Name: Luke Joyce

ID: 107355873

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: Luke Joyce
ID: 107355873

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]
   A[0], B[1]
   A[1], B[1]
   A[1], B[2]
   A[2], B[2]
   A[2], B[3]
   A[3], B[3]
   $->$ sorted

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.

Name: Luke Joyce
ID: 107355873

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

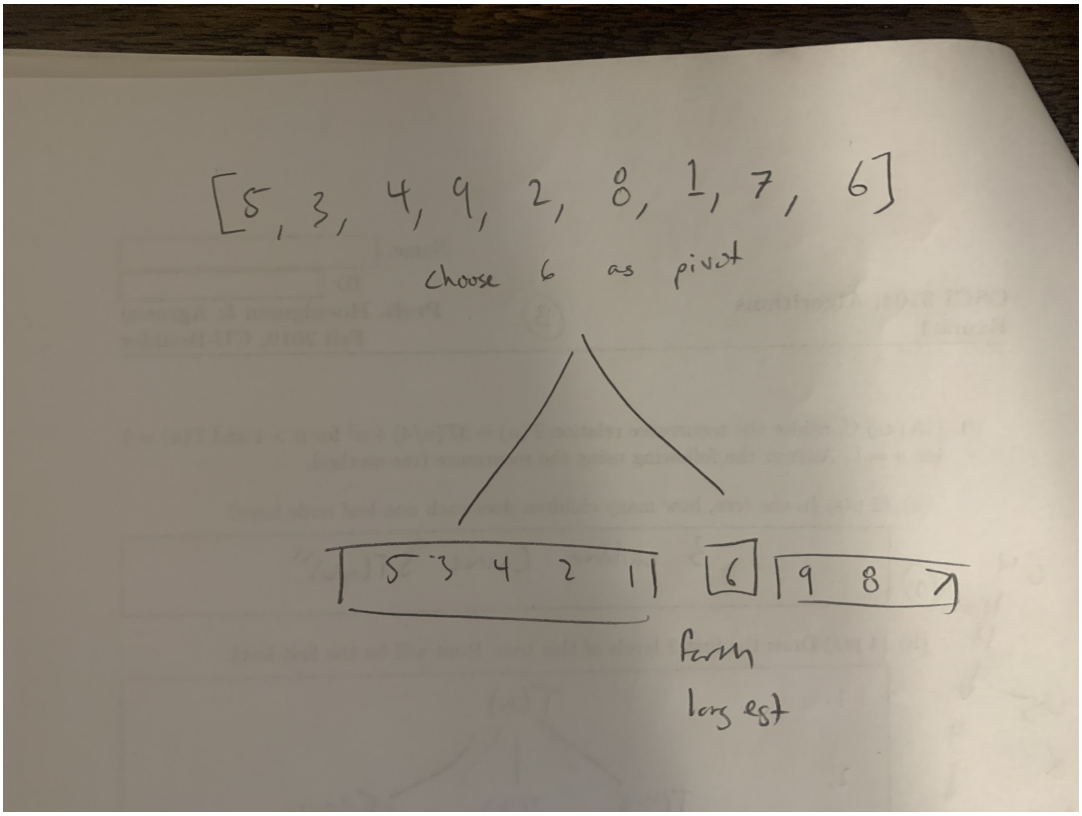

Figure 1: Caption

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

   Solution. The median of medians algorithm is a divide and conquer algorithm that divides a list into multiple lists, as quicksort does, and finds a median in each one. From the medians found, it finds a median of those values. This is helpful in finding a good pivot for sorting algorithms in linear time.

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.
   Start with [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]
   Divide this into 5 sub-arrays. We can divide it into as many sub-arrays as we want but 5 is easy because there are 25 elements in the array:
   [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].
   Sort each array and end up with:
   [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 5 arrays are the 3 element in each one: 18, 33, 32, 68, 57, respectively. We will go on and choose the median of that list, 33,as our pivot, and partition the original array A around that pivot:
   [6, 10, 18, 20, 0, 31, 22, 3, 32, 29, 23, 25], and [80, 82, 35, 99, 56, 73, 85, 60, 68, 99, 57, 72]
   We can repeat that process on this new sub-array with values greater than 33 by dividing it into 2 or 3 sub-arrays. Eventually once we have a small enough array, it doesn't consume that much time to perform a selection sort and find the 4th largest element from there.

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 because every array has at least one local maximum, and it only takes one run through the array to find local maximums. When there is only one "peak", or local maximum, then that is the maximum value in the array, because every other element in the array is smaller than one value next to it.

   (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.
   ```
   PeakedArray(A, low, high):
       mid = (low + high)/2
       if (A[mid] < A[mid+1]) && (a[mid] < A[mid-1])
           return mid
       if (A[mid] > A[mid+1])
           return PeakedArray(A, mid+1, high)
       else
           return PeakedArray(A, low, mid-1)
   ```