Name: Alex Book
ID: 108073300

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

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.

n	number of flips
2	0
2^2	1
2^3	11
2^4	74
2^5	317
2^6	998
2^7	3,917
2^8	16,622
2^9	61,914
2^{10}	257,877
2^{11}	1,047,261
2^{12}	4,126,806

(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.

The maximum number of flips contained by A occurs when the array happens to be in decreasing order (which is the opposite of what is desired). In this case, each element in the array would have all following elements count as a flip. This sum is equal to $\frac{n(n-1)}{2}$ for an array of size n.

For example, if an array has a length n=10 and is in decreasing order, there would be 9+8+7+6+5+4+3+2+1=45 flips, $\frac{10(10-1)}{2}$=45.

(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(n)
  A=randomArray(n)
  for i in range(n)
    for j in range(n-i-1)
      if A[j]>A[j+1]
        swap(A[j],A[j+1])
  return A
```

(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.

At the beginning of each iteration of the loop, A[j] is the largest value in the subarray A[0...j].

(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.

Loop invariant:
At the beginning of each iteration of the loop, the last i elements in the array are sorted and in place.

Initilialization:
Before the 0-th iteration, the last 0 elements in A are sorted (no elements are necessarily sorted yet, as the algorithm has not scanned any value of A).

Maintenance:
Assuming that the inner loop works correctly, each time the outer loop iterates, the largest element in the array should be sorted to the end of the subarray A[0...n-i-1] (leaving alone the previously-sorted last i-1 elements). Therefore, at the beginning of each loop, the last i elements are sorted and in place.

Termination:
The loop terminates when i=n. At this point, the whole array is sorted, so the last i elements (all array elements) are sorted and in place.

Name: Alex Book
ID: 108073300

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 $n \geq 0$,*

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

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:
Consider the case n=0. Left side is r^0=1. Right side is $\frac{1-r}{1-r}$=1. Therefore, the base case holds true.

Inductive Hypothesis:
Assume this equation is true for n=k. That is, $\sum_{i=0}^{k} r^i = \frac{(1-r^{k+1})}{(1-r)}$.

Inductive Step:
Consider the k+1 case. We have that:

$\sum_{i=0}^{k+1} r^i = \sum_{i=0}^{k} r^i + r^{k+1}$ (by associativity of addition)
$= \frac{(1-r^{k+1})}{(1-r)} + r^{k+1}$ (by inductive hypothesis)
$= \frac{(1-r^{k+1})+(1-r)(r^{k+1})}{(1-r)}$
$= \frac{1-(1-r-1)(r^{k+1})}{(1-r)}$
$= \frac{1-r^{k+2})}{(1-r)}$

The result follows by induction.

Name: Alex Book
ID: 108073300

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 i-th iteration, ret has the index of n if n exists in A[0...i-1] otherwise ret=-1.

Initialization:
Before the 0-th iteration, ret=-1, as the function has not yet scanned any values in A, and therefore must not have yet found n.

Maintenance:
Case 1: A[i]==n
ret is set to i, and now contains index of n.
Case 2: A[i]!=n
ret remains set to -1, since n was not found.
At the start of the next loop, ret equals the index of n if found, otherwise still equals -1.

Termination:
The loop terminates when i=length(A). If n was found anywhere in the array, ret is storing its index. Otherwise, it still equals -1.