CSCI 3104 Spring 2018 Problem Set #4
Alison Ostlund (05/22)
Worked with: Kyle Ward, Matt Donovan

---

1. (10 pts) Suppose that we modify the Partition algorithm in QuickSort in such a way that on on every third level of the recursion tree it chooses the worst possible pivot, and on all other levels of the recursion tree Partition chooses the best possible pivot. Write down a recurrence relation for this version of QuickSort and give its asymptotic solution. Then, give a verbal explanation of how this Partition algorithm changes the running time of QuickSort.

   Two best cases:
   $T_1(n) \leq 2T_2(\frac{n}{2}) + O(n)$
   $T_2(n) \leq 2T_3(\frac{n}{2}) + O(n)$
   Third level- Worst Case:
   $T_3(n) \leq T_1(n-1) + O(n)$
   $T_3(n) = T_1(n-1) + O(n)$
   $= 2T_2(\frac{n-1}{2}) + O(n-1) + O(n)$
   $= 2[2T_3(\frac{\frac{n-1}{2}}{2}) + O(\frac{n-1}{2}) + O(n-1)] + O(n)$
   $= 4T_3(\frac{n-1}{4}) + 2O(\frac{n-1}{2}) + O(n-1) + O(n) =$ Using Masters Method:$T(n) = a\ T\left(\frac{n}{b}\right) + f(n)$,
   $= \log_4(4) = 1$
   $a = 4, b = 4, c = 1,$
   $f(n) = n^1 \log_4 4 = 1 = clog_b a = c \Rightarrow \Theta(n^{log_b a} \cdot \log(n)) \Rightarrow \Theta(n \log(n))$
   This version of QS uses the worst case every third level, but it doesnt change the time complexity from the orignal QS. It still allows QS to run at its best case a majority of the time, so it will run in $\Theta(n \log(n))$.

2. (10 pts) Mr. Ollivander, of Ollivanders wand shop, has hired you as his assistant, to find the most powerful wand in the store. You are given a magical scale which weighs wands by how powerful they are (the scale dips lower for the wand which is more powerful). You are given n wands W1, . . . , Wn, each having distinct levels of power (no two are exactly equal).

(a) Consider the following algorithm to find the most powerful wand:

```
i. Divide the n wands into n/2 pairs of wands.
ii. Compare each wand with its pair, and retain the more powerful of the t
iii. Repeat this process until just one wand remains.
```

```
Illustrate the comparisons that the algorithm will do for the following n =

W1:3/2; W2:5/2 ; W3:1/2 ; W4:1 ; W5:2 ; W6:5/4 ; W7:1/4 ; W8:9/4

 [3/2   5/2   1/2   1   2   5/4   1/4   9/4]
    \           /     \              /
      [5/2  1]           [5/4  9/4]
        \  /                \  /
        [5/2]               [9/4]
           \                  /
                   [5/2]
```

(b) Show that for n wands, the algorithm (2a) uses at most n comparisons.

$T(n) = T(\frac{n}{2}) + \frac{n}{2}$

$T(\frac{n}{2}) = T(\frac{n}{4}) + \frac{n}{4} + \frac{n}{2}$

$T(\frac{n}{4}) = T(\frac{n}{8}) + \frac{n}{8} + \frac{n}{4} + \frac{n}{2}$

$\Rightarrow \sum_{i=1}^{n}(\frac{n}{2^i}) = n(1-(\frac{1}{2})^i)$

$1-(\frac{1}{2})^i) \rightarrow$ goes to 0 so left with n $\therefore$ n comparisons

(c) Describe an algorithm that uses the results of (2a) to find the second most powerful wand, using at most log2 n additional comparisons. There is no need for pseudocode; just write out the steps of the algorithm like we have written in (2a). Hint: if you follow sports, especially wrestling, read about the repechage.

```
i. Retain any comparisons made with the winning max from 2a, and place thes
ii. Compare the wands in the array in pairs
iii. Grab the largest wand of the compared pair and recurse until reach the
```

(d) Show the additional comparisons that your algorithm in (2c) will perform for the input given in (2a).

```
[9/4   3/2   1]
    \    /
   [ 9/4   1]
       \  /
        [9/4]
```

3. (20 pts) For obtuse historical reasons, Prof. Dumbledore asks his students to line up in ascending order by height in a very tight room with little extra space. Similar to Alex the African Grey parrot (look it up!), the students, being bored, decided to play a little trick on Prof. Dumbledore. They lined up in order by heightalmost. They made sure that each person was no more than k positions away from where they were supposed to be (in ascending order), but this allowed them to significantly mess up the precise ordering. Here is an example of an array with this property when k = 2:

   (a) Write down pseudocode for an algorithm that would sort such an array in placeso it fits in the tight roomin time O(n k log k). Your algorithm can use a function sort(A, l, r) that sorts the subarray A[l], . . . , A[r] in place in O((r l) log(r l)) steps (assuming r > l).

```
Function(A,n,k)
   for i to (n-k)
      sort(A, i, k+i)
```

   (b) Suppose you are given to an auxiliary room which can fit k + 1 students. Modify your previous algorithm to sort the given array in time O(nk).

```
AuxRoomFunc(A,n,k)
   for( i= 1 in A to (n-k))
       for( j = 0 to k)
          B[j+1] = A[j+i]
       B[0]= findMin(B)
       shift the indices greater than min to the left
       for x= 0 to k-1
           A[i+x] = B[x]
```

(c) With the same extra room as in the previous part, modify heap sort using a binary min heap of size k + 1 to sort the given array in time O(n log k).

```
     FuncHeap(A,n,k)
         for i to (n-k)  -> n times complexity
            MinHeapSort(A,i,k+i) -> Runs in O(log(k))
Source for time of MinHeapSort: Victor S.Adamchik, CMU, 2009 https://www.cs.
121/lectures/Binary%20Heaps/heaps.html)

 So total time is O(n log k)
```

(d) (5 pts extra credit) Include the correct story about Alex, with proper citation. If you wish, you may copy this story verbatim, but must indicate clearly that you have done so and, of course, still cite your source.

Once, Alex was given several different colored blocks (two red, three blue, and four green). Pepperberg asked him, What color three? expecting him to say blue. However, as Alex had been asked this question before, he seemed to have become bored. He answered five! This kept occurring until Pepperberg said, Fine, what color five? Alex replied, None. This was said to suggest that parrots, like children, get bored. Sometimes, Alex answered the questions incorrectly, despite knowing the correct answer. He may have been bored, says Pepperberg. But what he did was to figure out how to manipulate me into asking the question he wished to answer (showing an important level of awareness)...
Source: https://blog.wikimedia.org/2016/12/07/wait-what-alex/

4. (20 pts) Consider the following strategy for choosing a pivot element for the Partition subroutine of QuickSort, applied to an array A.

```
Let n be the number of elements of the array A.
If n  24, perform an Insertion Sort of A and return.
Otherwise:
    -Choose 2^[n(1/3)] elements at random from n; let S be the new list with
```

```
-Sort the list S using Insertion Sort and use the median m of S as a piv
-Partition using m as a pivot.
-Carry out QuickSort recursively on the two parts.
```

(a) How much time does it take to sort S and find its median? Give a Θ bound.

Because we know that Insertion Sort has a worst case running time of $\Theta(n^2)$ and we are given that the set size we are dealing with is $2n^{\frac{1}{3}}$ we are able to determine the running time for the set S. This is done by simply just plugging the problem set size into the time complexity of Insertion Sort: $\Theta(2(n^{\frac{1}{3}})^2 = \Theta(4n^{\frac{2}{3}}$. The median m takes the constant operations therefore we can ignore the 4 in the complexity, over all giving us a run time of :
T(n) = $\Theta(n^{\frac{2}{3}})$.

(b) If the element m obtained as the median of S is used as the pivot, what can we say about the sizes of the two partitions of the array A?

if m the pivot of set S is found, it will be one half of the total set S which is $2n^{\frac{1}{3}}$. Therefore we will get two halves of $n^{\frac{1}{3}}$. So when m is the the pivot the smallest portion of A will be $n^{\frac{1}{3}}$ leaving the other portion of A to be $(n - n^{\frac{1}{3}})$. This would cause the worst case for partition because the portion of $n^{\frac{1}{3}}$ is much smaller than $(n - n^{\frac{1}{3}})$.

(c) Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.

When n $\leq$ 24:
$\Theta(n^2)$
Otherwise:
$T(n) = T(n^{\frac{1}{3}})) + T(n - n^{\frac{1}{3}}) + O(n^{\frac{2}{3}}) + O(n)$

5. (20 pts extra credit) Recall that the Insertion Sort algorithm (Chapter 2.1 of CLRS) is an in-place sorting algorithm that takes $\Theta(n^2)$ time and Θ(n) space. In this problem, you will learn how to instrument your code and how to perform a numerical experiment that verifies the asymptotic analysis of Insertion Sorts running time. There are two functions and one experiment to do this.

```
(i) InsertionSort(A,n) takes as input an unordered array A, of length n, and retu
both an in-place sorted version of A and a count t of the number of atomic operat
performed by InsertionSort.
Recall: atomic operations include mathematical operations like , +, , and /, ass
operations like  and =, comparison operations like <, >, and ==, and RAM
indexing or referencing operations like [ ].
(ii) randomArray(n) takes as input an integer n and returns an array A such that
each 0  i < n, A[i] is a uniformly random integer between 1 and n. (It is okay i
a random permutation of the first n positive integers; see the end of Chapter 5.
```

(a) From scratch, implement the functions InsertionSort and randomArray. You may not use any library functions that make their implementation trivial. You may use a library function that implements a pseudorandom number generator in order to implement randomArray. Submit a paragraph that explains how you instrumented InsertionSort, i.e., explain which operations you counted and why these are the correct ones to count. Hint: your instrument code should only count the operations of the InsertionSort algorithm and not the operations of the instrument code you added to it.

The instrument code only counted the operations in Insertion sort. I placed a counter by each atomic operation in Insertion sort that would increment the counter each time the algorithm ran into a code line that makes use of a atomic operation.
Note: From Lecture Notes 1 ”Atomic operations: Simple mathematical (+, , , =) and simple functional (if,call,store, etc.) operations take 1 unit of time, or O(1) time. We call these atomic operations.”
These are correct because they only tak3 the time of O(1). So based on this definition the counter was incremented anytime there was a simple mathematical operation in Insertion Sort such as j = i-1 and then the counter was incremented with any calls or store operations like key = arr[i];. This only applies to the Insertion Sort because the implemented part of randomArray does not affect these operations rather it affects the size of the array we are sorting which will affect the best and worst cases of the time complexity of the algorithm as a whole. In order to keep track of this, I implemented the counters describes above.

(b) For each of n = $2^3, 2^5, ..., 2^{15}, 2^{16}$, run InsertionSort(randomArray(n),n) fives times and record the tuple (n,t), where t is the average number of operations your function counted over the five repetitions. Use whatever software you like to make a line plot of these 12 data points; overlay on your data a function of the form T(n) = An^2, where you choose the constant A so that the function is close to your data. Hint: To increase the aesthetics, use a log-log plot

SOLUTION