Name: Michael Rogers

ID: 105667404

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

Problem Set 8a (14 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.

Name: Michael Rogers
ID: 105667404

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 8a (14 points) **Fall 2019, CU-Boulder**

1. (2 pts) If the arrays, $A = [12, 14, 23, 34]$ and $B = [11, 13, 22, 35]$ are merged, list the indices in A and B that are compared to each other. For example, $A[0], B[0]$ means that $A[0]$ is compared to $B[0]$.

   Solution. Comparisons:

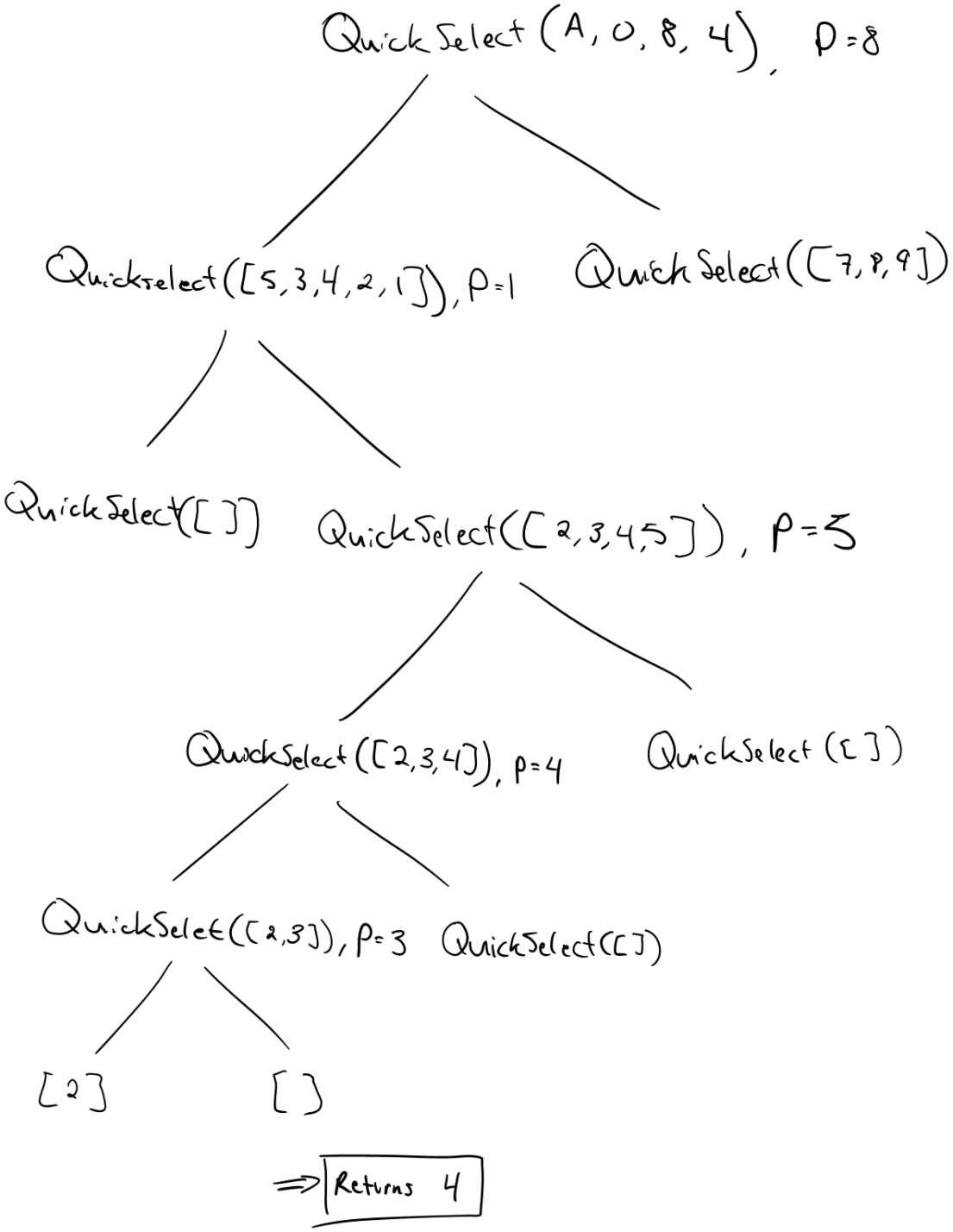
   A[0], B[0]
   A[0], B[1]
   A[1], B[1]
   A[1], B[2]
   A[2], B[2]
   A[2], B[3]
   A[3], B[3]

2. (3 pts) Illustrate how to apply the QuickSelect algorithm to find the $k = 4$th smallest element in the given array: `A = [5, 3, 4, 9, 2, 8, 1, 7, 6]` by showing the recursion call tree.

   Solution. Drawing on following page.

Name: Michael Rogers
ID: 105667404

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 8a (14 points) **Fall 2019, CU-Boulder**



3. (1 pt) Explain in 2-3 sentences the purpose of the Median of Medians algorithm.

   Solution. The idea of the median of medians algorithm is to find a pivot for quicksort to try to get as optimal of a solution as possible. This algorithm finds the medians from multiple lists of randomly selected elements, then selects the median of those medians. This produces a median that should be close to the middle of the array to be sorted.

4. (4 pts) Illustrate how to apply the Median of Medians algorithm (A Deterministic QuickSelect algorithm) to find the 4th largest element in the following array: `A = [6, 10, 80, 18, 20, 82, 33, 35, 0, 31, 99, 22, 56, 3, 32, 73, 85, 29, 60, 68, 99, 23, 57, 72, 25]`.

   Solution. Steps:
   Break A into lists consisting of 5 elements each and sort them:
   $A_1 = [6, 10, 18, 20, 80]$, $A_2 = [0, 31, 33, 35, 82]$, $A_3 = [3, 22, 32, 56, 99]$, $A_4 = [29, 60, 68, 73, 85]$, $A_5 = [23, 25, 57, 72, 99]$

   Place medians from each list into a new list M and find the median of medians:
   M = [18, 33, 32, 68, 57], Median of M = 32

   Use the median of medians as a pivot for A to reconstruct an A where all elements $<$ pivot are on the left and all elements $>$ pivot are to the right:
   A = [6, 10, 18, 20, 0, 31, 3, 22, 29, 23, 25, 32, 80, 33, 35, 82, 56, 99, 60, 68, 73, 85, 57, 72, 99]

   Now we can just look at the right side of the pivot because there are $>$ four elements that are bigger than the pivot:
   $A_r = [80, 33, 35, 82, 56, 99, 60, 68, 73, 85, 57, 72, 99]$

   Sort A_r:
   $SA_r = [33, 35, 56, 57, 60, 68, 72, 73, 80, 82, 85, 99, 99]$

   Get 4th to last element in SA_r:
   4th Largest element = 82

5. (4 pts) In Tuesday's lecture, we saw how the peaked array algorithm can find the maximum element in an array with one peak. For example, $A = [15, 16, 17, 14, 12]$ is a peaked array.

   (a) (2 pts) Explain how the peaked array algorithm works in sub-linear time? (You may use the recurrence relation to help with the explanation)

   Solution. The peaked array algorithm can operate in sub-linear time because it uses a similar divide-and-conquer technique. We divide the array in half, and if the middle element is larger than all of it's nieghbors, then it must be the peak. If the middle is less than the right nieghbor, then the peak must be right of the middle of the array. And the same for the left. This uses the same approach used in binary search which allows us to bring the runtime down from $O(n)$ to $O(log(n))$. For the example above, we would start with:
   Mid = 17 , $17 > 16$ and $17 > 14$ Therefore, 17 must be the peak in the array.

   (b) (2 pts) Re-write the peaked array algorithm to find a single valley in an array, such as $A = [56, 43, 32, 21, 23, 25, 57]$. The valley would be 21.

   Solution. Code:

```
def findPeak(A, s, t):
   mid = floor((s + t / 2)
   if t - s + 1 == 3: //Base Case
      return A[mid]
   if A[mid] < A[mid+1] and A[mid] < A[mid-1]:
      return A[mid]
   if A[mid] > A[mid+1]:
      return findPeak(A, mid+1, t)
   else:
      return findPeak(A, s, mid-1)
```

   Explaination: All I've done is reversed all of the comparison operators in the function. If the mid is less than it's surrounding neighbors, then it must be the valley. If it's greater than the element to the right, the valley must be in the right half of the array. It it isn't then it must be in the left.