ASSIGMENT OF ***ANALYSIS*** OF ALGORITHM

NAME:ALI ABDUL BASIT

ROLL NO:2K18/CSME/4

TEACHER:DR NAJMA CHANNA

DATE:20-02-2021

GITHUB LINK: https://github.com/basit376/Analysis-Of-Algorthem-Assigment.git

ALGORITHM OF INSERTION SORT

- Algorithm
- To sort an array of size n in ascending order:
- 1: Iterate from arr[1] to arr[n] over the array.
- 2: Compare the current element (key) to its predecessor.
- 3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

IMPLEMENTATION OF INSERTION SORT IN PYTHON

```
# Insertion sort in Python
def insertionSort(array):
  for step in range(1, len(array)):
    key = array[step]
    j = step - 1
    # Compare key with each element on the left of it until an element smaller than it is found
    # For descending order, change key<array[j] to key>array[j].
    while j >= 0 and key < array[j]:
      array[j + 1] = array[j]
      j = j - 1
    # Place key at after the element just smaller than it.
    array[j + 1] = key

data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

BUBBLE SORT IMPLEMENTATION IN PYTHON

```
# Creating a bubble sort function
def bubble_sort(list1):
    # Outer loop for traverse the entire list
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1

list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ", list1)
# Calling the bubble sort function
print("The sorted list is: ", bubble_sort(list1))
```

Divide and conquer implementation IN PYTHON

- Merge sort is one of the most prominent divide-and-conquer sorting algorithms in the modern era. It can be used to sort the values in any traversable data structure such as a list.

```
def mergeSort(arr):
  if len(arr) > 1:

    # Finding the mid of the array
    mid = len(arr) // 2

    # Dividing the array elements
    L = arr[:mid]

    # into 2 halves
    R = arr[mid:]

    # Sorting the first half
    mergeSort(L)

    # Sorting the second half
    mergeSort(R)

    i = j = k = 0
```

```
# Copy data to temp arrays L[] and R[]
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    # Checking if any element was left
    while i < len(L):
        arr[k] = L[i]
        i += 1
        k += 1
```

```
 while j < len(R):
          arr[k] = R[j]
          j += 1
          k += 1
# Code to print the list

def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

# Driver Code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)
```

Divide and conquer SECOUND EXAMPLE Implementation IN PYTHON

Like Merge Sort, QuickSort is a Divide and Conquer algorithm.

```
# THIS IS CODE FOR QUICK SORT
# divide function
def partition(arr,low,high):
  i = ( low-1 )
  pivot = arr[high] # pivot element
  for j in range(low , high):
    # If current element is smaller
    if arr[j] <= pivot:
      # increment
      i = i+1
      arr[i],arr[j] = arr[j],arr[i]
  arr[i+1],arr[high] = arr[high],arr[i+1]
  return ( i+1 )
```

```python
# sort
def quickSort(arr,low,high):
  if low < high:
    # index
    pi = partition(arr,low,high)
    # sort the partitions
    quickSort(arr,  low, pi-1)
    quickSort(arr,  pi+1, high)
# main
arr = [2,5,3,8,6,5,4,7]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
  print (arr[i],end=" ")
```

Divide and conquer Third EXAMPLE Implementation IN PYTHON

```
# Iterative Binary Search Function method Python Implementation
# It returns index of n in given list1 if present,
# else returns -1
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")
```

ALGORITHM FOR TOWER OF HANOI

- 1. Create function hanoi that takes the number of disks n and the names of the source, auxiliary and target pegs as arguments.
- 2. The base case is when the number of disks is 1, in which case simply move the one disk from source to target and return.
- 3. Move n – 1 disks from source peg to auxiliary peg using the target peg as the auxiliary.
- 4. Move the one remaining disk on the source to the target.
- 5. Move the n – 1 disks on the auxiliary peg to the target peg using the source peg as the auxiliary.

IMPLENTATION OF TOWER OF HANOI IN PYTHON

```
def hanoi(disks, source, auxiliary, target):
    if disks == 1:
        print('Move disk 1 from peg {} to peg {}.'.format(source, target))
        return

    hanoi(disks - 1, source, target, auxiliary)
    print('Move disk {} from peg {} to peg {}.'.format(disks, source, target))
    hanoi(disks - 1, auxiliary, source, target)


disks = int(input('Enter number of disks: '))
hanoi(disks, 'A', 'B', 'C')
```

THE END