Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1b (44 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.

---

1. *(34 pts total) 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) (8 pts) Write a Python code for an algorithm, which takes as input a positive integer n, **randomly shuffles an array of size n** with elements $[1, \ldots, n]$ and counts the total number of flips in the shuffled array.
   Also, run your code on a bunch of n values from $[2, 2^2, 2^3, ...2^{20}]$ and present your result in a table with one column as the value of n and another as the number of flips. Alternatively, you can present your table in form of a labeled plot with the

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1b (44 points) **Fall 2019, CU-Boulder**

2 columns forming the 2 axes.

Note: The .py file should run for you to get points and name the file as `Lastname-Firstname-MMDD-PSXi.pdf`. You need to submit the code via Canvas but the table or plot should be on the main .pdf.

Element Count	Flip Count
2	1
4	5
8	15
16	70
32	207
64	1136
128	4189
256	15986
512	63060
1024	259556
2048	1055185
4096	4188112

(b) (4 pts) At most, how many flips can A contain in terms of the array size n? Hint: The code you wrote in (a) can help you find this. Explain your answer with a short statement.

At most the number of total flips of array size n will be equal to the sum of each sub-array from array[k:n] for each k, which can be written as

$$\sum_{k=1}^{n} n - 1$$

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1b (44 points) **Fall 2019, CU-Boulder**

(c) (10 pts) We say that A is sorted if A has no flips. Design a sorting algorithm that, on each pass through A, examines each pair of consecutive elements. If a consecutive pair forms a flip, the algorithm swaps the elements (to fix the out of order pair). So, if your array A was [4,2,7,3,6,9,10], your first pass should swap 4 and 2, then compare (but not swap) 4 and 7, then swap 7 and 3, then swap 7 and 6, etc. Formulate pseudo-code for this algorithm, using nested for loops.
Hint: After the first pass of the outer loop think about where the largest element would be. The second pass can then safely ignore the largest element because it's already in it's desired location. You should keep repeating the process for all elements not in their desired spot.

def flipsort(array):
initiate i = 0 and j = length of array -1
have a condition that tests if array is empty which we know when j = -1
while loop runs as long as $j > i$, essentially as long as j is not equal to zero

Inside while loop will be an inner for loop that will range from i to j that will check the sub-array from array[0:j]. Inside this loop have a condition that checks if array[i] and array[i+1] are a flip, if this is indeed a flip , swap them.
after the inner loop finishes executing, decrement the value of j since the max element of the sub-array iteration that just finished will be in the proper spot

Reiterate over sub-array [0:j-1] until outer loop condition is met and terminates.

At termination of outer loop, the array will be sorted.

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1b (44 points) **Fall 2019, CU-Boulder**

Another way to view. . .
def flipsort(array):
i = 0
j = len(array)-1
conditional that checks if array is empty
outer loop: while j greater i
inner loop: for i in range (j) , loop that checks sub-array up until j
inner loop condition that checks if array[i] and array[i+1] are a flip
if that inner loop conditional is true then swap
At the exit of the inner loop ,

$$j = j - 1$$

since the max element of current iteration will be in the proper spot.

(d) (4 pts) Your algorithm has an inner loop and an outer loop. Provide the 'useful' loop invariant (LI) for the inner loop.You don't need to show the complete LI proof.

In the second loop the useful loop invariant is i because i will will either be less than j, telling us that all potential comparisons and swaps have not yet been seen or if i equals j then all potential swaps and comparison have been seen.

(e) (8 pts) Assume that the inner loop works correctly. Using a loop-invariant proof for the outer loop, formally prove that your pseudo-code correctly sorts the given array. Be sure that your loop invariant and proof cover the initialization, maintenance, and termination conditions.

LI: The loop invariant is variable j possessing multiple properties such that j is always equal to either 0, telling us that the array is sorted or, j is greater than zero, telling us that the array has not been fully sorted

Initiation: Loop is initiated at i = 0 and j = length of array -1 meaning that the entire array has not been checked for all potential flips and swaps, therefore array is not yet sorted and our LI holds true

Maintenance: If at the ith iteration when j is still greater than 0, the sub-array from [j:length of array -1] will be sorted but the sub-array from [0:j] will not be sorted and more swaps are needed, and our loop invariant holds true. If at the start of the ith iteration wher j is equal to zero, the loop will trigger a termination and the array from [0:length of array -1] will be sorted, and our LI holds true.

Termination: At termination, when when j equals zero, the given array will be sorted from least to greatest which is what our algorithm was intended for and therefore it works.

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Problem Set 1b (44 points) **Fall 2019, CU-Boulder**

2. *(6 pt) If r is a real number not equal to 1, then for every n ≥ 0,*

$$\sum_{i=0}^{n} r^i = \frac{(1 - r^{n+1})}{(1 - r)}.$$

Rewrite the inductive hypothesis from Q3 on PS1a and provide the inductive step to complete the proof by induction. You can refer to Q3 on PS1a to recollect the first 2 steps.

Base Case, let n = 0

$$\sum_{i=0}^{0} r^i = \frac{(1 - r^{0+1})}{(1 - r)} 1 = \frac{(1 - r)}{(1 - r)} 1 = 1$$

Inductive Hypothesis Assume,

$$\sum_{i=0}^{k} r^i = \frac{(1 - r^{k+1})}{(1 - r)}.$$

show n = $k + 1$ holds

$$\sum_{i=0}^{k} r^i + r^{k+1} = \frac{(1 - r^{k+1})}{(1 - r)} + r^{k+1}.$$

$$\frac{(1 - r^{k+1})}{(1 - r)} + \frac{r^{k+1}}{1}$$

$$\frac{(1 - r^{k+1})}{(1 - r)} + \frac{r^{k+1}}{(1 - r)}$$

$$\frac{(1 - r^{k+1})}{(1 - r)} + \frac{r^{k+1} - r^{k+2}}{(1 - r)}$$

$$\frac{1 - r^{k+2}}{1 - r}$$

By Principle of Math Induction the original equation holds for all values, n, greater than or equal to zero. QED

Name: Jonathan Phouminh

ID: 106054641

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1b (44 points) **Fall 2019, CU-Boulder**

3. *(4 pt) Refer to Q2b on PS1a and finish the LI based proof with all the steps.*

Loop Invariant: At the start of the ith iteration , ret has the index of n if n exists in a[0. . . i-1] otherwise ret = -1

Initialization: loop initialized at i = 0 thus searching the sub-array at a[0], leaving 2 cases.
case 1) a[0] is not equal to n, leaving ret to be equal to -1, LI holds true
case 2) a[0] is the element n and ret will equal i, LI holds true

Maintenance: For the ith iteration there will be two cases
case 1) if a[i] is equal to n, ret will equal i
case 2) if a[i] is not equal to n, ret will still be equal to -1
loop invariant holds true in both cases

Termination: Loop terminates when i is equal to n-1, ret will be equal to the index of the found elemnt n or -1 if not found, which is what we wanted our algorithm to do. Therefore our algorithm is correct.