Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 7b (47 points + 10 pts extra credit) **Fall 2019, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 7b (47 points + 10 pts extra credit) **Fall 2019, CU-Boulder**

Important: This assignment has two (Q2, Q3) coding questions.

- You need to submit two python files, one for each question.
- The .py file should run for you to get points and name the file as following -
  If Q2 asks for a python code, please submit it with the following naming convention - `Lastname-Firstname-PS7b-Q2.py`.
- You need to submit the code via Canvas but the table/plot/result should be on the main .pdf.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 7b (47 points + 10 pts extra credit) **Fall 2019, CU-Boulder**

1. (7 pts) Suppose that we modify the `Partition` algorithm in QuickSort in such a way that on alternating levels of the recursion tree, `Partition` either chooses the best possible pivot or the worst possible pivot.

   (a) (1 pt) What are the best possible and the worst possible pivots for Quicksort?

   Solution.
   The best pivot splits the array directly in half (n/2). The worst case does not actually split the array, instead just sort of puts the value at one end and the entire array is either to the left or right of it (n-1)

   (b) (4 pts) Write down a recurrence relation for this version of QuickSort and give its asymptotic solution.

   Solution.
   T(n)=T($\frac{n}{2}$)+T(n-1)+Θ(n)
   (Given based on the idea one goes left, the other case goes right)

   (c) (2 pts) Provide a verbal explanation of how this `Partition` algorithm affects the running time of QuickSort.

   Solution.
   The tree for the algorithm splits the left end to $\frac{n}{2}$ and the right to n-1 for each level. This ultimately creates an uneven area on each side overall, kind of like those wind chimes with a small length on the left and the long length on the right. This means the left length, through level log n, is cn while all points past then are less than or equal to cn. The right side still goes as far as n^2, combining to a θ(n n^2). This becomes $\Theta(n^2)$, thus making the partition just as bad as the worst case.

Name: Luna McBride
ID: 107607144

2. *(14 pts total) In PS1b, you were asked to count flips in a sorting algorithm with quadratic running time. The problem definition looked something like this:*
Let $A = \langle a_1, a_2, \ldots, a_n \rangle$ be an array of numbers. Let's define a 'flip' as a pair of distinct indices $i, j \in \{1, 2, \ldots, n\}$ such that $i < j$ but $a_i > a_j$. That is, a_i and a_j are out of order.
For example - In the array A = [1, 3, 5, 2, 4, 6], (3, 2), (5, 2) and (5, 4) are the only flips i.e. the total number of flips is 3. (Note that in this example the indices are the same as the actual values)

   (a) (14 pts) Write a Python program with the following features:

      i. (2 pts) Generates a sequence of n numbers in the range $[1, \ldots, n]$ and then randomly shuffles them.

      ii. (2 pts) Implements a $\theta(n^2)$ sorting routine that counts the number of flips in the array.

      iii. (5 pts) Implements a sorting routine with $\theta(nlgn)$ running time that counts the number of flips in the array. **Hint: Mergesort**

      iv. (5 pts) Run your code, both sorting algorithms, on values of n from $[2, 2^2, 2^3, ...2^{12}]$ and present your results in a table or labeled plot. Result with no supporting code will not get points.

   Follow the naming convention for python code mentioned on Page 2.

Solution.

N	Flips	MergeFlips
2	1	1
4	4	4
8	15	15
16	54	54
32	269	269
64	1135	1135
128	3994	3994
256	16442	16442
512	69638	69638
1024	258313	258313
2048	1063776	1063776
4096	4200699	4200699

Name: Luna McBride
ID: 107607144

3. (10 pts) Help the Mad Scientist calculate his h-index. According to Wikipedia: "A scientist has index h if h of their N papers have at least h citations each, and the other $N - h$ papers have no more than h citations each."

   For this question, write a Python program that calculates the h-index for a given input array. The array contains the number of citations for N papers, sorted in descending order (each citation is a non-negative integer). Your Python program needs to implement a divide and conquer algorithm that takes the *citations* array as input to outputs the h-index.

   Example:
   Input: citations = [6,5,3,1,0]
   Output: 3
   Explanation: [6,5,3,1,0] means the researcher has 5 papers with 6, 5, 3, 1, 0 citations respectively. Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, the h-index is 3.
   Note: If there are several possible values for h, the maximum value is the h-index.
   Hint: Think how will you find it by a linear scan? You can then make your "search" more efficient.

   Do not submit anything on the .pdf for this question.
   Follow the naming convention for the python code mentioned on Page 2.

Name: Luna McBride
ID: 107607144

4. (16 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 $n \leq 15$, perform an Insertion Sort of A and return.
   - Otherwise:
     - Choose $2\lfloor\sqrt{n}\rfloor$ elements at random from A; let S be the new list with the chosen elements.
     - Sort the list S using Insertion Sort and store the median of S as m.
     - Partition the sub-array of A using m as a pivot.
     - Carry out QuickSort recursively on the two parts.

   (a) (4 pts) Using the following array A with $n = 20$, show one iteration of this partitioning strategy on the array

   $$A = [34, 45, 32, 1, 23, 90, 12, 13, 43, 54, 65, 76, 67, 56, 45, 34, 44, 55, 23, 2]$$

   . Clearly identify all variables.

   Solution.
   Elements=2 $\lfloor(\sqrt{n})\rfloor$= 2*4=8
   Random number generator given indices (Googled "Random Number Generator" and clicked generate 1-20 a couple times): 14, 6, 1, 8, 17, 4, 13, 5, as such, S=[56, 90, 34, 13, 44, 1, 67, 23]

   Sort: S=[56, 90, 34, 13, 44, 1, 67, 23] $->$ S=[56, 90, 34, 13, 44, 1, 67, 23] $->$ S=[34, 56, 90, 13, 44, 1, 67, 23] $->$ S=[13, 34, 56, 90, 44, 1, 67, 23] $->$ S=[13, 34, 44, 56, 90, 1, 67, 23] $->$ S=[1, 13, 34, 44, 56, 90, 67, 23] $->$ S=[1, 13, 34, 44, 56, 67, 90, 23] $->$ S=[1, 13, 23, 34, 44, 56, 67, 90]

   m=$\frac{34+44}{2}$=$\frac{78}{2}$=39. 39 does not exist in the array, so we will take the 4th index for the middle and go with 34

   swap(34,2) for convenience sake. A = [2, 45, 32, 1, 23, 90, 12, 13, 43, 54, 65, 76, 67, 56, 45, 34, 44, 55, 23, 34]

2≤34? yes. Swap (2,2). 45≤34? no. 32≤34? yes. swap(32,45). 1≤34? yes. swap(1,45). 23≤34? yes. swap(23,45). 90≤34? no. 12≤34? yes. swap(12,45). 13≤34? yes. swap(13,90). 43≤34? no. 54≤34? no. 65≤34? no. 76≤34? no. 67≤34? no. 56≤34? no. 45≤34? no. 34≤ 34? yes. swap(34,45). 44≤34? no. 55≤34? no. 23≤34? yes. swap(23,90). End, so swap(34,43)
A = [2, 32, 1, 23, 12, 13, 34, 23, 34, 54, 65, 76, 67, 56, 45, 45, 44, 55, 90, 43]

Pivot: A[8]. Recursive calls: QuickSort(A,0,7), QuickSort(A,9,19)

(b) (4 pts) 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? **Hint: Think about the best and worst possible selections for the values in S.**

Solution.
This strategy uses a sample set of the data to get an approximate median. Medians are used to get the center of data, and as such, this partition function is trying its best to hit the center of the array (without using all the array to check because that is slow). In finding the median of a subset, this should give a representation of the median of the overall set, giving approximately half to each partition. Even in the worse case (say, the highest 8 out of 20 are randomly chosen), there are still at least $\sqrt{n}$ values between this value and the normal worst case (n-1). Thus, at the very least, the sizes of each partition are more centralized, ensuring at least the average case in worst situations and making quicksort $\theta(nlog(n))$

(c) (3 pts) How much time does it take to sort S and find its median? Give a Θ bound.

Solution.
Finding the median is pretty simple (in constant time), so less focus is given to that. Insertion Sort, however, has a typical bound of $\theta(k^2)$ (as there is two loops needed to make sure the insertion work correctly). However, since this is only working on $2\sqrt{n}$ values, we should plug that into the equation so it is specifically for our tested values.
$\theta((2\sqrt{n})^2) -> \theta(4n) -> \theta(n)$

Therefore, in terms of our values n, $\theta(n)$

(d) (5 pts) Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.

Solution.
Worst case for regular QuickSort: n-1. Worst case for the insertion given above: n

T(n)=T(n-1)+$\theta(n)$
T(n)=[T(n-2)+(n-1)]+n $->$ T(n-2)+2n-1
T(n)=[T(n-3)+(n-2)]+2n-1 $->$ T(n-3)+3n-3
T(n)=[T(n-4)+(n-3)]+3n-3 $->$ T(n-4)+4n-6
T(n)=T(n-k)+kn-$\sum_{i=0}^{k-1} i$
T(n)=T(n-k)+kn-$\frac{(k-1)k}{2}$
n-k=0 $->$ k=n
T(n)=T(0)+n^2-$\frac{(n-1)n}{2}$
T(n)=T(0)+n^2-$\frac{1}{2}n^2 + \frac{1}{2}n$
T(n)=1+$\frac{1}{2}n^2+\frac{1}{2}n$
$\theta(n^2)$

5. (10 pts extra credit) Implement the bottles and lids algorithm that you wrote in assignment 7a and show that it functions correctly on randomly generated arrays representing 100 bottles and lids. Your algorithm needs to use a divide and conquer strategy to receive credit for this question.