CS747: Foundations of Intelligent and Learning Agents

Programming Assignment #1

Report

9 Sep, 2021

Deepanshu (190050032)

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
2021-2022

Contents

Abstract

In this Assignment we try to analyse different algorithms in python. We analyse then basically on what regret they give on different parameters. Here algorithms include epsilon-greedy, ucb, kl-ucb, thompson-sampling, and some of their other variants.
All outputs are stored in outputData.txt which are reproducible and can be tested by anyone.

1. Task1

1.1 Epsilon-greedy

Draw random number from uniform distribution. If less than epsilon, explore the bandits. Else, exploit the bandits (based on emperical mean).

Experiment was run for :-

Instance = [1,2,3]
RandomSeed = [0,...,49]
Horizon = [100,400,1600,6400,25600,102400]

1.2 UCB

Here first we pull each arm once. After that, calculate ucb of each arm and the arm with highest ucb will get pulled.

Experiment was run for :-

Instance = [1,2,3]
RandomSeed = [0,...,49]
Scale = [2]
Horizon = [100,400,1600,6400,25600,102400]

1.3 KL-UCB

Similar to ucb, pull each arm once and then calculate q value for kl-ucb (using binary search with margin = 1e-5 in my case). Then the arm with highest q value will get pulled. Kl-ucb uses tighter lower bound and gives lower regret than ucb which can be observe in the graphs below.

Experiment was run for :-

Instance = [1,2,3]
RandomSeed = [0,...,49]
Scale = [2]
Horizon = [100,400,1600,6400,25600,102400]

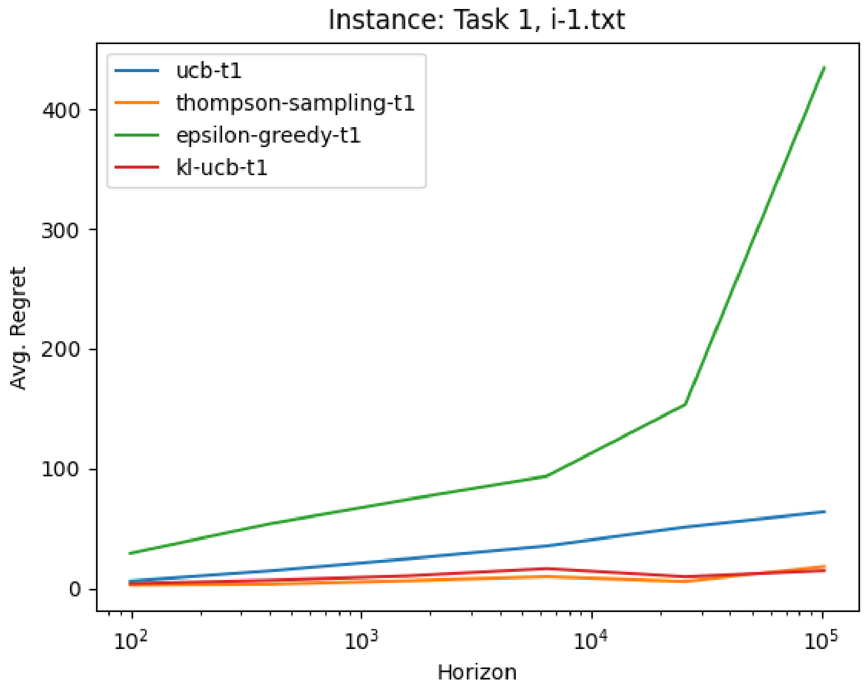
1.4 Thompson-Sampling

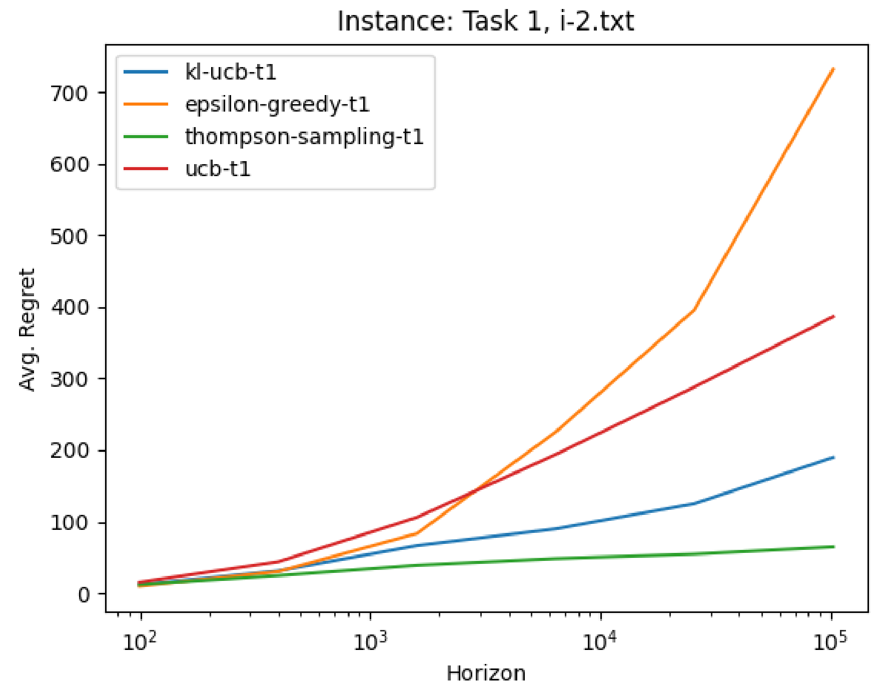
In thompson sampling we are dependent on Beta distibution to give us the arm to pull and the parameters Beta distribution takes is the number of success and failures of an arm. The arm with highest Beta value will get pulled.

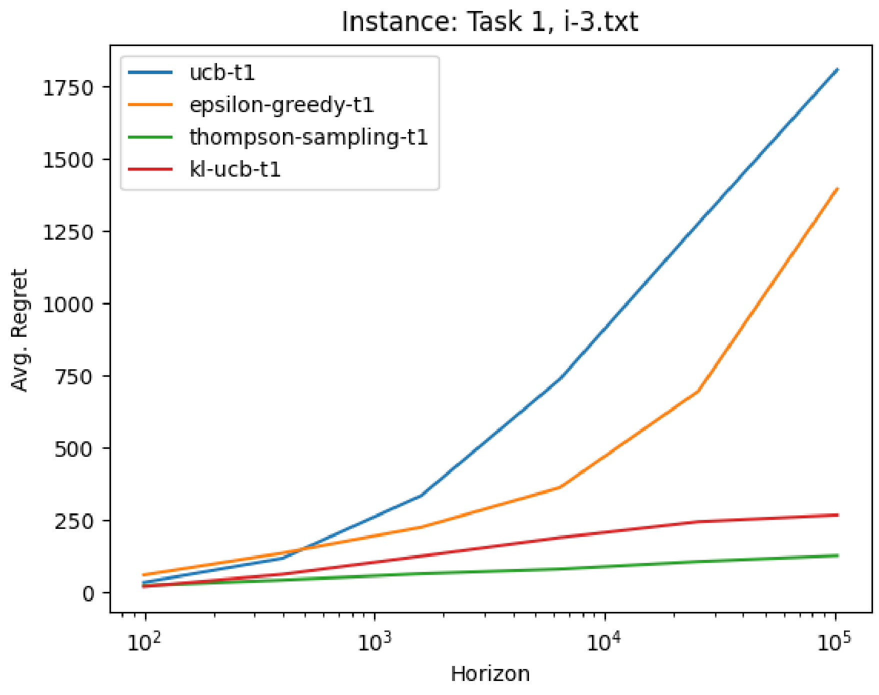
Experiment was run for :-

Instance = [1,2,3]
RandomSeed = [0,...,49]
Horizon = [100,400,1600,6400,25600,102400]

Graphs for Task-1







Some general observations are, UCB perform worst on instance-3. kl-ucb performs better than ucb on all 3 instances. In general, Thompson-sampling performs best among all.

2. Task2

In this we try to find c value for ucb to reduce the regret. In Task 1 the c value was 2 but in Task 2 we increase value of c from 0.02 to 0.3, and average regret in general decrease.

Experiment was run for :-

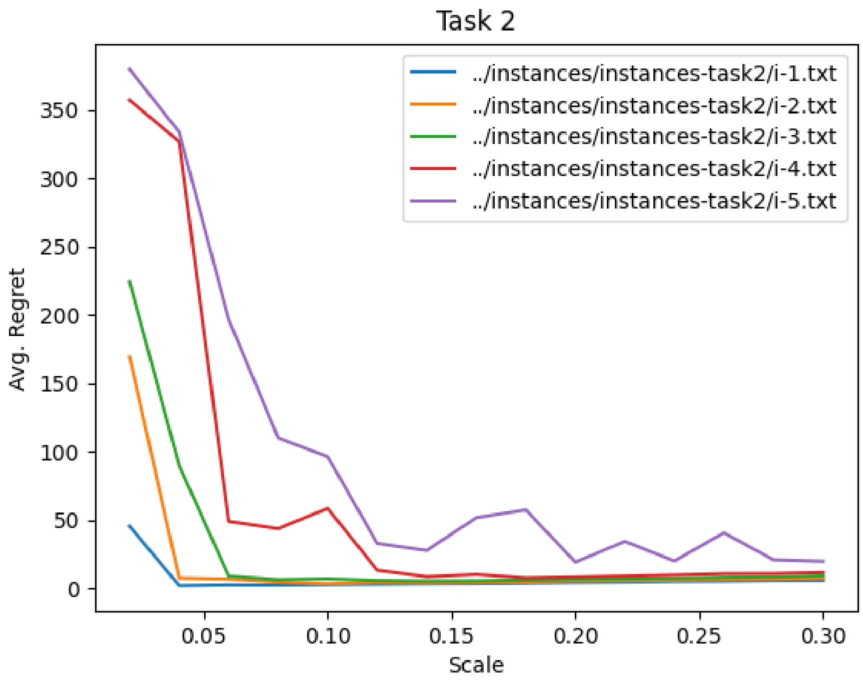
Instance = [1,2,3,4,5]
RandomSeed = [0,...,49]
Scale = [0.02,0.04,0.06,...,0.3]
Horizon = [10000]

Here is the trend :-

Instance	C for min Avg. Regret	Min. Avg. Regret
1	0.04	2.04
2	0.1	3.26
3	0.139	5.3
4	0.18	7.64
5	0.199	19.2

For low c values, ucb is almost equal to the Empirical mean of the reward of each arm. But as we increase the c values our estimate of ucb improves.
Things we can better observe from the graph.

Graph for Task-1


Task 2
Avg. Regret
Scale
../instances/instances-task2/i-1.txt
../instances/instances-task2/i-2.txt
../instances/instances-task2/i-3.txt
../instances/instances-task2/i-4.txt
../instances/instances-task2/i-5.txt
0
50
100
150
200
250
300
350
0.05
0.10
0.15
0.20
0.25
0.30

3. Task3

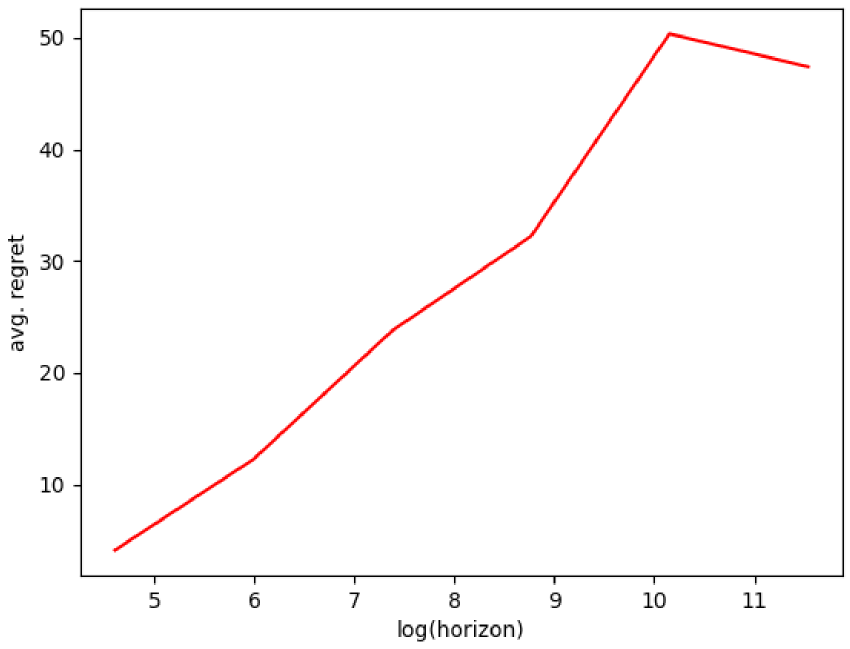
In this task I have used Thompson-Sampling algorithm. We draw samples from beta distribution just like we did in 1st task but here the reward is fractional so we add fractional reward to the success of a bandit and 1-reward to the faiure of a bandit. Also we chose the most optimal arm here by expected reward of that arm (used in the calculation of total regret) . Going for Thompson-Sampling in Task3 was a good decision because in the end it gave better regret than any other algorithm of task-1.

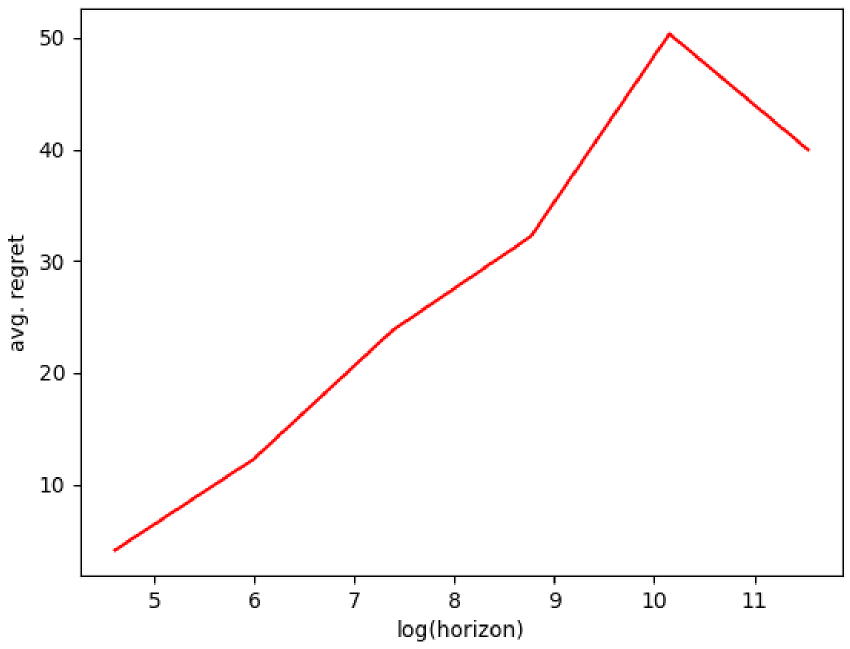
Experiment was run for :-

Instance = [1,2,3]
RandomSeed = [0,...,49]
Horizon = [100,400,1600,6400,25600,102400]

We can see the graphs for regret details here :-

Graphs for Task-3


50
40
30
20
10
avg. regret
5
6
7
8
9
10
11
log(horizon)


50
40
30
20
10
avg. regret
5
6
7
8
9
10
11
log(horizon)

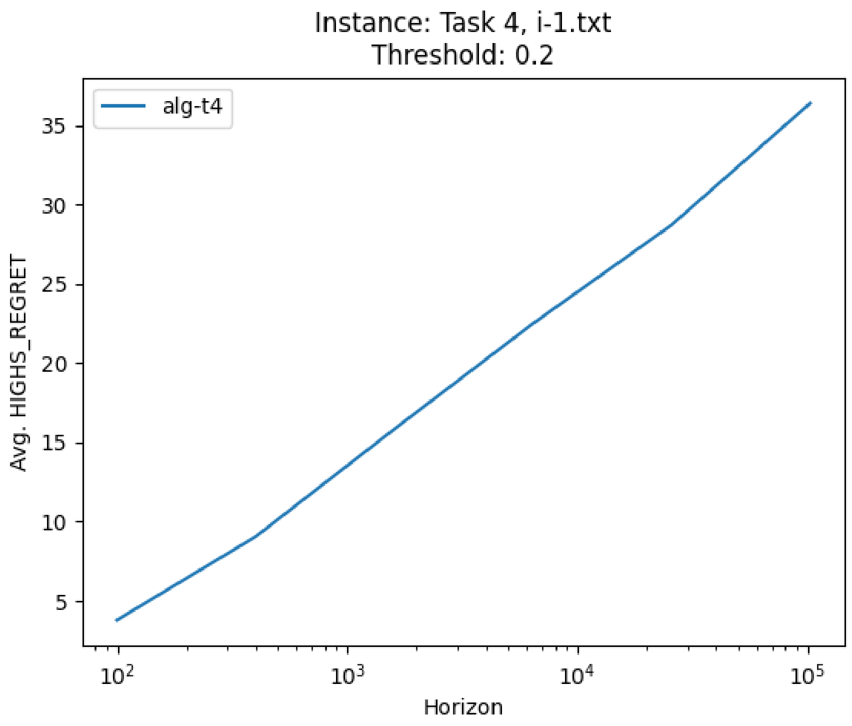
4. Task4

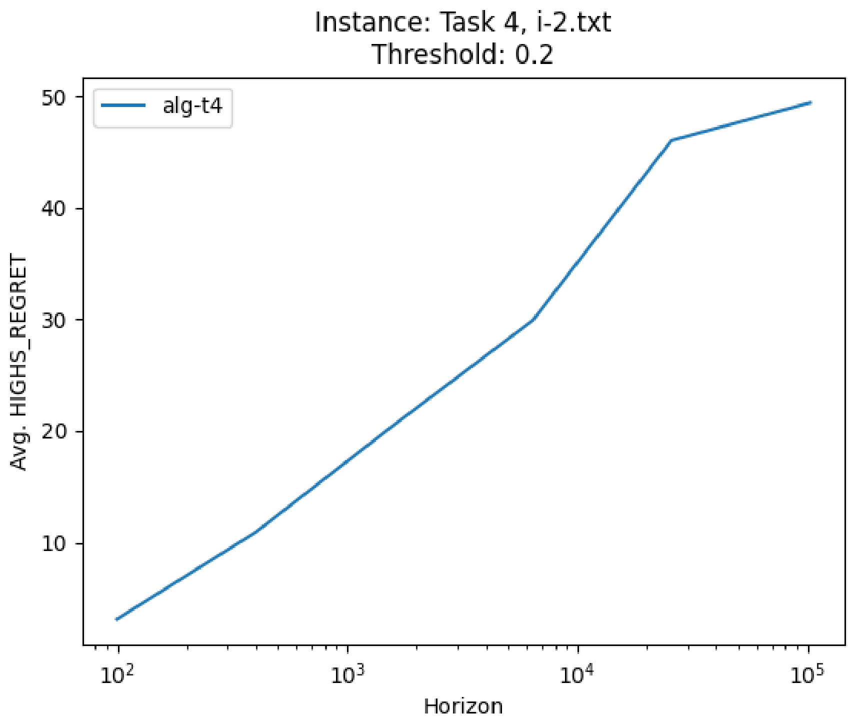
For Task4 I have used Thompson-Sampling with slight modifications.The underlying distribution remians the same, i.e., Beta. But rewards become fractional (same as task3, reward goes to bandits success and 1-reward goes to bandits failure). So we observe reward for each pull and this reward will contribute one to the final total reward if greater than or equal to the threshold, otherwise the contribution would be zero.
We can see the details in the grahps below :-

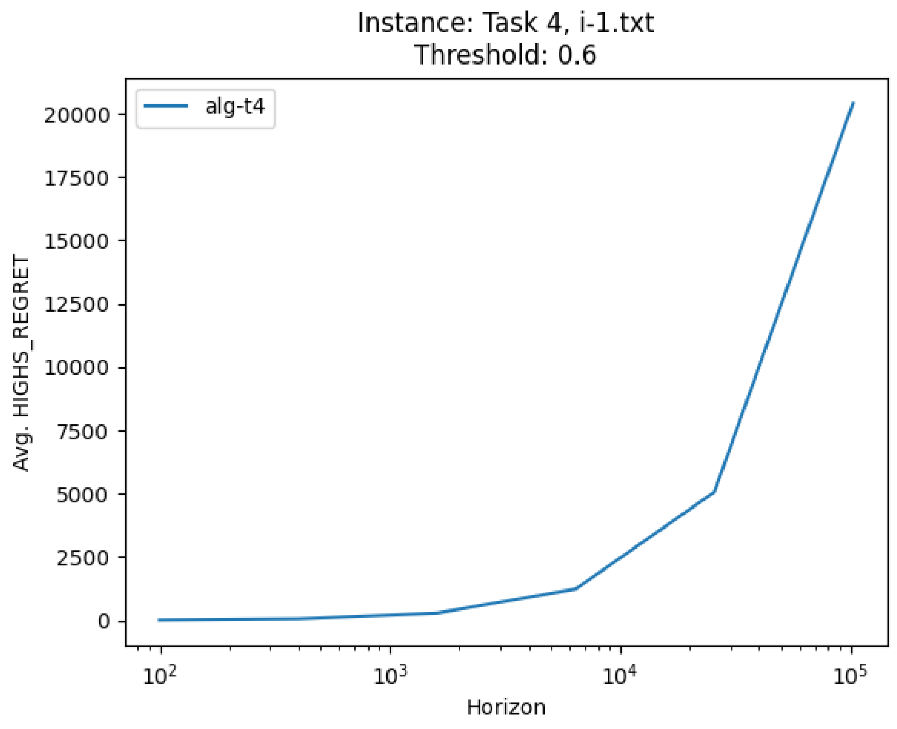
Experiment was run for :-

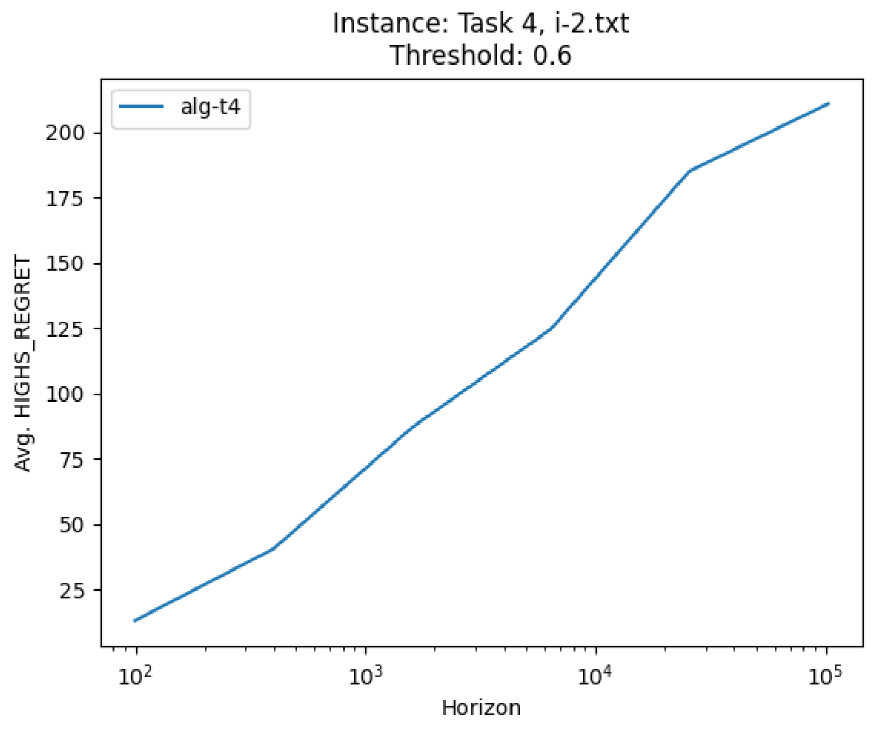
Instance = [1,2,3]
RandomSeed = [0,...,49]
Threshold = [0.2, 0.6]
Horizon = [100,400,1600,6400,25600,102400]

Graphs for Task-4


Instance: Task 4, i-1.txt
Threshold: 0.2
alg-t4
Avg. HIGHS_REGRET
35
30
25
20
15
10
5
10^2
10^3
10^4
10^5
Horizon


Instance: Task 4, i-2.txt
Threshold: 0.2
alg-t4
Avg. HIGHS_REGRET
50
40
30
20
10
10^2
10^3
10^4
10^5
Horizon


Instance: Task 4, i-1.txt
Threshold: 0.6
alg-t4
Avg. HIGHS_REGRET
Horizon
0
2500
5000
7500
10000
12500
15000
17500
20000
10^2
10^3
10^4
10^5


Instance: Task 4, i-2.txt
Threshold: 0.6
alg-t4
Avg. HIGHS_REGRET
Horizon
25
50
75
100
125
150
175
200
10^2
10^3
10^4
10^5