

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
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 $A = \langle a_1, a_2, \dots, a_n \rangle$ be an array of numbers. Let's define a 'flip' as a pair of distinct indices $i, j \in \{1, 2, \dots, n\}$ 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 $[1, \dots, n]$ and counts the total number of flips in the shuffled array.

Also, run your code on a bunch of n values from $[2, 2^2, 2^3, \dots, 2^{20}]$ 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: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
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.

Copy and pasted from the results of my code. This is just one randomized implementation, which is likely different than other runs of the same code:

N Flips

— —

2 0

4 2

8 12

16 72

32 256

64 1059

128 4330

256 16104

512 66219

1024 269978

2048 1042675

4096 4217964

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

- (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.

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

The worst case (or the most flips) occurs when every value is in reverse order. That means every combination downward is another flip. For number, say, 16, that means it pairs with all 15 numbers until one, making 15 combinations from 16 to the end. Repeat that with every number from n to the end, then that is $n-1$, then $n-2$... to 1, which fits really well as a sum.

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
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 its desired location. You should keep repeating the process for all elements not in their desired spot.

```
def sortalgorithmpseudo (array):  
  —while not sorted:  
    —for the length of the array up to last unsorted value  
      —check if values are a flip  
      —if a flip, then swap the two values  
      —if not a flip, leave alone  
    —end for loop  
    —check if it is sorted, if not, wrap back  
  —once sorted, break while loop
```

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder



Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

- (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.

Note: $i-1$ used since the i is incremented. This would be the i before the increment

At the end of each loop, we know that the two values were flipped if not in $x_1 < x_2$ order (or not flipped if in the correct order). There does not technically need the possibility of being equal in this case following the instructions in part A, however, part C makes it unclear whether this is the case, and as such, I will use a less than-equal to sign just in case two indices have the same value

Therefore:

Loop Invariance: $A[i - 2] \leq A[i - 1]$

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

- (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.

A useful invariance would be that the end value of the section looked at $((\text{array length}-1)-i)$ (under the "hint" part of part c) and have that as the maximum value. The same less than-equal to discussion from 2d will be applied here. i is also used as a scalar here to get to the end.

Loop Invariance: $(A[0] \dots A[(\text{len}(A)-1)-i]) \leq A[(\text{len}(A)-1)-(i-1))]$

Initialization: The value of the right side overflows past the length of the array. Since the array is not sorted, having the largest value not within the scope makes sense until we can build the array for its size.

Maintenance: With each pass of the while loop, the values keep swapping in the for loop, incidentally putting the absolute biggest in the last spot. We could just use the absolute biggest is always at the end after each pass, however, the biggest not counting the one from the last round would make more logical sense to explain a sorting algorithm.

Termination: The last round comes to having the last value compared to itself, following every other one being put as the biggest of the round. The smallest becomes the smallest by default and the array is henceforth sorted

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
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.

Inductive Hypothesis: For some integer k such that $k \geq 0$ and $n! = 1$, we can assume $\sum_{i=0}^k r^i = \frac{(1-r^{k+1})}{(1-r)}$

Inductive Step: If this relation fits with k , then it should reasonably be said that $k+1$ should work as well.

$\sum_{i=0}^{k+1} r^i$, Which is the same as $\sum_{i=0}^k r^i + r^{k+1}$
 Since $\sum_{i=0}^k r^i = \frac{(1-r^{k+1})}{(1-r)}$, we can plug in $\frac{(1-r^{k+1})}{(1-r)}$

$$\begin{aligned} & \frac{(1-r^{k+1})}{(1-r)} + r^{k+1} \\ & \frac{(1-r^{k+1})}{(1-r)} + \frac{(1-r)r^{k+1}}{1-r} \\ & \frac{(1-r^{k+1})}{(1-r)} + \frac{(r^{k+1}-r^{k+2})}{1-r} \\ & \frac{(1-r^{k+1})+(r^{k+1}-r^{k+2})}{(1-r)} \\ & \frac{1-r^{k+1}+r^{k+1}-r^{k+2}}{(1-r)} \\ & \frac{1-r^{k+2}}{(1-r)} \\ & \frac{1-r^{(k+1)+1}}{(1-r)} \\ & \frac{1-r^{k+1}}{(1-r)} \rightarrow \frac{1-r^{(k+1)+1}}{(1-r)} \text{ as } k \rightarrow k+1 \text{ (it is just } k \text{ suited for } k+1) \end{aligned}$$

Therefore, via weak induction, the relation $(\sum_{i=0}^n r^i = \frac{(1-r^{n+1})}{(1-r)})$ works for a k and a $k+1$, and thus is a suitable relation for all values $n \geq 0$

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 1b (44 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

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

Loop Invariance: if n exists in $A[0 \dots i-1]$, $ret = \text{index of } n$. Else, $ret = -1$ (following the 1a answer set).

Initialization: No values have been compared, so ret simply takes on its initialized value of -1 .

Maintenance: The loop compares the value of $A[i]$ (now $A[i-1]$) to the needed value we are searching for. If this is the value, we have obtained the index of n and that is what ret becomes. Otherwise, ret remains as -1 . Since this is the two stipulations in the invariance, the values fit.

Termination: The invariance terminates with the array. If the value was found, ret has held that index since it was found. If it was not in the array, ret remains as -1 . Both defined in the invariance, so even if you do find the end without the value you wanted, the invariance holds.