Name: - Olohi Goodness John
Peers: -
References: - https://brilliant.org/wiki/median-finding-algorithm/ , https://en.wikipedia.org/wiki/Median$_{o}$
$//www.youtube.com/watch?v = sNtu2oGDRvU$

Median-of-Medians Finding Algorithm

https://brilliant.org/wiki/median-finding-algorithm/

- The problem, its inputs and outputs
- the algorithm you are presenting and an explanation of it. Why it is a divide and conquer algorithm. Main ideas. Simple high-level pseudocode
- time complexity in terms of time and space
- illustrate how your algorithm would work on a simple (but non-trivial) case

Description of the Median-of-Median Algorithm

The median-of-median algorithm is used to find the kth smallest element in an unsorted array. It allows to find the kth smallest element without sorting the entire input array. Direct sorting of the input array would take $O(nlogn)$ time.
The median-of-median algorithm allows to get the element in $O(n)$ time. It, receives as input, the unsorted array and, k, the rank of the element we are in search of, in the array. It then returns the element whose rank in the array, in ascending order from 1, is k.

My Implementation of the Median-of-Median Algorithm

I implemented the median-of-median algorithm by subdividing the input array into chunks of 5, with the last chunk being <= 5. I then "brute-sorted" each chunk and found the median. I then compared the medians found across all chunks to find the median amongst them. Using the overall median as a pivot, I iterated through the input array and partitioned it into two subarrays: arrays with elements less than pivot and arrays with elements greater than pivot.
If the pivot was at position, k, where k represents the length of the subarray with all numbers less than the pivot, then the kth smallest element = current pivot.
if not, then the function recursively calls itself on the left or right subarrays depending on whether k was greater or less than the current pivot. The act of partitioning the input array

into smaller parts and solving those makes the Median-of-Median algorithm a divide and conquer algorithm.
This algorithm is only guaranteed to work for arrays with unique elements because in cases where a pivot had a duplicate, it treats all duplicates as the same element.

```
def median_of_medians(list_l, k):
Let list_of_chunks be a list
for i=1 to len(list_l) with steps of 5:
    append list_l[i:i+5] to list_of_chunks
medians = []
for chunk in list_of_chunks:
    # sort chunk
    # append chunk's median to medians
if len(medians) <= 5:
    pivot = median in medians
else:
    pivot = median_of_medians(medians, len(medians)//2)

left = []
right = []
# append elements in list_l to left, if less than pivot or
# right, if greater than pivot
position = len(left)
if k < position:
    # call the function on the left side of the array
elif k > position:
    # call the function on the right side of the array, keeping in consideration
    # the position shift
else:
    # if k == position, then pivot is the kth smallest element
    return pivot
```

Time and Space Complexity of the Median-of-Medians Algorithm

The time complexity of the algorithm, as I implemented it, is $O(n)$. The operation of sectioning the input array into chunks take $O(n)$ because it involves iterating through the input array once.

Sorting each chunk is about $O(c)$, since 5 elements is a small list. Thus, the sorting portion yields $O(n)$ also. Similarly, partitioning the array into left and right subarrays takes $O(n)$ time. As such, the time complexity is a constant multiple of $O(n)$, which means the overall runtime complexity, in Big-O notation, is $O(n)$.

Sample Walkthrough with the Median-of-Medians Algorithm

Given an input array and k value:

```
[10, 8, 9, 6, 2, 1, 7, 4, 3], 3
Divide into chunks of five-- the last chunk can have smaller than 5 elements
[10, 8, 9, 6, 2]                    [1, 7, 4, 3]
We sort both chunks and compile their medians into a list
[2, 6, 8, 9, 10]                    [1, 3, 4, 7]
Medians = [8, 4]
We sort the medians and set the pivot to the median of the medians
[4, 8]
pivot = 8
Ordering the elements based on their sizes relative to the pivot,
left = [6, 2, 1, 7, 4, 3]          right = [10, 9]
the position of pivot relative to left and right is 6
6          >       3
Therefore, we iterate through the left subarray
[6, 2, 1, 7, 4, 3]
We perform the same operations:
left = [6, 2, 1, 7, 4]             right = [3]
Sorted: [1, 2, 4, 6, 7]                    [3]
Median = [4, 3]
Sorted: [3, 4]
Pivot = 4
left  = [1, 2, 3]                     right = [6, 7]
position of pivot = 3
3        =      3
Therefore, we return the pivot, 4.
4 is the 3rd smallest element in the array
```