HOME WORK - 3

Name: Sai Rohit Kalyan Gandham.

StudentID: 1008070724

EmailID: Sxg0724@mavs.uta.edu

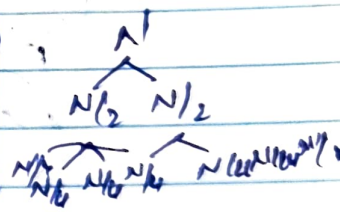
Sol) 4-2:

Given an array is passed by pointer. Time = O(1)
* an array by copying is Time $\geq \Omega(N)$ N: Size.
* an array passed by copying and Subrange that might be accurd procedure Time = $\Theta(q-p+1)$
Subarray A[p..q]

a) Given as per then $T(n) = T(n/2) + c \cdot \Theta(\lg n)$

2. $O(n \lg n)$

$T(n) = T(n/2) + cN$
$= 2cN + T(n/4)$
$= 3cN + T(n/8)$
$= \sum_{i=0}^{\lg n - 1} (2^i cN/2^i)$
$= cN \lg N$
$= O(n \lg n)$



3. $T(n) = T(n/2) + cn$ $\quad aT(n/b) + \Theta(N)$

master method

$n^d = n^1 \quad d=1, a=1, b=2$

$\log_b a > d \Rightarrow \log_2 1 = 0$

$0 < d \Rightarrow f(n) \Rightarrow O(f(n))$

$= O(n)$.

b) Master's method.

Sol) $T(n) = 2T(n/2) + cn$

Given $2T(n/2) + cn$ compare with master method formal. $aT(n/b) + cn$.

$a = 2, b = 2$ and $f(n) = c(n)$.

$\Rightarrow n^d = n^1 \Rightarrow \boxed{d = 1}$

$\Rightarrow \log_b a, \log_2 d = 1$

$d = 1 = \log_b a = \log_2 = 2$ so $n \log n$

$n \log n$ Hence proved

② Given $T(n) = 2T(n/2) + cn + 2N$

By Not master theorm so change using Substion method.

$T(n) = 2T(n/2) + cn + 2N \rightarrow ①$

$T(n/2) = 2T(n/4) + \frac{c(n)}{2} + 2N \rightarrow ②$

$T(n/4) = 2T(n/8) + c(n/4) + 2N \rightarrow ③$

Now Eq ② into Equation ①.

$T(n) = 2(2T(n/4) + cn + 2N) + cn + 2N$

$= 4T(n/4) + 2c(n/2) + 4N + cn + 2N.$

$= 4T(n/4) + 2c(n/2) + 6N + cn$

$= 4T(n/4) + 4c(n/2) + 6N.$

$\hookrightarrow$ Eq ③

$\Rightarrow 4T(n/4) + 2c(n/2) + cn + 4N$

$2\{2T(n/8) + c(n/4) + 2N) + 2c(n/2) + cn + 4N)$

$8T(n/8) + 4c(n/4) + 8N + 2c(n/2) + cn + 4N$

$8T(n/8) + 4c(n/4) + 2cn + 8N$

$= \sum_{i=0}^{\lg n - 1} 2^3(n/2^3) + 2^2 c(n/2^2) + 2^1 cn + 2^3 N)$

$= \sum_{i=0}^{\lg n - 1} 2^i N + cn$

$= cn \lg n + nN - N = O(nN)$

Sol) ③ $T(n) = 2T(n/2) + cn + 2(n/2)$

$= 2T(n/2) + n(c+1)$ → master theorm.

Compare with master thoorm.

$aT(n/b) + f(n)$

$\Rightarrow a = 2, b = 2, n^d = n^1 \Rightarrow d = 1$

$\log_b a = d \Rightarrow \log_2 2 = d = 1$ all Equal

Ans ② $(n^d \log n) \Rightarrow n^1 \log n$

$: O(n \log n)$

7.2

Sol) Given problem Statement of is Support all Element values are Same?
To find: Randomized Quicksort Running time?

Sol) The partition value (or) of the Randomized Quicksort is return's (or) because all Element Same. look Sudo Code: in python.

```
def partitionIndex(nums, start, stop):
    pivot = start
    i = start + 1
    for j in range(start+1, stop+1):
        if num[j] ≤ num[pivot]
            (num[i], num[j] = num[j], num[i])
            i = i+1
    (num[pivot], num[i-1] = num[i-1], num[pivot])
    pivot = i-1
    return pivot
def randomizedpartition(arr, start, stop)
    randompivot = random.randrange(start, stop)
    (arr[start], arr[randompivot]) =
        (arr[randompivot], arr[start])
    return partitionIndex(arr, start, stop).
def QuickSort(arr, start, stop):
    if (start < stop):
        pivotIndex = partitionrandom(arr, start, stop)
```

Quicksort (arr, start, pivotIndex-1)
Quicksort (arr, ~~start~~ pivotIndex-1, ~~stop~~)

- So here each split will be (n-1)-to-1
  So
  $O(n^2)$.

(b) Like as per the above code of Randomized Quick sort,

Sol)
- The new(partition) is also similar to the partition, Except one when you arrange the Elements according to (pivot q: it moves all elements [by q: A[t] := Y] from the right of the pivot.
- So what happens is the procedure also send second pass at the array, so the $O(n) + O(n) \Rightarrow O(n) = O(n-p)$

(c)
- Since all the Elements are same we have compare Best Case instead of Worst case.
- The QuickSort works on divide and Conquer. so the Equation looks like this
  $T(n) = 2T(n/2) + O(n)$
- applying master theorem.
  $aT(n/b) + n^d$
- $a=2$, $b=2$ and $d=1$
- $\log_b a = \log_2 = 1$ and also $d=1$ Case(2)

* ie $n^d = f(n)$ $\quad f(n) = n\log(n)$

$n^d = n^1 \log(n)$

$\Rightarrow O(n\log(n))$.

(d) This is code please find below.

Ans 4)

(c) The modified code for Tail-recursion-QuicuSort so that worst case stack dept will be $O(\lg n)$ maintain the $O(n \lg n)$ Expected running time of the algorithms.

Sol)

Original Tail Recursive QuicuSort

* The pseudo code is

Tail Recursive QuicuSort (A, p, r)

while $p < r$

q : partition (A, p, r)

Tail. recursive - Quution $(A, p, q-1)$

$p = q + 1$

Modified Tail Recursive QuickSort

```
Optimised-Tail-Recursive-QuickSort (A,p,r)
    while p<r
    if  q < floor ((p+r)/2)
        New-Tail-Recursive-QuickSort(A,p,q-1)
            p=q+1
    Else
        New-Tail-Recursive-QuickSort (A,q+1,r)
            r=q-1
```

Name: Sai Rohit Kalyan Gandham

StudentID: 1002070724

EmailID: sxg0724@mavs.uta.edu

7.2)

d) **********RANDOMIZED_QUICKSORT()*********

```
import random

def quicksortPivotAsFirstElement(arrayToSort, startPointer , endPointer):
   if(startPointer < endPointer):
      pivotindex = partitionForRandomNumber(arrayToSort, startPointer, endPointer)
      quicksortPivotAsFirstElement(arrayToSort , startPointer , pivotindex-1)
      quicksortPivotAsFirstElement(arrayToSort, pivotindex + 1, endPointer)

def partitionForRandomNumber(arrayToSort , startPointer, endPointer):
   randpivot = random.randrange(startPointer, endPointer)
   arrayToSort[startPointer], arrayToSort[randpivot] = arrayToSort[randpivot], arrayToSort[startPointer]
   return partition(arrayToSort, startPointer, endPointer)

def partition(arrayToSort,startPointer,endPointer):
   pivot = startPointer

   intialIndex = startPointer + 1
   for secondIndex in range(startPointer + 1, endPointer + 1):
      quicksortPivotAsFirstElement.x += 1
      if arrayToSort[secondIndex] <= arrayToSort[pivot]:
         arrayToSort[intialIndex] , arrayToSort[secondIndex] = arrayToSort[secondIndex] ,
arrayToSort[intialIndex]
         intialIndex = intialIndex + 1
```

```
    arrayToSort[pivot] , arrayToSort[intialIndex - 1] = arrayToSort[intialIndex - 1] , arrayToSort[pivot]
    pivot = intialIndex - 1
    return (pivot)


if __name__ == '__main__':
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quicksortPivotAsFirstElement.x = 0
    result = quicksortPivotAsFirstElement(arrayToSort, 0, len(array) - 1)
    print(arrayToSort)
    print("The No of recursive calls for RANDOMIZED_QUICKSORT: ",quicksortPivotAsFirstElement.x)
```

In [125]:
```
import random

def quicksortPivotAsFirstElement(arrayToSort, startPointer , endPointer):
    if(startPointer < endPointer):
        pivotindex = partitionForRandomNumber(arrayToSort, startPointer, endPointer)
        quicksortPivotAsFirstElement(arrayToSort , startPointer , pivotindex-1)
        quicksortPivotAsFirstElement(arrayToSort, pivotindex + 1, endPointer)

def partitionForRandomNumber(arrayToSort , startPointer, endPointer):
    randpivot = random.randrange(startPointer, endPointer)
    arrayToSort[startPointer], arrayToSort[randpivot] = arrayToSort[randpivot], arrayToSort[startPointer]
    return partition(arrayToSort, startPointer, endPointer)

def partition(arrayToSort,startPointer,endPointer):
    pivot = startPointer

    intialIndex = startPointer + 1
    for secondIndex in range(startPointer + 1, endPointer + 1):
        quicksortPivotAsFirstElement.x += 1
        if arrayToSort[secondIndex] <= arrayToSort[pivot]:
            arrayToSort[intialIndex] , arrayToSort[secondIndex] = arrayToSort[secondIndex] , arrayToSort[intialIndex]
            intialIndex = intialIndex + 1
    arrayToSort[pivot] , arrayToSort[intialIndex - 1] = arrayToSort[intialIndex - 1] , arrayToSort[pivot]
    pivot = intialIndex - 1
    return (pivot)


if __name__ == '__main__':
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quicksortPivotAsFirstElement.x = 0
    result = quicksortPivotAsFirstElement(arrayToSort, 0, len(array) - 1)
    print(arrayToSort)
    print("The No of recursive calls for RANDOMIZED_QUICKSORT: ",quicksortPivotAsFirstElement.x)
```

```
[2, 3, 5, 5, 6, 8, 8, 8, 8, 10, 11, 11, 13]
The No of recursive calls for RANDOMIZED_QUICKSORT:  32
```

**********RANDOMIZED_QUICKSORT()`*********

```
import random
```

```
random.seed()

def swap(arrayToSort, firstElement, secondElement):
    tempVariable = arrayToSort[firstElement]
    arrayToSort[firstElement] = arrayToSort[secondElement]
    arrayToSort[secondElement] = tempVariable

def partition(arrayToSort, startPointer, endPointer):
    pivot = random.randint(startPointer, endPointer)
    mark = startPointer
    swap(arrayToSort, pivot, endPointer)

    for i in range(startPointer, endPointer):
        if arrayToSort[i] <= arrayToSort[endPointer]:
            quicksortPivotAsLastElement.y += 1
            swap(arrayToSort, i, mark)
            mark += 1
    swap(arrayToSort, mark, endPointer)

    return mark

def do_quicksortPivotAsLastElement(arrayToSort, startPointer, endPointer):

    if startPointer < endPointer:
        pivot = partition(arrayToSort, startPointer, endPointer)
        do_quicksortPivotAsLastElement(arrayToSort, startPointer, pivot - 1)
        do_quicksortPivotAsLastElement(arrayToSort, pivot + 1, endPointer)

def quicksortPivotAsLastElement(arrayToSort):
```

```
    do_quicksortPivotAsLastElement(arrayToSort, 0, len(arrayToSort) - 1)


if __name__ == "__main__":
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quicksortPivotAsLastElement.y = 0
    quicksortPivotAsLastElement(arrayToSort)


    print(arrayToSort)
    print("The No of recursive calls for RANDOMIZED_QUICKSORT``: ", quicksortPivotAsLastElement.y)
```

In [134]:
```
import random
random.seed()

def swap(arrayToSort, firstElement, secondElement):
    tempVariable = arrayToSort[firstElement]
    arrayToSort[firstElement] = arrayToSort[secondElement]
    arrayToSort[secondElement] = tempVariable

def partition(arrayToSort, startPointer, endPointer):
    pivot = random.randint(startPointer, endPointer)
    mark = startPointer
    swap(arrayToSort, pivot, endPointer)

    for i in range(startPointer, endPointer):
        if arrayToSort[i] <= arrayToSort[endPointer]:
            quicksortPivotAsLastElement.y += 1
            swap(arrayToSort, i, mark)
            mark += 1
    swap(arrayToSort, mark, endPointer)

    return mark

def do_quicksortPivotAsLastElement(arrayToSort, startPointer, endPointer):

    if startPointer < endPointer:
        pivot = partition(arrayToSort, startPointer, endPointer)
        do_quicksortPivotAsLastElement(arrayToSort, startPointer, pivot - 1)
        do_quicksortPivotAsLastElement(arrayToSort, pivot + 1, endPointer)

def quicksortPivotAsLastElement(arrayToSort):
    do_quicksortPivotAsLastElement(arrayToSort, 0, len(arrayToSort) - 1)

if __name__ == "__main__":
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quicksortPivotAsLastElement.y = 0
    quicksortPivotAsLastElement(arrayToSort)

    print(arrayToSort)
    print("The No of recursive calls for RANDOMIZED_QUICKSORT``: ", quicksortPivotAsLastElement.y)
```

```
[2, 3, 5, 5, 6, 8, 8, 8, 8, 10, 11, 11, 13]
The No of recursive calls for RANDOMIZED_QUICKSORT``:  15
```

7-4)

d)

```
y_tailNormalQuickSort = []
y_optimizedTailQuickSort = []

def partition(arrayToSort, startPointer, endPointer):

 pivot = arrayToSort[endPointer]
 i = startPointer - 1

 for j in range(startPointer, endPointer):
  if arrayToSort[j] <= pivot:
   i = i + 1
   (arrayToSort[i], arrayToSort[j]) = (arrayToSort[j], arrayToSort[i])
 (arrayToSort[i + 1], arrayToSort[endPointer]) = (arrayToSort[endPointer], arrayToSort[i + 1])
 return i + 1

def quickSortNormal(arrayToSort, startPointer, endPointer):
  while (startPointer < endPointer):
     quickSort.x += 1
     pi = partition(arrayToSort, startPointer, endPointer)
     y_tailNormalQuickSort.append(0)
     quickSort(arrayToSort, startPointer, pi - 1)
     y_tailNormalQuickSort.append(1)
     low = pi+1

def quickSort(arrayToSort, startPointer, endPointer):
  while (startPointer < endPointer):
     pi = partition(arrayToSort, startPointer, endPointer);
```

```
        if (pi - startPointer < endPointer - pi):
            quickSort.x += 1
            y_optimizedTailQuickSort.append(0)
            quickSort(arrayToSort, startPointer, pi - 1);
            y_optimizedTailQuickSort.append(0)
            startPointer = pi + 1;
        else:
            quickSort.x += 1
            y_optimizedTailQuickSort.append(0)
            quickSort(arrayToSort, pi + 1, endPointer);
            y_optimizedTailQuickSort.append(1)
            endPointer = pi - 1;

if __name__ == '__main__':
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quickSort.x  = 0
    data = quickSort(arrayToSort, 0, len(arrayToSort) - 1)
    print(arrayToSort)
    tailNormalQuickSort = quickSort.x
    print(tailNormalQuickSort)

    print(".............Normal Quick Sort........")
    arrayToSort2 = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    data = quickSortNormal(arrayToSort2, 0, len(arrayToSort2) - 1)
    print(arrayToSort2)
    optimizedTailQuickSort = quickSort.x
    print(optimizedTailQuickSort)

    x_pointsNormal = []
```

x_pointsOptimesed = []

for i in range(0, len(y_tailNormalQuickSort)):

    x_pointsNormal.append(i)

for i in range(0, len(y_optimizedTailQuickSort)):

    x_pointsOptimesed.append(i)

```
y_tailNormalQuickSort = []
y_optimizedTailQuickSort = []

def partition(arrayToSort, startPointer, endPointer):

  pivot = arrayToSort[endPointer]
  i = startPointer - 1

  for j in range(startPointer, endPointer):
    if arrayToSort[j] <= pivot:
      i = i + 1
      (arrayToSort[i], arrayToSort[j]) = (arrayToSort[j], arrayToSort[i])
  (arrayToSort[i + 1], arrayToSort[endPointer]) = (arrayToSort[endPointer], arrayToSort[i + 1])
  return i + 1

def quickSortNormal(arrayToSort, startPointer, endPointer):
    while (startPointer < endPointer):
        quickSort.x += 1
        pi = partition(arrayToSort, startPointer, endPointer)
        y_tailNormalQuickSort.append(0)
        quickSort(arrayToSort, startPointer, pi - 1)
        y_tailNormalQuickSort.append(1)
        low = pi+1

def quickSort(arrayToSort, startPointer, endPointer):
    while (startPointer < endPointer):
        pi = partition(arrayToSort, startPointer, endPointer);
        if (pi - startPointer < endPointer - pi):
            quickSort.x += 1
            y_optimizedTailQuickSort.append(0)
            quickSort(arrayToSort, startPointer, pi - 1);
            y_optimizedTailQuickSort.append(0)
            startPointer = pi + 1;
        else:
            quickSort.x += 1
            y_optimizedTailQuickSort.append(0)
            quickSort(arrayToSort, pi + 1, endPointer);
            y_optimizedTailQuickSort.append(1)
            endPointer = pi - 1;

if __name__ == '__main__':
    arrayToSort = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quickSort.x  = 0
    data = quickSort(arrayToSort, 0, len(arrayToSort) - 1)
    print(arrayToSort)
```

```
if __name__ == '__main__':

    print("...........  Optimized_TAIL-RECURSIVE-QUICKSORT ........")
    array = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    quickSort.x  = 0
    data = quickSort(array, 0, len(array) - 1)
    print(array)
    tailNormalQuickSort = quickSort.x
    print("The No of recursive calls for Optimized_TAIL-RECURSIVE-QUICKSORT", tailNormalQuickSort)
    print("\n")

    print("............. TAIL-RECURSIVE-QUICKSORT ........")
    array2 = [5, 6, 8, 10, 11, 13, 8, 8, 3, 5, 2, 11, 8]
    data = quickSortNormal(array2, 0, len(array2) - 1)
    print(array2)
    optimizedTailQuickSort = quickSort.x
    print("The No of recursive calls for TAIL-RECURSIVE-QUICKSORT: ", optimizedTailQuickSort)

    x_pointsNormal = []
    x_pointsOptimesed = []
    for i in range(0, len(y_tailNormalQuickSort)):
        x_pointsNormal.append(i)

    for i in range(0, len(y_optimizedTailQuickSort)):
        x_pointsOptimesed.append(i)
```

```
...........  Optimized_TAIL-RECURSIVE-QUICKSORT ........
[2, 3, 5, 5, 6, 8, 8, 8, 8, 10, 11, 11, 13]
The No of recursive calls for Optimized_TAIL-RECURSIVE-QUICKSORT 10


............. TAIL-RECURSIVE-QUICKSORT ........
[2, 3, 5, 5, 6, 8, 8, 8, 8, 10, 11, 11, 13]
The No of recursive calls for TAIL-RECURSIVE-QUICKSORT:  20
```

……..Ploting……..

```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (15,10)

plt.plot(x_pointsNormal, y_tailNormalQuickSort, 'g', label='Insertion Sort')

plt.plot(x_pointsOptimesed, y_optimizedTailQuickSort, 'b', label='Merge Sort')

plt.title('Length of list vs Execution Time')

plt.xlabel('Length of list')

plt.ylabel('Execution Time')

plt.legend()
```

plt.show()

In [9]:
```
import matplotlib.pyplot as plt


plt.rcParams["figure.figsize"] = (10,5)
plt.plot(x_pointsNormal, y_tailNormalQuickSort, 'r', label='Tail Recursive Quick Sort')
plt.plot(x_pointsOptimesed, y_optimizedTailQuickSort, 'b', label='Optimized_TAIL-RECURSIVE-QUICKSORT')
plt.title('List of Stack push & Pop vs Number of Recursions')
plt.xlabel('List of Stack push & Pop ')
plt.ylabel('Number of Recursions')
plt.legend()
plt.show()
```