Name: Jonathan Phouminh

ID: 106054641

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: Jonathan Phouminh

ID: 106054641

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.

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

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

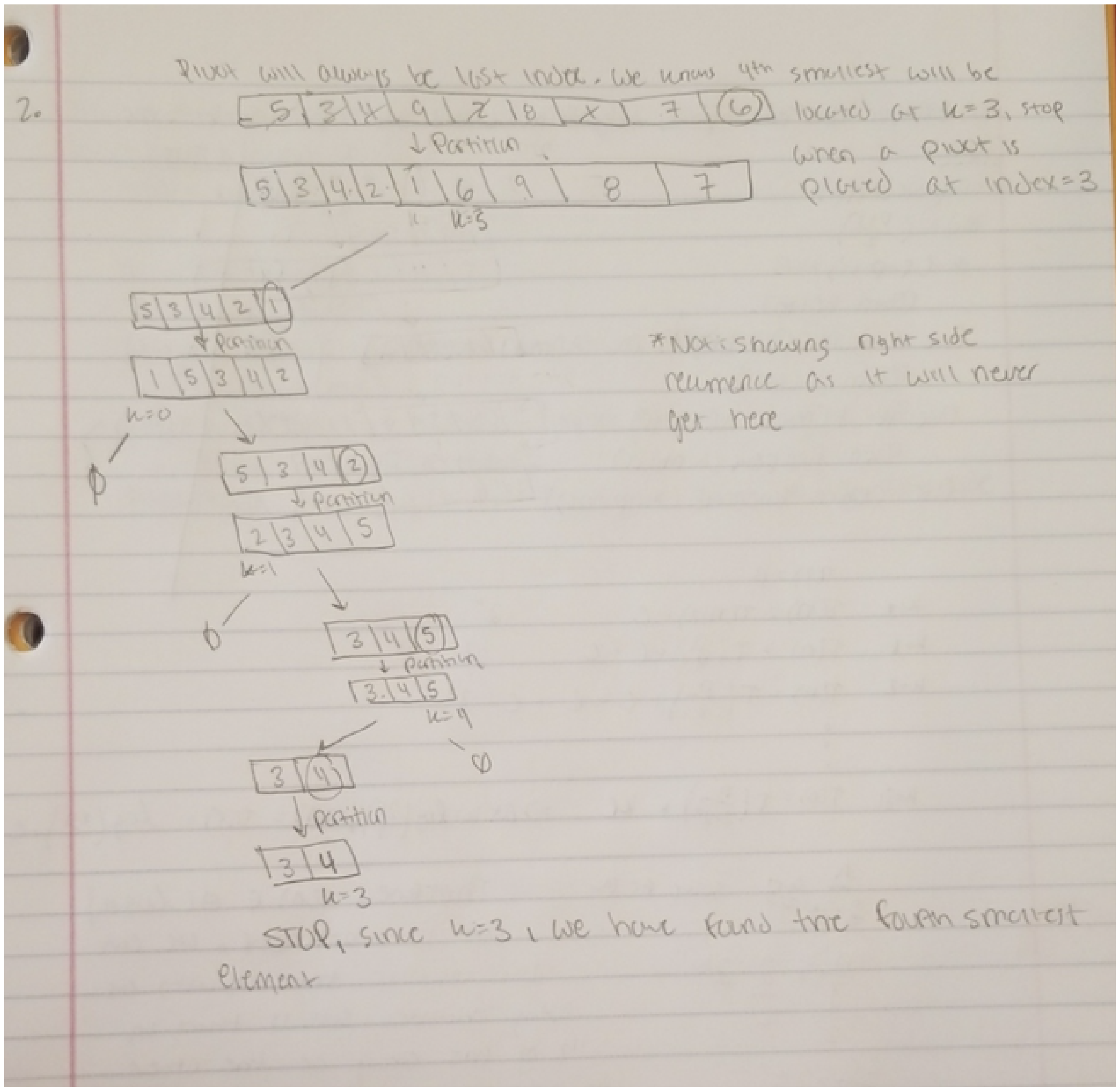
   Solution.

Name: Jonathan Phouminh

ID: 106054641

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

Problem Set 8a (14 points) **Fall 2019, CU-Boulder**



Pivot will always be last index. We know 4th smallest will be located at k=3, stop when a pivot is placed at index=3

*Not showing right side recurrence as it will never get here

STOP, since k=3, we have found the fourth smallest element

Name: Jonathan Phouminh

ID: 106054641

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.

   This algorithm lets us achieve an optimal pivot position at each iteration for our partition sub-routine. It returns an approximation to the median of the original array in $\Theta(n)$ time which is what we want.

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.

   We will perform the quicksort algorithm, but by using the MoM to find an approximate median for each sub array we will ensure optimal partitions. In the first iteration we will find the median of 5 different sub arrays arrays of the original array to get the approximate median.

```
    first sort each sub-array then find the median
A1 = [6, 10, 80, 18, 20] ; median = 18
A2 = [82, 33, 35, 0, 31] ; median = 33
A3 = [99, 22, 56, 3, 32] ; median = 32
A4 = [73, 85, 29, 60 , 68] ; median = 68
A5 = [94, 23, 57, 72, 25] ; median = 57

    make an array of all of the medians then find the median of that after sort
MoM = [18, 33, 32, 68, 57]

MoM = 33
```

   now we use the MoM as our pivot in the first iteration of partition subroutine. Then we carry out the quickselect algorithm until a pivot is placed in $A[n-4]$ since that will be where the 4th largest element will be. Essentially, applying this MoM algorithm will improve the overall runtime of the quickselect algorithm

Name: Jonathan Phouminh

ID: 106054641

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

Problem Set 8a (14 points) **Fall 2019, CU-Boulder**

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.

```
Cost | Time
c1     1
c2     1
c3     1
c4     T(n/2)  , only one of these because we only recurse once
```

   If we look at the algorithm it can be easily visualed that it only considers 1/2 of each subarray at each iteration, completely disregarding other half, so it cannot be the case that this function looks at all n values, but we can see that this function runs in sub linear time if we unroll it.

   Recurrence: $T(3) = a$, $T(\frac{n}{2}) + C$

```
 k=0   T(n) = T(n/2) + C
 k=1   T(n) = T(n/4) + C
 k=2   T(n) = T(n/8) + C
     .
     .
     .
 k=j   T(n) = T(n/2*2^j) + kc

 When we solve for when we hit the base case we will
 get that k = log(2n/3) thus kc will equal log(2n/3)*C.
 Therefore T(n) is upper bounded by O(log(n)) and
 thus our algorithm is sub-linear
```

   (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.

Name: Jonathan Phouminh
ID: 106054641

```
findValley(a,p,r):
    mid = floor((p+r)/2)

    if p-r+1 == 3:
        return A[mid]
    else if A[mid-1] > A[mid] and A[mid+1] > A[mid]:
        return A[mid]
    else if A[mid+1] < A[mid]: # valley must be on the right side
        return findValley(a, mid, r)
    else: return findValley(a,p,mid)  # valley must be on the left side
```

Collaborated with: Zach Chommala, Bao Nguyen