

Name: Michael Rogers

ID: 105667404

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: Michael Rogers

ID: 105667404

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: Michael Rogers

ID: 105667404

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. Worst Case: One sub-array has $n - 1$ elements and the other sub-array has zero elements. This would mean either the pivot is the last or first element in the array.

Best Case: Each sub-array have $n/2$ elements. This would mean the pivot is exact in the middle of the array.

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

Solution. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Using Master's Theorem:

$\log_b a = \log_2 2 = 1, k = 1$

Case 2:

$$\Rightarrow T(n) \in \Theta(n \log n)$$

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

Solution. The **Partition** function in this algorithm essentially determines the entire running time of the algorithm. **Partition** determines where the pivot point is, and as you can see in part a, the difference of the pivot point is dependent on the elements of the array. If the elements cause **Partition** to pick the last or first element in the array, then it will greatly hurt the runtime because it will have to sort $n-1$ elements.

Name: Michael Rogers

ID: 105667404

CSCI 3104, Algorithms

Profs. Hoenigman & Agrawal

Problem Set 7b (47 points + 10 pts extra credit)

Fall 2019, CU-Boulder

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 $A = \langle a_1, a_2, \dots, a_n \rangle$ 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) (14 pts) Write a Python program with the following features:

- i. (2 pts) Generates a sequence of n numbers in the range $[1, \dots, n]$ 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 $\theta(n \lg n)$ 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 $[2, 2^2, 2^3, \dots, 2^{12}]$ 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.

Name: Michael Rogers

ID: 105667404

CSCI 3104, Algorithms

Profs. Hoenigman & Agrawal

Problem Set 7b (47 points + 10 pts extra credit)

Fall 2019, CU-Boulder

Solution. Table of solutions:

slowSort	fastSort
2	4
15	8
73	16
223	32
980	64
3946	128
15729	256
63249	512
258675	1024
1076155	2048
4240854	4096

My counting of flips is not correct. I couldn't figure out how to fix it, but the algorithm does implement mergesort and sorts the array.

Name: Michael Rogers

ID: 105667404

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: Michael Rogers

ID: 105667404

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. First iteration:

$$S = [34, 1, 54, 65, 67, 34, 23, 2], \text{ Sorted } S = [1, 2, 23, 34, 34, 54, 65, 67], m = 34$$

$$A_1 = [34, 32, 1, 23, 12, 13, 23, 2], A_2 = [45, 90, 43, 54, 65, 76, 67, 56, 45, 44, 55]$$

After this first iteration, the algorithm will run recursively on the two sub arrays until the array is sorted.

Name: Michael Rogers

ID: 105667404

CSCI 3104, Algorithms

Profs. Hoenigman & Agrawal

Problem Set 7b (47 points + 10 pts extra credit)

Fall 2019, CU-Boulder

- (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. Unless S is the highest or lowest $2\lfloor\sqrt{n}\rfloor$ elements in the array, then on average, the two arrays will be similar in size. At the worst case senario, there will be $\lfloor\sqrt{n}\rfloor$ numbers in one sub-array.

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

Solution. There are only $2\lfloor\sqrt{n}\rfloor$ elements in the array. Therefore, in order to find the median we need to find the element in the middle of the array. This will take $\Theta(\sqrt{n})$ to find it's median because the array is sorted.

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

Solution. Insertion sort takes $O(n^2)$ to sort a given array at worst case. Since we won't be sorting an array with Insertion sort will be used on at most $2\lfloor\sqrt{n}\rfloor$ elements. This means that the theata bound to sort is $O((2\sqrt{n})^2) = O(4n)$. As found in part c, finding the median requires $\Theta(\sqrt{n})$ to find the so the reccurence is $T(n) = T(n - \sqrt{n}) + \Theta(\sqrt{n})$

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.