Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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.
   Best possible pivot: choosing the middle index in the array
   Worst possible pivot: choosing either the first or last index in the array

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

   Solution.
   $T_1(n) = 2T_2(\frac{n}{2}) + \Theta(n)$
   $T_2(n) = T_1(n-1) + T_1(0) + \Theta(n)$
   plug T_2 into T_1 to get general solution:
   $T_1(n) = 2(T_1(\frac{n}{2} - 1) + T_1(0) + \Theta(\frac{n}{2})) + \Theta(n)$
   $T_1(n) = 2T_1(\frac{n}{2} - 1) + 2T_1(0) + 2\Theta(\frac{n}{2}) + \Theta(n)$
   $T_1(n) = 2T_1(\frac{n}{2} - 1) + \Theta(n)$
   Rhonda specified in office hours that we can drop the '-1' in the recursion size, as it is insignificant as n grows larger and larger.
   Therefore, you are left with the following recurrence relation:
   $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
   Apply master's theorem:
   a=2, b=2, c=1, $log_b a = 1$
   $c = log_b a$, so $T(n) \in \Theta(nlogn)$

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

   Solution.
   This version of the partition algorithm switches between choosing the best and worst possible pivots, causing the running time of QuickSort to converge to an 'average' case (somewhere between the best and worst cases). This also shows us that the average case of partition (and thus the average case of QuickSort) tends to be closer to the best case (nlogn) than the worst case (n^2).

Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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

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 $[2, 2^2, 2^3, \ldots.2^{12}]$ 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.

Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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

Solution.

n	$\Theta(n^2)$ flip counter	$\Theta(nlogn)$ flip counter
2	0	0
2^2	5	5
2^3	11	11
2^4	63	63
2^5	255	255
2^6	967	967
2^7	4454	4454
2^8	16832	16832
2^9	66047	66047
2^{10}	263029	263029
2^{11}	1037164	1037164
2^{12}	4201855	4201855

Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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

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: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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

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.
   n=20
   n>15, so choose $2\lfloor\sqrt{20}\rfloor = 8$ elements at random from A
   $S = [34, 45, 23, 13, 54, 76, 45, 34]$ (values chosen by using a random number generator to pick the indices)
   Sorted $S = [13, 23, 34, 34, 45, 45, 54, 76]$, $m = \frac{34+45}{2} = 39.5$
   $A_1 = [34, 32, 1, 23, 12, 13, 34, 23, 2]$
   $A_2 = [45, 90, 43, 54, 65, 76, 67, 56, 45, 44, 55]$

Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

(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.
Best case: The median of S is as close as possible to the median of A, so that A is split into two subsets of equal size (or nearly equal if there are an odd number of elements.

Worst case: The algorithm randomly selects the $2\lfloor\sqrt{n}\rfloor$ largest or smallest elements in A, so that the median of S would be as far away from the median of A as possible, causing A_1 and A_2 to be as uneven in size as possible.

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

Solution.
$\Theta((2\sqrt{n})^2) + \Theta(1) = \Theta(4n) + \Theta(1) = \Theta(n)$

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

Solution.
Worst case: when m is such that A is split into A[0...$\lfloor\sqrt{n}\rfloor$-1] and A[$\lfloor\sqrt{n}\rfloor$...n]. There are then two sub problems with sizes $\lfloor\sqrt{n}\rfloor$ and $n - \lfloor\sqrt{n}\rfloor$, as well as the partitioning with runtime $\Theta(n)$ and the creation of the two subarrays A_1 and A_2, of runtime $\Theta(n)$. There also exists the case of when the size n of the passed array is less than or equal to 15, in which case there is just an insertion sort, which has runtime complexity of $\Theta(n^2)$.
$T(n \leq 15) = \Theta(n^2)$
$T(n) = T(\lfloor\sqrt{n}\rfloor) + T(n - \lfloor\sqrt{n}\rfloor) + \Theta(n)$

Name: Alex Book
ID: 108073300
Collaboration: Varun Narayanswamy, Jamie Foster, Irvin Carbajal, Simon Koeten

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

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.