Sean Morrissey

Professor Ramoza Ahsan

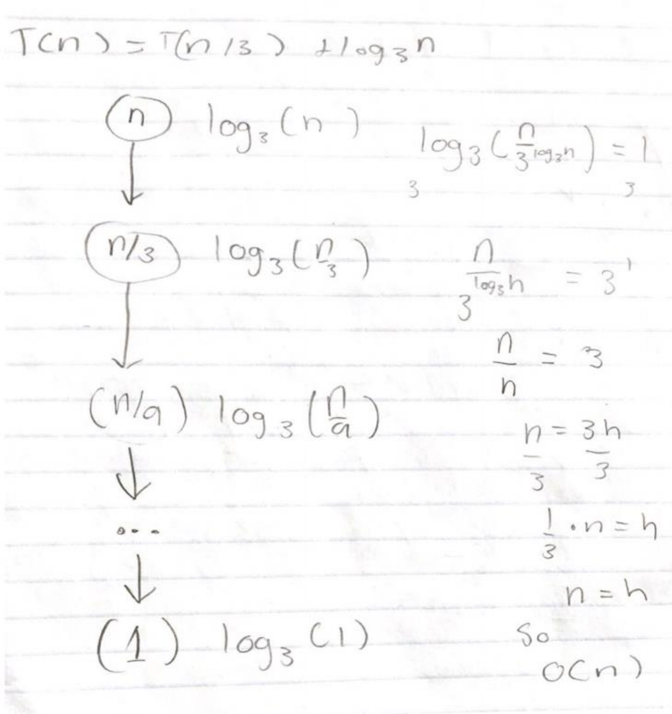
CS 2223 Algorithms

1 April 2019

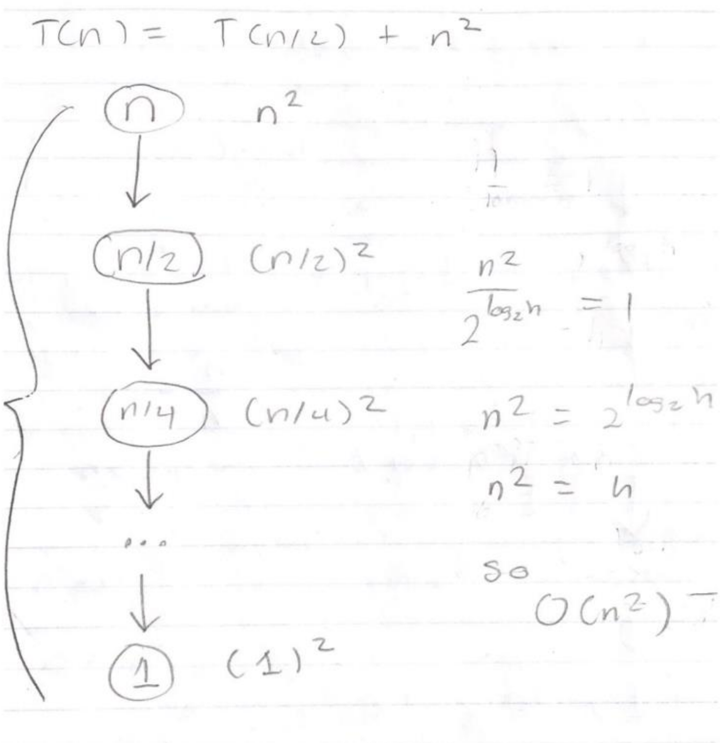
Assignment 2

Question 1.

1. T(n) = 3T(3/5 n) + n
   - A =3, B =5/3, α = 1, β =$log_{5/3}3$
   - O($n^{log_{5/3}3}$)
2. T(n) = 3T(n/4) + n
   - A =3, B =4, α = 1, β =$log_4 3$
   - O(n)
3. T(n) = 7T(n/2) + n^2
   - A =7, B =2, α = 2, β =$log_2 7$
   - O($n^{log_2 7}$)
4. T(n) = T(3/4n) + 3 n^2 + n
   - A =1, B =4/3, α = 2, β =$log_{4/3}1$
   - O(n^2)
5. T(n) = T(n/3) + $log_3 n$



6. $T(n) = T(n/2) + n^2$



Question 2.

```
#Question 2 Morrissey


def isreverse(str1, str2):
    return str2 == reverse(str1)


def reverse(string):
    if len(string) == 0:
        return string
    else:
        return reverse(string[1:]) + string[0]

def main():
    i=0
    k=0
    print("Please enter a string: ")
    givenStr = input();
    print("Please enter another string: ")
    givenStr2 = input();
    print(isreverse(givenStr, givenStr2))

main()
```

The function call to the helper function reverse() leads into creating the recurrence of the function isreversed(). In reverse(), there is a call to len(), which in python, is constant time; therefore, it has no effect on the recurrence time. In the recursive call to reverse(), I am calling it on a string that is of size n-1, if n is the size of the string passed into to reverse(). Thus, every call the string is shorter by n-1 times which means in the recurrence function T(n) is equal to T(n-1).

With that, the recurrence for the function is…

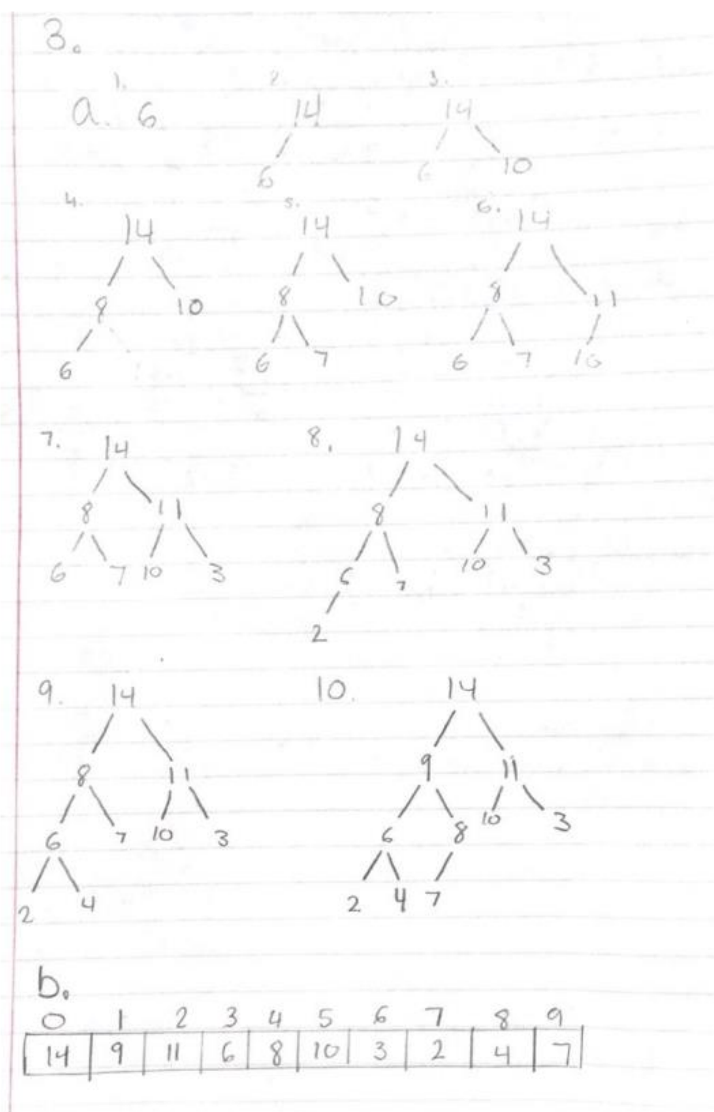
$T(n) = T(n-1) + c$

Or

$T(n) = T(n-1) + O(1)$

As a result, the big O notation of the function is O(n).

Question 3.



Question 4.

1. True. The recurrence function of T(n) = 3T(n/2) + n does imply O($n^{log_2 3}$) because if one were to solve for the big O runtime with the Master Theorem, it will be O($n^{log_2 3}$).
    - A = 3, B = 2, α = 1, β = $log_2 3$; Case 3 applies
2. False. The recurrence equation for merge sort is T(n) 2T(n/2) + n. By using the Master Theorem, we know that the big O runtime is O(n log n). Because $n^2 logn$ is a runtime that is greater than $nlogn$, it is impossible for Ω($n^2 logn$) to be true for merge sort.
3. False. If we expand the given recurrence function, we end up getting something that looks like the following:
    a. T(n-1) + log(n)
    b. T(n-2) + log(n-1) + log(n)
    c. T(n-3) + log(n-2) + log(n-1) + log(n)

    If we assume n = 0, then

    T(0) + log(1) + log(2) + log(3) … log(n-1) + log(n)

    Which is the same as…

    T(0) + log(n!)

Therefore, for the given recurrence function, the runtime is Θ(log(n!)), but this does not work for quick sort. The worst case for quick sort is O(n^2), which is when a sorted array is passed in. Thus, the statement is false because the runtime of the recurrence is not the same as quick sort's worst case.

Question 5.

```
#Question 5 Morrissey

def inversionCount(alist):
    count = 0
    leftcount = 0
    rightcount = 0
    newlist = []
    if len(alist) > 1:
       mid = len(alist) // 2
       lefthalf = alist[:mid]
       righthalf = alist[mid:]
       leftcount, lefthalf = inversionCount(lefthalf)
       rightcount, righthalf = inversionCount(righthalf)

       i = 0
       j = 0

       while i < len(lefthalf) and j < len(righthalf):
         if lefthalf[i] < righthalf[j]:
             newlist.append(lefthalf[i])
             i += 1
         else:
             newlist.append(righthalf[j])
             j += 1
             count += len(lefthalf[i:])

       while i < len(lefthalf):
          newlist.append(lefthalf[i])
          i += 1

       while j < len(righthalf):
          newlist.append(righthalf[j])
          j += 1
    else:
        newlist = alist

    return count + leftcount + rightcount, newlist

def main():
    initlist = [4,1]
    inversions, finalList = inversionCount(initlist)
    print("Unsorted List: ", initlist)
    print("Sorted List: ",finalList)
    print("The amount of inversions in this list are: ", inversions)
main()
```

Because the function inversionCount() function is basically merge sort with a counter, the runtime and recurrence function are practically the same. Using a counter and modifying it are constant operations, so they do not have an affect on the recurrence function of merge sort because n has greater growth than a constant, O(1).

Therefore, the recurrence function of inversionCount() is…

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

Using the Master Theorem to find the big O runtime in the worst case…

A= 2, B = 2, α = 1, β = $log_2 2$ = 1

Case 2 -> Θ(n log n)

Thus, the runtime in the worst case for inversionCount() is O(n log n).