Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 7a (14 points) **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 7a (14 points) **Fall 2019, CU-Boulder**

1. (1 pt) Provide a one-sentence description of each of the components of a divide and conquer algorithm.

   Solution.
   Divide: splits up the problem into smaller problems, which are easier to solve than the large counterpart.

   Conquer: Once small enough that smaller values do not provide much, solve the tiny problems. Otherwise, divide again.

   Combine: bring the solved small problems back to the big. If all went well, the large problem is also solved now as well.

2. (3 pts) Use the array $A = [2, 5, 1, 6, 7, 9, 3]$ for the following questions

   (a) (1 pt) What is the value of the pivot in the call *partition*$(A, 0, 6)$?
   Solution.
   This depends on the strategy we use. However, if we assume the basic strategy is used, the pivot is the last value of the array. In this case, it would be A[6], which is 3.

   (b) (1 pt) What is the index of that pivot value at the end of that call to *partition*()?
   Solution.
   (Just some quicksort in my head guided by what I typed after this to make sure I have this right)
   2< 3? yes, swap(2,2). 5<3? no. 1<3? yes, swap(5,1). 6<3? no. 7<3? no. 9<3? no. Now at 3, so swap(5,3) (5 being after the last value less than 3).

   There are 2 values less than 3, so 3 is set at A[2] with a new array A=[2,1,3,6,7,9,5], where those to the left of 3 are less than 3 and those on the right are greater than or equal to 3.

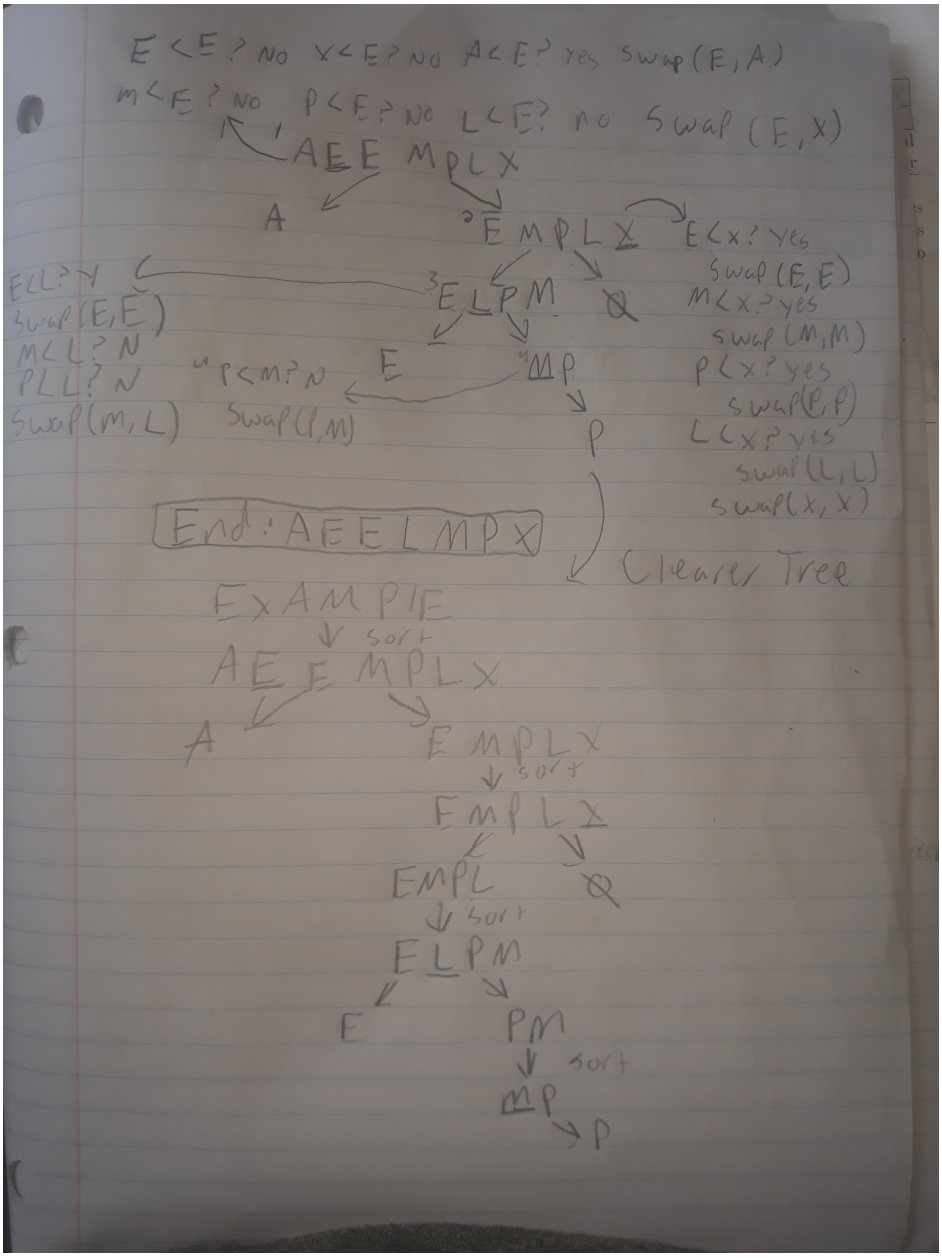
   (c) (1 pt) On the next recursive call to Quicksort, what sub-array does *partition*() evaluate?
   Solution.
   Quicksort has two recursive calls, typically being called on left of the pivot then on right (Though I guess you could do it right then left, but let's go with typical notation). This would mean the call would be to the left of 3 in A[0]-A[1], leaving the partition function with *partition*$(A, 0, 1)$.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 7a (14 points) **Fall 2019, CU-Boulder**

3. (4 pts) Draw the tree of recursive calls that Quicksort makes to sort the list E, X, A, M, P, L, E in alphabetical order. Use the last element in the sub-list in each recursive call as the pivot.

   Solution.



Name: Luna McBride
ID: 107607144

4. (6 pts) You are given a collection of n bottles of different widths and n lids of different widths and you need to find which lid goes with which bottle. You can compare a lid to a bottle, from which you can determine if the lid is larger than the bottle, smaller than the bottle, or the correct size. However, there is no way to compare the bottles or the lids directly to each other, i.e. you can't compare lids to lids or bottles to bottles. Design an algorithm for this problem with an average-case efficiency of $\Theta(nlgn)$

   Solution.
   Quicksort(A,B,p,r): //A=array of lids, B= array of bottles
   − − > if p<r:
   − −/ − − > q=Partition(A,B,p,r) //Split the arrays
   − −/ − − > Quicksort(A,B,p,q-1) //Call sort on left side
   − −/ − − > Quicksort(A,B,q+1,r) //Call sort on right side
   End Basic Quicksort

   Partition(A,B,p,r):
   − − > x=A[r] //Hold the last cap
   − − > i=p-1

   − − > //For loop using the cap to sort the bottles
   − − > for j in range(p,r): //For the bottles
   − −/ − − > if B[j] ≤ x: //If cap is too big/bottle too small
   − −/ − −/ − − > i+=1
   − −/ − −/ − − > swap(B[j],B[i]) //Bring the smaller bottles backward
   − − > swap(B[i+1],B[r]) //put the correct bottle in place

   − − > y=B[i+1]
   − − > ni=p-1

   − − > //For loop using the picked bottle to sort the caps
   −− > for ew in range(p,r): //Checks if the caps are less/greater than the bottle picked
   − −/ − − > if A[ew]!=x: //Blocks the previous cap
   − −/ − −/ − − > if A[ew] ≤ y: //If the cap is too small for the center bottle
   − −/ − −/ − −/ − − > ni+=1
   − −/ − −/ − −/ − − > swap(A[ew],A[ni]) //Cap is smaller, swap it backward

   − − > swap(A[ni+1],A[r]) //Bring the original cap to the right spot
   − − > return i+1 //Return where these pivots were located