**Problem Statement:**

To Implement and Solve Dinner Ball Problem using Data Structure and Sorting Algorithm.

**Solution**: This can be resolved by using sorting algorithm by storing the heights of M boys and N girls into an array and comparing the values. Following point, we considered.

**General Block Diagram**:

**Input** **Process** **Output**

Print YES incase every boy finds a girls shorter than him, else print NO. (outputPS10.txt)

Sort the heights of Boys and Girls. Then compare the height of each boy with that of girls.

Input File With total number of test cases, M boys and N Girls with heights (inputPS10.txt)

**Detailed Design:**

* **Input File Read:**

The program is reading the external file **inputPS10.txt** line by line as input to start the process. The first line will give the total number of test cases**(t)**. Each Test case contains three lines. The first line gives the Number boys **m** and number girls **n** that are invited for the party. The second line contains height of **m** boys. The third line contains the heights on **n** girls.

Each of these lines is read and split based on the space separation and converted into integers from strings. The values of heights are taken as list of integers for further processing.

Here a list is considered as it is easy to sort the same and since it is indexed, the comparison also becomes easy.

* **Main Logic:**

If the number of girls is less that the number of boys, NO is returned without any further processing as there is no way that all boys will get a girl. Otherwise, YES id returned by default. Post this, the heights of boys and girls are sorted using a sorting function and individual height values are now compared such that the height of a boy must be greater than the height of the girl. NO is returned in case this condition is not met.

* **Output File Write:**

The returned value either YES or NO is written into the output file **outputPS10.txt** for each Test Case.

* **Time Complexity Analysis:**

class MoonLight :

def main( args):

file1 = open("inputPS10.txt","r") O(1)

file2 = open("outputPS10.txt","w") O(1)

t = int(file1.readline()) O(1)

for i in range(t): O(t)

counts = file1.readline() O(1)

counts\_split = counts.split() O(1)

m = int(counts\_split[0]) O(1)

n = int(counts\_split[1]) O(1)

boys = [0] \* (m) O(1)

girls = [0] \* (n) O(1)

m\_values = file1.readline() O(1)

m\_values\_split = m\_values.split() O(1)

for j in range(m): O(m-1)

boys[j] = int(m\_values\_split[j])

n\_values = file1.readline() O(1)

n\_values\_split = n\_values.split() O(1)

for k in range(n): O(n-1)

girls[k] = int(n\_values\_split[k])

if (n < m) : O(1)

print("NO") O(1)

file2.write("NO \n") O(1)

else :

boys = MoonLight.sortHeight(boys, 200)

girls = MoonLight.sortHeight(girls, 200)

girlForEachBoy = True O(1)

for l in range(m): O(m-1)

if boys[l] <= girls[l]:

girlForEachBoy = False

if (girlForEachBoy) : O(1)

print("YES") O(1)

file2.write("YES \n") O(1)

else :

print("NO") O(1)

file2.write("NO \n") O(1)

file1.close() O(1)

file2.close() O(1)

def sortHeight(input, maxValue):

freq = [0] \* (maxValue + 1) O(1)

i = 0 O(1)

while (i < len(input)): O(i)

freq[input[i]] += 1

i += 1

totalCount = 0 O(1)

i = 0 O(1)

while (i <= maxValue): O(i)

oldCount = freq[i]

freq[i] = totalCount

totalCount += oldCount

i += 1

output = [0] \* (len(input)) O(1)

i = 0 O(1)

while (i < len(input)): O(i)

output[freq[input[i]]] = input[i]

freq[input[i]] += 1

i += 1

return output O(1)

if \_\_name\_\_=="\_\_main\_\_":

MoonLight.main([])

The time complexity for this program is dependent on n, i and few constant values. Hence, ignoring the smaller values like constants and i, it can be said that its time complexity is O(n).