Brute Force

def minimiseMaxDistance(arr: [int], k: int) -> float:

    initial\_slots=[0]\*(len(arr)-1)

    for j in range(1,k+1):

        index=-1

        slot\_to\_mini=-1

        for i in range(len(arr)-1):

            diff=arr[i+1]-arr[i]

            dist=diff/(initial\_slots[i]+1)

            if dist>slot\_to\_mini:

                slot\_to\_mini=dist

                index=i

        initial\_slots[index]+=1

    maxi=-float('inf')

    #because i can just calculate the length of each divided slot which is in initial\_slots

    for i in range(len(arr)-1):

        diff=arr[i+1]-arr[i]

        dist=diff/(initial\_slots[i]+1)

        maxi=max(dist,maxi)

    return maxi

Optimal

def returnk(mid,arr):

    k=0

    for i in range(len(arr)-1):

        dist=arr[i+1]-arr[i]

        a=dist//mid

        if a\*mid==arr[i+1]-arr[i]:

            a-=1

        k+=a

    return k    #we are returning slots

def minimiseMaxDistance(arr: [int], k: int) -> float:

    n=len(arr)

    low=0

    high=0

    for i in range(n-1):

        high=max(high,arr[i+1]-arr[i])

    while(high-low>1e-6):

        #cause given constraint

        mid=(high+low)/2.0

        a=returnk(mid,arr)

        if a>k:

            low=mid

        else:

            high=mid

    return high