3320 Programming Assignment 2

Tristan Fry

March 2023

1 The Algorithm

The algorithm is a divide and conquer approach to finding the k closest points to the origin. We make use of the median of medians algorithm in order to achieve linear runtime. The way the median of medina algorithm works is that the algorithm divides the input array into groups of 5 elements each, and then finds the median of each group. Then, the algorithm recursively applies the median of medians algorithm to find the median of medians. This median of medians becomes the pivot, which partitions the input list into three parts, elements to the left, equal, and to the right of the pivot. By dividing the input array into groups of five, the algorithm gives a worst case time complexity of $O(n)$.

2 Pseudocode

1: $Distances \leftarrow [\emptyset]$
2: **for** x and y values in S **do**
3: $\quad Distances \leftarrow x^2, y^2$
4: **end for**
5: **function** MEDIAN OF MEDIANS(S, K)
6: $\quad n \leftarrow length(S)$
7: $\quad sortedS \leftarrow sorted(S)$
8: $\quad$ **if** $n \leq 5$ **then**
9: $\quad\quad$ **return** sortedS
10: $\quad$ **end if**
11: $\quad$ Divide S into 5 groups
12: $\quad$ Create a new list of medians of each group
13: $\quad P \leftarrow$ MEDIAN OF MEDIANS(medinas, $length(medians)/2$)
14: $\quad Left \leftarrow elements < p$
15: $\quad Right \leftarrow elements > p$
16: $\quad Mid \leftarrow elements == p$
17: $\quad$ **if** $k \leq Length(Left)$ **then**
18: $\quad\quad$ **return** MEDIAN OF MEDIANS(Left, k)
19: $\quad$ **else if** $k \leq length(Left) + length(Mid)$ **then**
20: $\quad\quad$ **return** P

```
    else
        return MEDIAN OF MEDIANS(Right, k−length(Left)−length(Right))
    end if
    Smallest← MEDIAN OF MEDIANS(distances, k)
    R← [∅]
    for i in rnage of length(S) do
        if Distances[i] ≤ Smallest then
            R← points[i]
            if length(R) == k then
                Break
            end if
        end if
    end for
    return R
end function
```

3 Runtime

The median of median algorithm runs in $O(n)$ time because it uses a brute force method to find the median, as well as utilizing a constant factor, in this case I used 5. The worst case time complexity of the median of medians algorithm is O(n), because in the worst case, each recursive call partitions the list into a group of size 1 and a group of size n-1. This means that there are n levels of recursion, and each level takes O(n) time to compute. Thus, the total time complexity of the algorithm is O(n) + O(n) = O(n).

4 Correctness

We will use a mathematical induction proof in order to show correctness.

Base Case : Suppose $n = 1$. Since the algorithm returns the closest distance to the origin, if there is only one point in the input array, then this of course would be the closest distance to the origin.

InducticeStep : Suppose the algorithm works for $k = n$. We need to show that if we increment k by 1 , the algorithm returns $n + 1$ closest points to the origin.

To find the (n+1) closest point, we need to find the (n+1) smallest distance to the origin. Since the algorithm correctly finds the kth smallest distance using the median of medians algorithm. We can use the following steps to to find the (n+1) smallest distance:

1. Find the kth smallest distance using the median of medians algorithm.

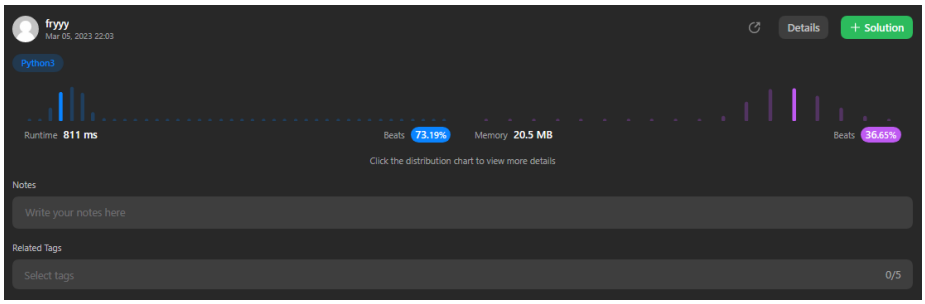
2. Collect all the points whose distance from the origin is less than or equal to the kth smallest distance.

3. If the number of collected points is greater than or equal to n + 1, then return the first n + 1 points in the collection

4. Else, repeat the above steps with the remaining points

Thus, The proof of correctness by induction is complete.

5 Leetcode Submission



6 Code

```
class Solution:
def kClosest(self, points, k):
    distances = []
    for x, y in points:
        distances.append(x**2 + y**2)

    def median_of_medians(S, k):
        n = len(S)
        if n <= 5:
            return sorted(S)[k-1]

        groups = []
        for i in range(0, n, 5):
            groups.append(sorted(S[i:i+5]))
        medians = []
        for group in groups:
            medians.append(group[len(group) // 2])
        pivot = median_of_medians(medians, len(medians) // 2)

        left = []
        right = []
        mid = []
```

```
    for x in S:
        if x < pivot:
            left.append(x)
        elif x > pivot:
            right.append(x)
        else:
            mid.append(x)

    if k <= len(left):
        return median_of_medians(left, k)
    elif k <= len(left) + len(mid):
        return pivot
    else:
        return median_of_medians(right, k - len(left) - len(mid))

smallest = median_of_medians(distances, k)

result = []
for i in range(len(points)):
    if distances[i] <= smallest:
        result.append(points[i])
        if len(result) == k:
            break

return result
```