Name: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

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 - split up the problem space into smaller chunks as part of the original state space
   conquer - if the solution of the subproblem is trivial then solve it
   combine - combine conquered problems to create global solution

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

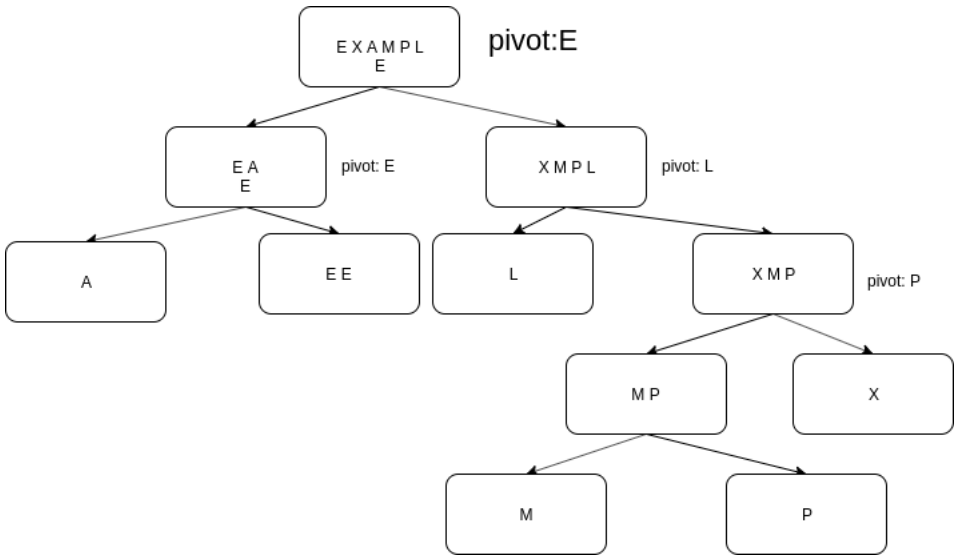
   (a) (1 pt) What is the value of the pivot in the call $partition(A, 0, 6)$?
   Solution. 3 if we are using a deterministic partition. if we are using random pivot picking then it could be any of the values in the array A.

   (b) (1 pt) What is the index of that pivot value at the end of that call to $partition()$?
   Solution. 3 (using deterministic)

   (c) (1 pt) On the next recursive call to Quicksort, what sub-array does $partition()$ evaluate?
   Solution. it evaluates $A' = [2, 1, 3]$ as it is the left array.

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: Nikolaas Bender
ID: 106977096

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

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 $\Theta(nlogn)$

   Solution. choose a cap to compare to bottles. use the cap to split bottles into 3 sections less than, equal to, greater than. then use the equal to bottle as found before to sort the caps. repeat until each section is left with one bottle and cap. recombine each single element array. the list is sorted.

   This is a python implementation of the above algorithm. note the n^2 is to prove that my algorithm is more efficient. paste this into a python file if you have any questions.

```
import numpy as np
from random import shuffle
import time
import threading
import matplotlib.pyplot as plt



TESTS = 1000


def part(arr, p):
 eq = []
 le = []
 gr = []
 for n in arr:
 if n < p:
 le.append(n)
 if n > p:
 gr.append(n)
 if n == p:
 eq.append(n)
```

```
 # if len(eq) != 1:
 #  print("error, it didn't find the equal to element")
 return le, eq, gr

def sort(b, c):
 # print(b, c, len(b))
 bret = []
 cret = []
 if len(b) <= 1:
 return b, c
 # Pivot for bottles is a random cap
 p = c[np.random.randint(low=0, high=len(b))]
 ble, beq, bgr = part(b, p)
 # Pivot for caps is the bottle that the random cap links up to
 p = beq[0]
 cle, ceq, cgr = part(c, p)
 sble, scle = sort(ble, cle)
 sbgr, scgr = sort(bgr, cgr)
 bret.extend(sble)
 cret.extend(scle)
 bret.extend(beq)
 cret.extend(ceq)
 bret.extend(sbgr)
 cret.extend(scgr)
 return bret, cret

def n2sort(b, c):
 bubblebottles = threading.Thread(target=bubblesort, args=(b,))
 bubblecaps = threading.Thread(target=bubblesort, args=(c,))

 # starting thread 1
 bubblebottles.start()
 # starting thread 2
 bubblecaps.start()

 # retbottles = bubblebottles.get()
 # retcaps = bubblecaps.get()
```

```
 # wait until thread 1 is completely executed
 retbottles = bubblebottles.join()
 # wait until thread 2 is completely executed
 retcaps = bubblecaps.join()

 return retbottles, retcaps

def bubblesort(nums):
 # We set swapped to True so the loop looks runs at least once
 swapped = True
 while swapped:
 swapped = False
 for i in range(len(nums) - 1):
 if nums[i] > nums[i + 1]:
 # Swap the elements
 nums[i], nums[i + 1] = nums[i + 1], nums[i]
 # Set the flag to True so we'll loop again
 swapped = True


quicktimes = []
n2times = []

def testquick():
 for i in range(0, TESTS):
 bottles = [item for item in range(0, i)]
 caps = [item for item in range(0, i)]
 shuffle(bottles)
 shuffle(caps)
 start = time.time()
 sortbottles, sortcaps = sort(bottles, caps)
 quicktimes.append(time.time() - start)

def testn2():
 for i in range(0, TESTS):
 bottles = [item for item in range(0, i)]
 caps = [item for item in range(0, i)]
 shuffle(bottles)
```

```
 shuffle(caps)
 start = time.time()
 sortbottles, sortcaps = n2sort(bottles, caps)
 n2times.append(time.time() - start)


slowsort = threading.Thread(target=testn2, args=())
fastsort = threading.Thread(target=testquick, args=())

# starting thread 1
slowsort.start()
# starting thread 2
fastsort.start()

# wait until thread 1 is completely executed
slowsort.join()
# wait until thread 2 is completely executed
fastsort.join()

# sortbottles, sortcaps = sort(bottles, caps)

# print(sortbottles, sortcaps)

trials = [item for item in range(0, TESTS)]

plt.plot(trials, quicktimes, 'r--', trials, n2times, 'bs')
plt.xlabel('length of unsorted array')
plt.ylabel('time to compute')
plt.show()
```