Largest Minimum Distance

B. Chetan Rao
IIB2019018

Kalpana Khutela
IIB2019019

Devang Bharti
IIB2019020

***Abstract*—In this report we used a divide and conquer algorithm to find the largest minimum distance.and then we also calculated the time and space complexity**

Keyword : longest minimum distance, space complexity, time complexity.

I. Introduction

In this problem, we have to find the largest minimum distance for given k such k¡=n where n is length of array.

For example: if we have arr of 1 2 5 9 and k=2 then largest of all possible minimum distances will be 8

we have given a brief idea of our algorithm **part II**.

II. Algorithm Description and Analysis

1) The algorithm asks for the size of array 'n' as an input.

2) Now the algorithm asks for n integers which are now stored in an array.

3) Now we used the an application of divide and conquer approach named binary search to solve this problem.

4) First sort the array. Maximum possible value result will be equal to a[n-1] –a[0] (for k = 2) and lower bound l = 1 and not arr[0] (because minimum distance between each element can be one) and upper bound r = arr[n-1] (where n is the size of array arr).

5) We start with the middle of maximum possible result. If middle is a feasible solution, we search on right half of mid. Else we search is left half. To check feasibility, we place k elements under given mid-distance.

6) If total number of elements found that can be placed at a minimum distance equal to mid, is greater than or equal to k we return mid as a possible answer and increase left = mid + 1 .

7) If total number of elements found that can be placed at a minimum distance equal to mid, is less than k we return false and in that case right= mid-1.

For Example -

Let input be n = 6 and k = 3 and array be: 1,2,7,5,11,12

After sorting the given array : 1,2,5,7,11,12
Here, left = 1 and right =12

Applying binary search :

mid = (12+1)/2 = 6
Number of points possible with distance 6 equal to 2, So result = 5 and right= 6-1=5 and left = 1

mid = (5+1)/2 = 3
Number of points possible with distance 3 equal to 3, So left = 3 and right = 5

mid= (3+5)/2 =4
Number of points possible with distance equal to 3, So left = 4 and right = 5

Then for mid =5 number of possible points is 3 and it is the maximum possible answer.

We calculate the longest common prefix of the two strings, which comes out to be "Ind" and return it.

III. Pseudo Code

Function int-minimum-distance
Pass : int arr[ ], int n, int k

$sort(a)$
$intresult = -1$
$intleft = 1$
$intresult = arr[n-1]$
while $left < right$ **do**
  $mid \leftarrow (left + right)/2$
  if $Is - feasible(mid, arr, n, k)$ **then**
    $res = max(res, mid)$
    $left = mid + 1$
  else
    $right = mid$
    return res

IV. Pseudo Code II

Function Is-Feasible
Pass : int mid[ ], int arr, int n, int k
  $intposition = arr[0]$

```
    elements = 1 i ← 0  to  n
    if arr[i] − position >= mid then
      position = arr[i]
      elements + +
      if elements == k then
        return 1
      end if
    end if
  end for
  return 0
```

V. Time Complexity

The overall complexity of the question is O(n*log-n). For Sorting the given array time complexity will be O(n*log-n). Our algorithm uses binary search to find the maximum distance possible for given value of k and the complexity of binary search is O(log n). To check the feasibility for the given value of k and mid value ,we will traverse the array .Time complexity in worst case will be O(n). Overall time complexity = O(n*log-n)

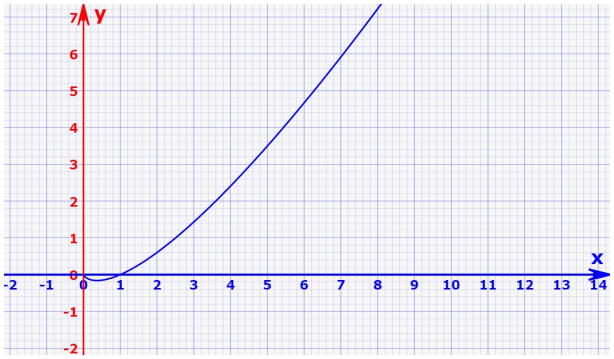

Fig. 1. Time complexity curve

VI. Auxiliary Space Complexity

No extra space is used in this algorithm , so auxiliary space is constant.Only the input array is of size n. Space Complexity = Input Space + Auxiliary Space ,which in turn equal to O(n).

Fig. 2. Auxiliary space complexity curve

VII. Conclusion

The above proposed algorithm efficiently gives the maximum possible minimum distance between two consecutive elements for k elements placed on given n distances on a horizontal line. The algorithm proposed is very efficient both space and time wise with O(n*log-n) time complexity.

VIII. References

1) Introduction to divide and conquer algorithm
2) Tutorials point