Name: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

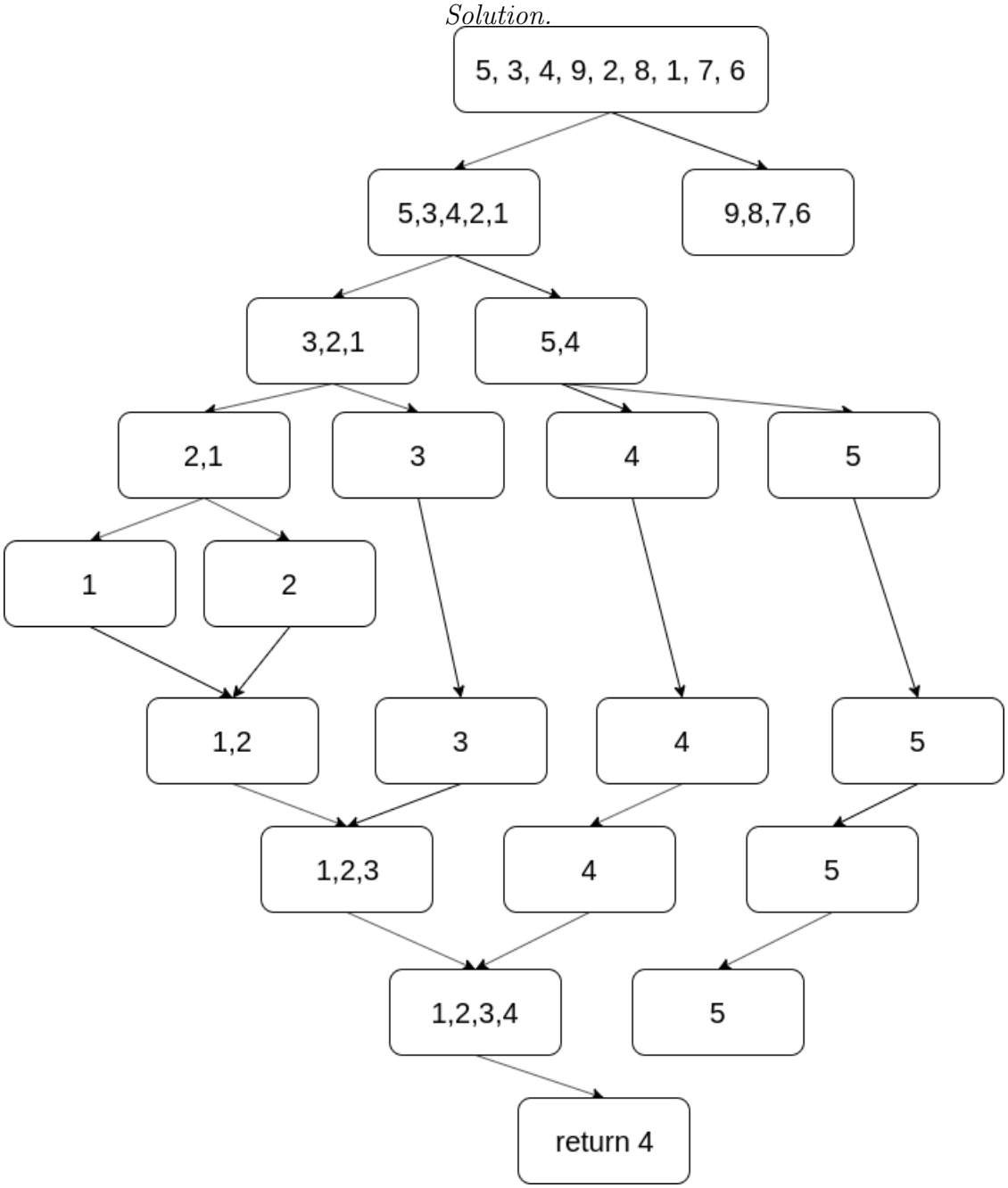
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 8a (14 points) **Fall 2019, CU-Boulder**

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



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

   Solution. It does a reasonably good job of selecting the median of an array by finding

the median value of smaller sub arrays then using those medians to find a median for the whole array. It has a linear time complexity and makes choosing good pivots for quick sort and quick select much easier and more reliable than using random selection of pivots.

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. $[6, 10, 80, 18, 20], [82, 33, 35, 0, 31], [99, 22, 56, 3, 32], [73, 85, 29, 60, 68], [99, 23, 57, 72, 25]$
   $[6, 10, 18, 20, 80], [0, 31, 33, 35, 82], [3, 22, 32, 56, 99], [29, 60, 68, 73, 85], [23, 25, 57, 72, 99]$
   the medians of these arrays are $[18, 33, 32, 68, 57] \rightarrow [18, 32, 33, 57, 68]$
   the median of the medians array is 33, this will be the pivot for our original array.
   using 33 as our pivot we get the two following sub arrays
   $[6, 10, 18, 20, 33, 31, 22, 3, 32, 29, 23, 25], [80, 82, 35, 99, 56, 73, 85, 60, 68, 99, 57, 72]$ Then the process is repeated on the lower sub array until the 4th smallest element is found.

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 algorithm works through a constant time at each level of the array then the array is split in half and each of the sub arrays are checked too. The depth is $log(n)$ and each level is constant. thats how its done in sub linear time.

   (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.

```
def findMinUtil(arr, low, high, n):

    # Find index of middle element
    # (low + high)/2
    mid = low + (high - low)/2
    mid = int(mid)

    # Compare middle element with its
    # neighbours (if neighbours exist)
    if ((mid == 0 or arr[mid - 1] >= arr[mid]) and (mid == n - 1 or arr[mid + 1
        return mid

    elif (mid > 0 and arr[mid - 1] < arr[mid]):
        return findMinUtil(arr, low, (mid - 1), n)

    else:
        return findMinUtil(arr, (mid + 1), high, n)
```

   technically this code returns the index of the lowest element however its easy to turn an index into a value by going to that index in the array.