        if myLinkedList[itemPointer] == itemSearch:
            found = True
        else:
            itemPointer = myLinkedListPointers[itemPointer]
    return itemPointer

```

In [3]:
```
#enter item to search for
item = int(input("Please enter item to be found "))
result = find(item)
if result != -1:
    print("Item found")
else:
    print("Item not found")
```

```
Please enter item to be found 5
Item not found
```

In [4]:
```
#enter item to search for
item = 42
result = find(item)
if result != -1:
    print("Item found")
else:
    print("Item not found")
```

```
Item found
```

In [5]:
```
startPointer
```

Out[5]:
```
4
```

In [6]:
```
def insert(itemAdd):
    global startPointer, heapStartPointer
    if heapStartPointer == nullPointer:
        print("Linked List full")
    else:
        tempPointer = startPointer
        startPointer = heapStartPointer
        heapStartPointer = myLinkedListPointers[heapStartPointer]
        myLinkedList[startPointer] = itemAdd
        myLinkedListPointers[startPointer] = tempPointer
```

In [7]:
```
insert(35)
insert(98)
insert(50)
insert(45)
insert(56)
insert(78)

```

In [8]:
```
myLinkedList
```

Out[8]:
```
[27, 19, 36, 42, 16, 35, 98, 50, 45, 56, 78, None]
```

In [9]:
```
insert(6)
myLinkedList
```

Out[9]:
```
[27, 19, 36, 42, 16, 35, 98, 50, 45, 56, 78, 6]
```

In [10]:
```
nullPointer
```

Out[10]:
```
-1
```

In [11]:
```
1 startPointer
2
```

Out[11]: 11

In [12]:
```
1 heapStartPointer
```

Out[12]: -1

In [13]:
```
1 myLinkedList
```

Out[13]: [27, 19, 36, 42, 16, 35, 98, 50, 45, 56, 78, 6]

In [ ]:
```
1
```

19.2 Recursion

In [ ]:
```
1 def factorial(x):
2     # base case
3     if x == 1:
4         return 1
5     else:
6
7         return x * factorial(x-1)
8 print ('The factorial of 5 is', factorial(5))
```

Sample Paper 2021

In [ ]:
```
1
```

In [1]:
```
1 # Question 1 (a)
2 TheData =  [20,3,4,8,12,99,4,26,4]
```

In [2]:

```
# 1 (b)
def InsertionSort(TheData):
    for count in range(0,len(TheData)):
        DataToInsert = TheData[count]
        Inserted = 0
        NextValue = count -1
        while NextValue >= 0 and Inserted != 1:
            if DataToInsert < TheData[NextValue]:
                TheData[NextValue+ 1] = TheData[NextValue]
                NextValue = NextValue - 1
                TheData[NextValue + 1] =  DataToInsert
            else:
                Inserted = 1


```

In [3]:

```
# 1 (c)
def PrintArray(TheData):
    for count in range(0, len(TheData)):
        print(TheData[count])
```

In [4]:

```
# 1 (d)
print("The data before sorting ")
PrintArray(TheData)
InsertionSort(TheData)
print("The data after sorting ")
PrintArray(TheData)
```

```
The data before sorting
20
3
4
8
12
99
4
26
4
The data after sorting
3
4
4
4
8
12
20
26
99
```

In [5]:
```
#1 (e)

def Search(number):
    for i in range(len(TheData)):

        if number == TheData[i]:
            print("Found")
            return True
    else:
        print("Not found")
        return False

```

In [7]:
```
Search(5)
```

```
Not found
```

Out[7]:
```
False
```

In [ ]:
```

```