Name: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

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: $\boxed{\text{Nikolaas Bender}}$
ID: $\boxed{106977096}$

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 worst case scenarios for quicksort are when the list is already sorted in ascending order, sorted in descending order, or all elements are the same.
   best case, a pivot would split the array in half.
   worst case, the pivot splits at one of the absolute ends of the array.

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

   Solution. $2T(\frac{n}{2}-1)+Cn$
   this is derived from $T_1(n) = T_2(\frac{n}{2} + T_2(\frac{n}{2}) + O(n)$ and $T_2(n) = T_1(n-1) + T_1(1) + O(n)$

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

   Solution. The algorithm that goes between best and worst. The algorithm is splits the array in half then almost replicates it because it chooses the worst case. That choosing the worst case and effectively doubling the depth of the tree is a constant time. It ends up being $nlog(n)$ despite choosing worst split half the time.

Name: Nikolaas Bender
ID: 106977096

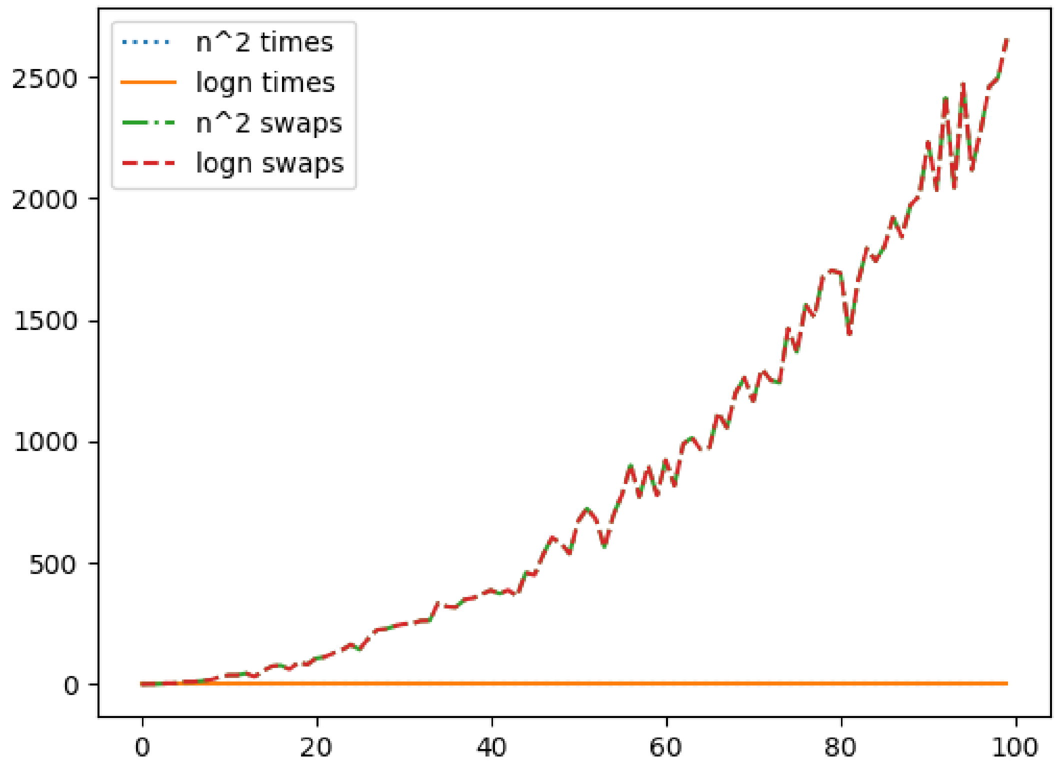
2. *(14 pts total) In PS1b, you were asked to count flips in a sorting algorithm with quadratic running time. The problem definition looked something like this:*
Let $A = \langle a_1, a_2, \ldots, a_n \rangle$ be an array of numbers. Let's define a 'flip' as a pair of distinct indices $i, j \in \{1, 2, \ldots, n\}$ 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.

Solution.


n^2 times
logn times
n^2 swaps
logn swaps
0
500
1000
1500
2000
2500
0
20
40
60
80
100

Name: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

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. randomly selected indices to create S
   $[12, 65, 34, 76, 44, 67, 55, 43] \rightarrow [12, 34, 43, 44, 55, 65, 67, 76]$
   $m = 44$ or 55 I will use 44
   left $= [34, 32, 1, 23, 12, 13, 43, 34, 44, 23, 2]$
   right $= [45, 90, 54, 65, 76, 67, 56, 45, 55]$
   both right and left have fewer than 15 elements so both will be sorted with insertion sort.
   sorted left $= [1, 2, 12, 13, 23, 23, 32, 34, 34, 43]$
   sorted right $= [45, 45, 54, 55, 56, 65, 67, 76, 90]$
   these would be concatenated together to be:
   $[1, 2, 12, 13, 23, 23, 32, 34, 34, 43, 45, 45, 54, 55, 56, 65, 67, 76, 90]$

(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. m is more likely to be a good pivot because it is the median of a probably representative sample of the total data set so the median of the random subset is probably close to the median of the total set.

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

Solution. for the random selection of S and sorting it the time complexity is n.
for splits greater than 15 the complexity is $nlog(n)$.
for splits less than 15 the time complexity is at worst is constant 15^2

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

Solution. $n^3log(n)$

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.

```
    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()
```