Rafid
Shaon
CSC 4520
Sep 29 2022

HW2: Runtime and QuickSort

Instructions

The goal of this homework is to practice what we covered in Lecture **This assignment is due on SEP 29, 2022 at 11:59PM Eastern Time.**

Most of this homework does not require coding, but instead, expects you to analyze existing code. Please read each question carefully. **We've provided an example question and ideal answer in example.txt to help you get started on the runtime analysis questions.**

Submitting This Assignment

iCollege

Grading and Corrections

Since this is a written assignment, There are no tests or examples.

This assignment will be graded out of 10 points. The point values of each problem are listed in the title. We will use manual inspection and written rubrics to assign points in a fair, standardized way.

Academic Integrity

Remember that you can consult outside resources and work with other students as long as you write up your own solutions and cite any links or people you received help from within citations.txt.

Q1 Mysterious Function (30 point)

What's the worst case runtime of the following function? Please remember to define n, provide a tight upper bound.

```
public static void mystery1(int z) {
   System.out.println(z);
   if (z >= 10) {
     mystery1(z/10);
     System.out.println(z);
   }
 }
```

Q1: Mysterious Function

Worst Case Runtime = $O(log_{10} n)$ or O(log base 10 n)

n is defined as integer z. The function is recursively calling z/10 every time and returning some value when true, and returning the function on base condition when false.

Using Recurrence Relation the function is T(n) = T(n/10) + k
a = 1; b = 10; k = 0
1 = (10^0) → 1 = 1
Using the Master Theorem, a = b^k → T(N) $\in n^k(log_b n)$
T(N) = $n^0(log_{10} n) \rightarrow 1(log_{10} n) \rightarrow O(log_{10} n)$

The tight upper bound is also $O(log_{10} n)$.

Q2 Exponentiation (Fast?) (40 points)

- What's the best case, worst case, and average-case runtime of pow? **Assume n = power**. Please remember to define n, provide a tight upper bound.

```
algorithm pow
  Input: positive integer b, non-negative integer p
  Output: computing b^p (i.e. b raised to power of p)

  if p = 0
    return 1
  else if p = 1
    return b
  else if p is even
    temp = pow(b, p / 2)
    return temp * temp
  else
    return b * b * pow(b, p-2)
```

Q2: Exponentiation (Fast?)

assuming that n = power

Best Case: O(1)
This is when p = 0 or p = 1.

Average Case: O(log(n))

Worst Case: O(n)
This is when p is even, making the recursive relation $T(n) = T(n - 2) + O(1) \rightarrow O(n)$.

Q3 QuickSort (30 point)

Given the QuickSort implementation from class, provide an 18-element list that will take the **least** number of recursive calls of QuickSort to complete.

As a **counter-example**, here is a list that will cause QuickSort to make the **MOST** number of recursive calls:

```
public static List<Integer> input() {
   return Arrays.asList(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
 }
```

And here's the QuickSort algorithm, for convenience:

```
algorithm QuickSort
  Input: lists of integers lst of size N
  Output: new list with the elements of lst in sorted order

  if N < 2
    return lst
  pivot  =  lst[N-1]
  left = new empty list
  right = new empty list
  for index i = 0, 1, 2, ... N-2
    if lst[i] <= pivot
      left.add(lst[i])
    else
      right.add(lst[i])
  return QuickSort(left) + [pivot] + QuickSort(right)
```

Q3: QuickSort

least number of recursive calls = best case scenario
best case is when pivot is in the middle

pivot = list(n-1) / 2

Array Examples:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]

[1, 17, 16, 2, 5, 4, 12, 13, 9, 10, 15, 7, 6, 14, 8, 3, 11, 18]

*any 18-element list will work as long as pivot is in the middle of list