## Practical 4

### **Aim:** Implement hiring problem and analyze its complexity

**Theory:**

1. If a sample is too small, we don’t get information enough for setting the benchmark for remaining candidates.
2. If a sample is too large, though we get plenty of information but we have also burned too many of the potential candidates. This leaves us with very few candidates to choose from, and hence making the strategy a poor one.
3. The best strategy is to choose the perfect or optimal sample size (ideal sample size) which can be done using 1/e law that is rejecting n/e candidates (this n/e is the sample size).

**Algorithm:**

HIRE-ASSISTANT(n)

1. best = 0 // candidate 0 is a least-qualified dummy candidate
2. for i = 1 to n
3. interview candidate i
4. if candidate I is better than candidate best
5. best = i
6. hire candidate i

RANDOMIZED-HIRE-ASSISTANT(n)

1. randomly permute the list of candidates
2. best = 0 // candidate 0 is a least-qualified dummy candidate
3. for i = 1 to n
4. interview candidate i
5. if candidate I is better than candidate best
6. best = i
7. hire candidate i

**Code:**

import random

import math

e = 2.71828;

# To find closest integer of num.

def roundNo(num):

if(num < 0):

return (num - 0.5)

else:

return (num + 0.5);

# Finds best candidate using n/e rule.

# candidate[] represents talents of n candidates.

def printBestCandidate(candidate, n):

# Calculating sample size for benchmarking.

sample\_size = roundNo(n / e);

print("\n\nSample size is",

math.floor(sample\_size));

# Finding best candidate in sample size

best = 0;

for i in range(1, int(sample\_size)):

if (candidate[i] > candidate[best]):

best = i;

# Finding the first best candidate that

# is better than benchmark set.

for i in range(int(sample\_size), n):

if (candidate[i] >= candidate[best]):

best = i;

break;

if (best >= int(sample\_size)):

print("\nBest candidate found is",

math.floor(best + 1),

"with talent", math.floor(candidate[best]));

else:

print("Couldn't find a best candidate");

# Driver code

n = 8;

# n = 8 candidates and candidate

# array contains talents of n

# candidate where the largest

# number means highest talented

# candidate.

candidate = [0] \* (n);

# generating random numbers between 1 to 8

# for talent of candidate

for i in range(n):

candidate[i] = 1 + random.randint(1, 8);

print("Candidate : ", end = "");

for i in range(n):

print((i + 1), end = " ");

print("\nTalents : ", end = "");

for i in range(n):

print(candidate[i], end = " ");

print("Neeraj Appari 021")

printBestCandidate(candidate, n);

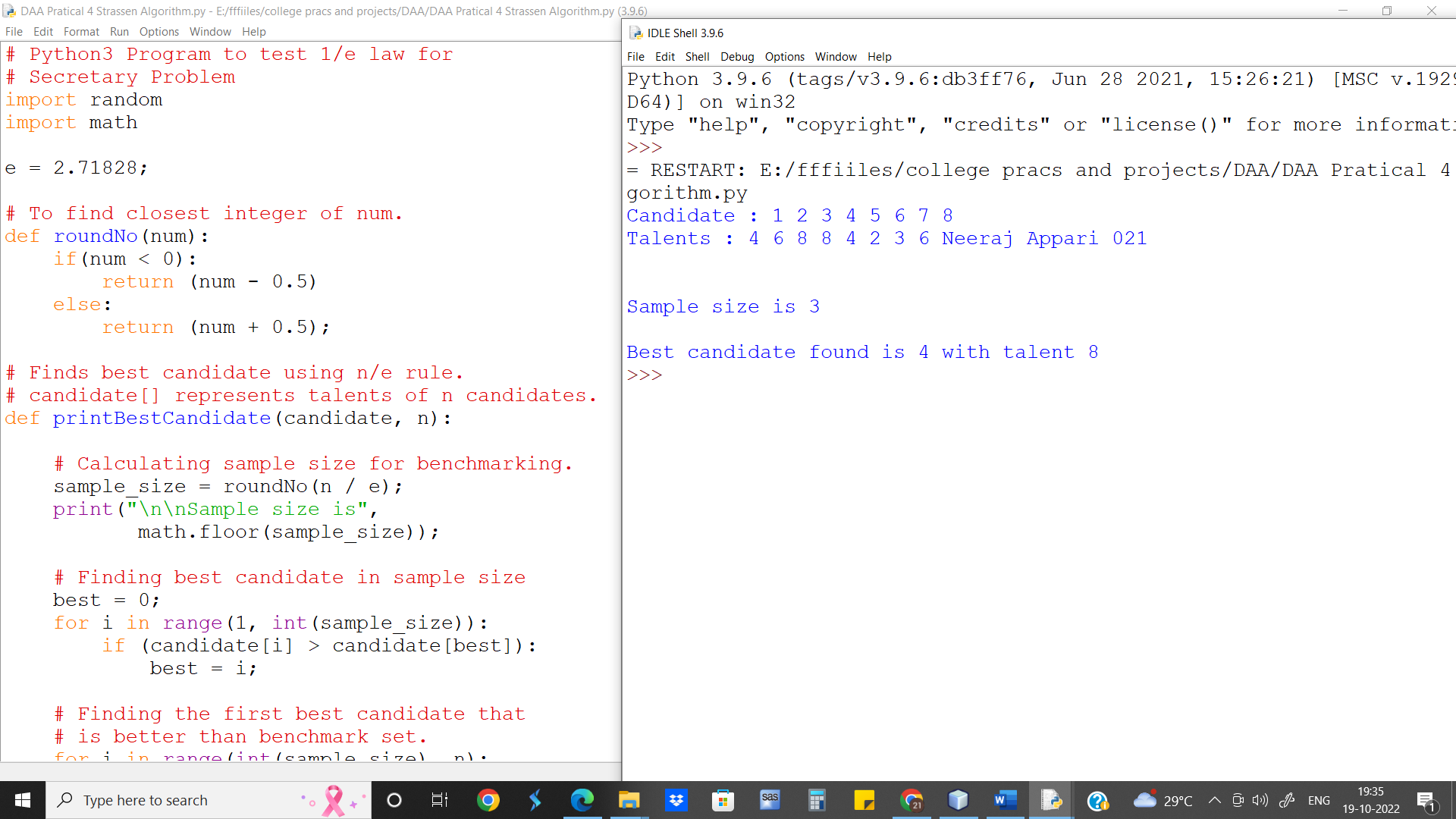
**Output**:

Candidate : 1 2 3 4 5 6 7 8

Talents : 4 6 8 8 4 2 3 6 Neeraj Appari 021

Sample size is 3

Best candidate found is 4 with talent 8



**Conclusion:** The Optimal Strategy doesn’t always find the best candidate but selects the almost best candidates most of the times.

The worst-case runtime complexity of Insertion Sort is O(chn).