NAME : Wasit Shafi Roll no : 18MCA054

Q1. Write a python program for the implementation of quick-sort algorithm.
Try on this example: {1, 4, 3, 67, 32, 21, 25, 54, 76}. Using pen and paper show the state of list after each step of selecting the pivot.

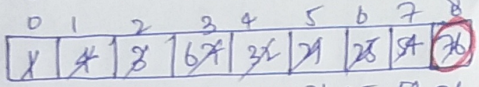
Sol.

```
# Program
def getPartitionIndex(arr, low, high):
    i = low - 1
    for j in range(low, high):
        if arr[j] <= arr[high]:
            i = i + 1
            arr[i], arr[j] = arr[j], arr[i]
    i = i + 1
    arr[i], arr[high] = arr[high], arr[i]
    return i

def quickSort(arr, low, high):
    if low < high:
        partitionIndex = getPartitionIndex(arr, low, high)
        quickSort(arr, low, partitionIndex - 1)
        quickSort(arr, partitionIndex + 1, high)

if __name__ == "__main__":
    arr = [1, 4, 3, 67, 32, 21, 25, 54, 76]
    print('Array Before quickSort() : ', arr)
    quickSort(arr, 0, len(arr) - 1)
    print('Array After quickSort()  : ', arr)
```

Name: Wasit Shafi Roll no: 18MCA0054

QuickSort(low = 0, High = 8), PartitionIndex = 8

getPartitionIndex() Arr

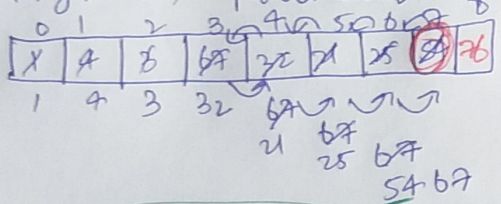


, Pivot = 76

Return 8

QuickSort (low = 0, High = 7) PartitionIndex = 6

getPartitionIndex() arr

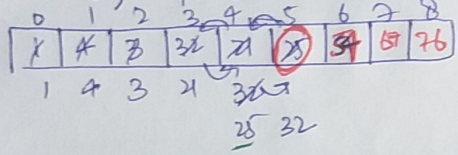


Pivot = 54

Return 6

Quicksort (low = 0, High = 5), PartitionIndex = 4

getPartitionIndex() arr

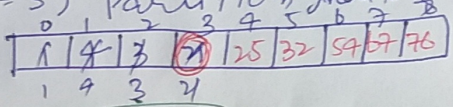


Pivot = 25

Return 4

Quick sort (low = 0, High = 3) PartitionIndex = 3

getPartitionIndex() arr

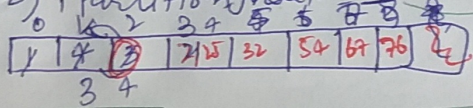


Pivot = 21

Return 3

QuickSort(low = 0, High = 2), PartitionIndex = 1

getPartitionIndex() arr



Pivot = 3

Return 1

QuickSort (low = 0, High = 0) X

QuickSort (low = 2, High = 2) X

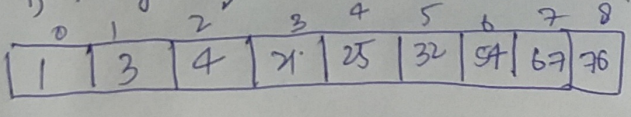
QuickSort (low = 4, High = 3) X

QuickSort (low = 5, High = 5) X

QuickSort (low = 7, High = 7) X

QuickSort (low = 9, High = 8) X

Array after QuickSort =



Q2. Generate a list of 1 million random number in the range(1,9000). Sort the list using merge sort and quick sort. Print the runtime difference of the two algorithm.

Sol.

```
#program
import time
import random

def getPartitionIndex(arr, low, high):
    i = low - 1
    for j in range(low, high):
        if (arr[j] <= arr[high]):
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i

def quickSort(arr, low, high):
    if (low < high):
        partitionIndex = getPartitionIndex(arr, low, high)
        quickSort(arr, low, partitionIndex - 1)
        quickSort(arr, partitionIndex + 1, high)

def merge(arr, low, mid, high):
        i = 0
        j = 0
        k = low
        n1 = mid - low + 1
        n2 = high - mid
        left = arr[low : mid + 1]
        right = arr[mid + 1 : high + 1]

        while(i < n1 and j < n2):
            if(left[i] <= right[j]):
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1

        while(i < n1):
             arr[k] = left[i]
             k +=  1
             i += 1
        while(j < n2):
             arr[k] = right[j]
             k += 1
             j += 1

def mergeSort(arr, low, high):
    if (low < high):
```

```
        mid = low + (high - low) // 2
        mergeSort(arr, low, mid)
        mergeSort(arr, mid + 1, high)
        merge(arr, low, mid, high)

if __name__ == "__main__":
    arr = []
    brr = []
    N = 1000000
    MAX = 9000
    for i in range(N):
        randomNumber = random.randint(1, MAX)
        arr.append(randomNumber)
        brr.append(randomNumber)
    # print('Array[] : ', arr)

    # Quick Sort
    # Quick sort is more efficient and works faster than merge sort in case of smaller array size.
    start = time.time()
    quickSort(arr, 0, len(arr) - 1)
    end = time.time()
    print('Time taken to sort array using quick sort is :', float("{:.4f}".format(end - start)), 'sec')

    # Merge Sort
    # Merge sort is more efficient and works faster than quick sort in case of larger array size.
    start = time.time()
    mergeSort(brr, 0, len(brr) - 1)
    end = time.time()
    print('Time taken to sort array using merge sort is :', float("{:.4f}".format(end - start)), 'sec')
```

Q3. Implement the given problem in python:
Given a rod of length n inches and an array of prices that contains prices of all pieces of size smaller than n.
Determine the maximum value obtainable by cutting up the rod and selling the pieces. For example, if length of the
rod is 8 and the values of different pieces are given as following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)
length | 1 2 3 4 5 6 7 8
---
price
| 1 5 8 9 10 17 17 20
And if the prices are as following, then the maximum obtainable value is 24 (by cutting in eight pieces of length 1)
length | 1 2 3 4 5 6 7 8
---
price
| 3 5 8 9 10 17 17 20

Sol.

```
# Program

INT_MIN = -32767
```

```
def cutRod(price, n):
	val = [0 for x in range(n+1)]
	val[0] = 0

	for i in range(1, n+1):
		max_val = INT_MIN
		for j in range(i):
			max_val = max(max_val, price[j] + val[i-j-1])
		val[i] = max_val

	return val[n]

arr = [1, 5, 8, 9, 10, 17, 17, 20]
#arr = [3, 5, 8, 9, 10, 17, 17, 20]
size = len(arr)
print("Maximum Obtainable Value is " + str(cutRod(arr, size)))
```