Name: Alex Book
ID: 108073300

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: Alex Book
ID: 108073300

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: Break problem into several smaller instances of the same problem, each with same features of original problem.
   Conquer: If smaller instance is trivial, solve it, otherwise divide it again.
   Combine: Combine results for smaller problems into one solution for larger problem.

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

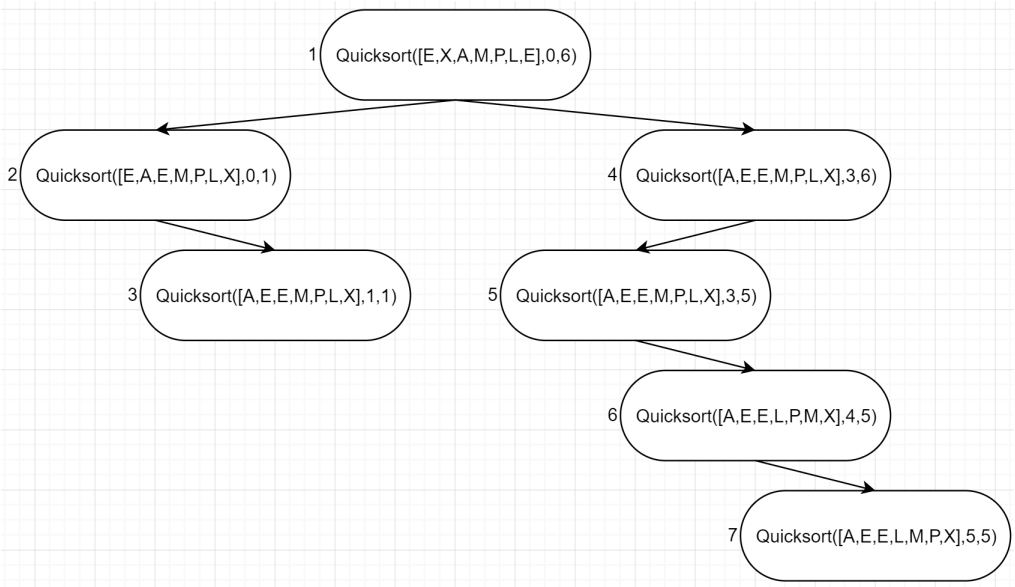
   (a) (1 pt) What is the value of the pivot in the call *partition*$(A, 0, 6)$?
   Solution.
   3 (value of last element in the array)

   (b) (1 pt) What is the index of that pivot value at the end of that call to *partition*()?
   Solution.
   2 (2 elements are smaller than it, 4 elements are larger than it)

   (c) (1 pt) On the next recursive call to Quicksort, what sub-array does *partition*() evaluate?
   Solution.
   A=[2,1] (evaluates the left-side subarray with the next call)

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.
   Note that the algorithm given in lecture sorts the entire left side of the array first (left of the pivot), then the right side (right of the pivot), updating the overall array as it goes, so iterations on the same level of the array may not necessarily be using the exact same array. The order in which the nodes are iterated is denoted by the number to the left of each node, starting at iteration 1.



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.
   Rhonda specified in office hours that turning in code/pseudocode for this problem would be okay. Explanation for specifics are in the comments, but the main note is that bottles are never compared to bottles and lids/caps are never compared to lids/caps, only to each other, as desired by the problem description.

```
def partition(A,p):
    # partitions array A into 3 sections:
    # left = elements less than p, in order that they appear in A
    # equal = singleton array of element in A that is equal to p
    # right = elements greater than p, in order that they appear in A
    left=[]
    equal=[]
    right=[]
    for i in A:
        if i<p:
            left.append(i)
        if i>p:
            right.append(i)
        if i==p:
            equal.append(i)
    return left,equal,right

def bottlesCapsSort(bottles,caps):
    if len(bottles)<=1 and len(caps)<=1:
        # if the passed arrays are only 1 item, they are already sorted
        return bottles,caps
    sortedBottles = []
    sortedCaps = []
    x=caps[-1] # pivot to sort bottles is last element in caps
    leftB,equalB,rightB = partition(bottles,x)
    # uses sorted term in bottles to sort caps
    leftC,equalC,rightC = partition(caps,equalB[0])
    # solved arrays are simply the sorted elements returned by recursion
    solvedLeftB,solvedLeftC = bottlesCapsSort(leftB,leftC)
    solvedRightB,solvedRightC = bottlesCapsSort(rightB,rightC)
    # add items from left first so the return arrays match in ascending order
    sortedBottles.extend(solvedLeftB)
    sortedBottles.extend(equalB)
    sortedBottles.extend(solvedRightB)
    sortedCaps.extend(solvedLeftC)
    sortedCaps.extend(equalC)
    sortedCaps.extend(solvedRightC)
    return sortedBottles,sortedCaps
```

Name: Alex Book
ID: 108073300

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