ADS Homework 7

Name - Tewodros Adane
Email - tadane@jacobs-university.de

Problem 7.1 – Quicksort with Partition Versions

a) The implementation of Lomuto Quicksort in Python is shown below:

```
def lomutoQuickSort(array, p, r):
    if p < r:
        q = lomutoPartition(array, p, r)
        lomutoQuickSort(array, p, q)
        lomutoQuickSort(array, q + 1, r)

def lomutoPartition(array, p, r):
    pivot = array[p]
    i = p
    for j in range(p + 1, r + 1):
        if array[j] < pivot:
            i += 1
            array[i], array[j] = array[j], array[i]
    array[p], array[i] = array[i], array[p]
    return i
```

b) The implementation of Hoare Quicksort in Python is shown below:

```
def hoareQuickSort(array, p, r):
    if p < r:
        q = hoarePartition(array, p, r)
        hoareQuickSort(array, p, q)
        hoareQuickSort(array, q + 1, r)

def hoarePartition(array, p, r):
    pivot = array[(p + r) // 2]
    i = p - 1
    j = r + 1
    while True:
        i += 1
        while array[i] < pivot:
            i += 1
        j -= 1
        while array[j] > pivot:
            j -= 1
        if i >= j:
            return j
        array[i], array[j] = array[j], array[i]
```

c) The implementation of Median of three Quicksort in Python is shown below:

```
def medianOf3QuickSort(array, p, r):
    if p < r:
        q = medianOf3Partition(array, p, r)
        medianOf3QuickSort(array, p, q)
        medianOf3QuickSort(array, q + 1, r)

def medianOf3Partition(array, p, r):
    mid = (p + r) // 2

    if array[p] > array[r]:
        array[p], array[r] = array[r], array[p]
    if array[p] > array[mid]:
        array[p], array[mid] = array[mid], array[p]
    if array[mid] > array[r]:
        array[mid], array[r] = array[r], array[mid]

    array[mid], array[r - 1] = array[r - 1], array[mid]

    pivot = array[r - 1]
    i = p - 1
    j = r + 1

    while True:
        i += 1
        while array[i] < pivot:
            i += 1
        j -= 1
        while array[j] > pivot:
            j -= 1
        if i >= j:
            return j
        array[i], array[j] = array[j], array[i]
```

d) The average time taken by the different Quicksort algorithms is the following:

Lomuto : 0.002078980131149292

Hoare : 0.001806887602806091

Median : 0.002180302560329437

After running each algorithm 100,000 times, we can see that Hoare's algorithm is the fastest of all. This is because it performs less swaps than Lomuto's Quicksort by approaching the array from both sides instead of iterating from start to end. Theoretically, median of three quicksort should perform better than all because as Hoare's algorithm it approaches from both sides, but it can also choose a better pivot than Hoare's algorithm by comparing three potential indices. However, on my program it has been slowest, I suspect this is because when choosing the pivot, the program is making three additional comparisons to place the left, mid, and right elements in the right order.

Problem 7.2 – Modified Quicksort

a) The implementation of a dual-pivot Quicksort in Python is shown below:

```
def quickSortDualPivot(array, p, r):
    if p < r:
        a, b = dualPartition(array, p, r)
        quickSortDualPivot(array, p, a)
        quickSortDualPivot(array, a + 1, b - 1)
        quickSortDualPivot(array, b + 1, r)

def dualPartition(array, p, r):
    if array[p] > array[p + 1]:
        array[p], array[p + 1] = array[p + 1], array[p]
    array[p + 1], array[r] = array[r], array[p + 1]
    leftPivot = array[p]
    rightPivot = array[r]

    i = p + 1
    j = r - 1
    k = i

    while k <= j:
        if array[k] <= leftPivot:
            array[k], array[i] = array[i], array[k]
            i += 1
        elif array[k] >= rightPivot:
            while array[j] > rightPivot and k < j:
                j -= 1
            array[k], array[j] = array[j], array[k]
            j -= 1
            if array[k] < leftPivot:
                array[k], array[i] = array[i], array[k]
                i += 1
        k += 1
    i -= 1
    j += 1

    array[p], array[i] = array[i], array[p]
    array[r], array[j] = array[j], array[r]
    return i, j
```

b) Best-case: For a Quicksort algorithm with two pivots the best-case is when the left pivot is at index one-third of the array length and the right pivot is at two-thirds of the array length. This way the three subarrays would be partitioned with the same number of elements each. Therefore, the time complexity can be determined by:

$$T(n) = 3T(n/3) + \Theta(n)$$

Using master method:

$$a = 3, b = 3, \ \log_3 3 = 1$$

$$f(n) = \Theta(n)$$

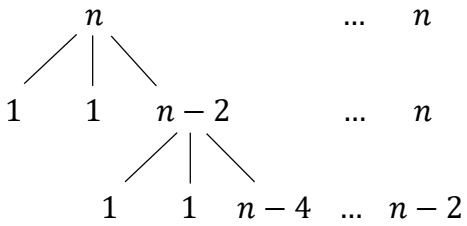
$$T(n) = \Theta\left(n^{\log_3 3}\right) = f(n), \ case\ 2\ of\ master\ method$$

$$Therefore, \ T(n) \in \Theta(n \log n)$$

Worst-case: For a Quicksort algorithm with two pivots the worst-case is when the left pivot is the smallest element, and the right pivot is the largest element. Thus, the middle subarray would have all the elements and the left and right subarrays would be empty. This usually happens when the array is increasingly or decreasingly sorted. The time complexity can be determined by:

$$T(n) = T(n-2) + T(1) + T(1) + \Theta(n)$$

Using recurrence tree method:



$$T(n) = n + n + (n-2) + \cdots + 2 + 0 = n(n+1), \ sum\ of\ the\ first\ n\ even\ numbers$$

$$T(n) = O(n^2 + n) = O(n^2)$$

c) The implementation of a Randomized dual-pivot Quicksort in Python is shown below:

```
def quickSortRandomized(array, p, r):
    if p < r:
        a, b = randomDualPartition(array, p, r)
        quickSortRandomized(array, p, a)
        quickSortRandomized(array, a + 1, b - 1)
        quickSortRandomized(array, b + 1, r)

def randomDualPartition(array, p, r):
    left = randrange(r - p) + p
    right = randrange(r - p) + p
    array[p], array[left] = array[left], array[p]
    array[r], array[right] = array[right], array[r]

    if array[p] > array[r]:
        array[p], array[r] = array[r], array[p]

    leftPivot = array[p]
    rightPivot = array[r]

    i = p + 1
    j = r - 1
    k = i

    while k <= j:
        if array[k] <= leftPivot:
            array[k], array[i] = array[i], array[k]
            i += 1
        elif array[k] >= rightPivot:
            while array[j] > rightPivot and k < j:
                j -= 1
            array[k], array[j] = array[j], array[k]
            j -= 1
            if array[k] < leftPivot:
                array[k], array[i] = array[i], array[k]
                i += 1
        k += 1
    i -= 1
    j += 1

    array[p], array[i] = array[i], array[p]
    array[r], array[j] = array[j], array[r]
    return i, j
```

Source:

- Wikipedia: https://en.wikipedia.org/wiki/Quicksort
- GeeksForGeeks: https://www.geeksforgeeks.org/dual-pivot-quicksort