**SIDHANTA BARIK**

**2241002049**

1. **OUTPUT: -**

Name: Sidhanta Barik, RegNo: 2241002049

-------------------------------------------------------

Sum of first 10 natural nos. using Recursion = 55

Sum of first 10 natural nos. using Loop = 55

Base Case: When n is 1, return 0

Recursive Step: Return n + recursive\_sum(n-1)

Time complexity of recursive\_sum(n) is O(n) because it performs n recursive calls.

Time complexity of iterative\_sum(n) is O(n) because it performs n iterations in the loop.

*Recursion* is less efficient in this case compared to iteration because each recursive call creates a new stack frame, which consumes memory and CPU time.

In contrast, *iteration* uses a loop, which is more memory-efficient and faster.

1. **OUTPUT: -**

Name: Sidhanta Barik, RegNo: 2241002049

-------------------------------------------------------

# n = 5 and k = 2

Last Person Remaining using Recursion: 2

Last Person Remaining using Loop: 2

# n = 7 and k = 3

Last Person Remaining using Recursion: 3

Last Person Remaining using Loop: 3

Base Case: When n is 1, return 0

Recursive Formula: (josephus(n-1, k) + k) % n

* Time complexity of josephus(n, k) is O(n) because it performs n recursive calls.
* Time Complexity of josephus\_iterative(n, k) is O(n) because it performs n iterations in the loop, but, it is more efficient than the recursive version because it does not create new stack frames for each iteration.

1. **OUTPUT: -**

Name: Sidhanta Barik, RegNo: 2241002049

-------------------------------------------------------

# b = 1000000 cannot be used because python cannot print the answer of 2^1000000 & throws a ValueError.

2^10000 (Using Recursion):-

19950631168807583848837421626835850838234968318861924548520089498529438830221946631919961684036194597899331129423209124271556491349413781117593785932096323957855730046793794526765246551266059895520550086918193311542508608460618104685509074866089624888090489894838009253941633257850621568309473902556912388065225096643874441046759871626985453222868538161694315775629640762836880760732228535091641476183956381458969463899410840960536267821064621427333394036525565649530603142680234969400335934316651459297773279665775606172582031407994198179607378245683762280037302885487251900834464581454650557929601414833921615734588139257095379769119277800826957735674444123062018757836325502728323789270710373802866393031428133241401624195671690574061419654342324638801248856147305207431992259611796250130992860241708340807605932320161268492288496255841312844061536738951487114256315111089745514203313820202931640957596464756010405845841566072044962867016515061920631004186422275908670900574606417856951911456055068251250406007519842261898059237118054444788072906395242548339221982707404473162376760846613033778706039803413197133493654622700563169937455508241780972810983291314403571877524768509857276937926433221599399876886660808368837838027643282775172273657572744784112294389733810861607423253291974813120197604178281965697475898164531258434135959862784130128185406283476649088690521047580882615823961985770122407044330583075869039319604603404973156583208672105913300903752823415539745394397715257455290510212310947321610753474825740775273986348298498340756937955646638621874569499279016572103701364433135817214311791398222983845847334440270964182851005072927748364550578634501100852987812389473928699540834346158807043959118985815145779177143619698728131459483783202081474982171858011389071228250905826817436220577475921417653715687725614904582904992461028630081535583308130101987675856234343538955409175623400844887526162643568648833519463720377293240094456246923254350400678027273837755376406726898636241037491410966718557050759098100246789880178271925953381282421954028302759408448955014676668389697996886241636313376393903373455801407636741877711055384225739499110186468219696581651485130494222369947714763069155468217682876200362777257723781365331611196811280792669481887201298643660768551639860534602297871557517947385246369446923087894265948217008051120322365496288169035739121368338393591756418733850510970271613915439590991598154654417336311656936031122249937969999226781732358023111862644575299135758175008199839236284615249881088960232244362173771618086357015468484058622329792853875623486556440536962622018963571028812361567512543338303270029097668650568557157505516727518899194129711337690149916181315171544007728650573189557450920330185304847113818315407324053319038462084036421763703911550639789000742853672196280903477974533320468368795868580237952218629120080742819551317948157624448298518461509704888027274721574688131594750409732115080498190455803416826949787141316063210686391511681774304792596709376

2^10000 (Using Loop):-

19950631168807583848837421626835850838234968318861924548520089498529438830221946631919961684036194597899331129423209124271556491349413781117593785932096323957855730046793794526765246551266059895520550086918193311542508608460618104685509074866089624888090489894838009253941633257850621568309473902556912388065225096643874441046759871626985453222868538161694315775629640762836880760732228535091641476183956381458969463899410840960536267821064621427333394036525565649530603142680234969400335934316651459297773279665775606172582031407994198179607378245683762280037302885487251900834464581454650557929601414833921615734588139257095379769119277800826957735674444123062018757836325502728323789270710373802866393031428133241401624195671690574061419654342324638801248856147305207431992259611796250130992860241708340807605932320161268492288496255841312844061536738951487114256315111089745514203313820202931640957596464756010405845841566072044962867016515061920631004186422275908670900574606417856951911456055068251250406007519842261898059237118054444788072906395242548339221982707404473162376760846613033778706039803413197133493654622700563169937455508241780972810983291314403571877524768509857276937926433221599399876886660808368837838027643282775172273657572744784112294389733810861607423253291974813120197604178281965697475898164531258434135959862784130128185406283476649088690521047580882615823961985770122407044330583075869039319604603404973156583208672105913300903752823415539745394397715257455290510212310947321610753474825740775273986348298498340756937955646638621874569499279016572103701364433135817214311791398222983845847334440270964182851005072927748364550578634501100852987812389473928699540834346158807043959118985815145779177143619698728131459483783202081474982171858011389071228250905826817436220577475921417653715687725614904582904992461028630081535583308130101987675856234343538955409175623400844887526162643568648833519463720377293240094456246923254350400678027273837755376406726898636241037491410966718557050759098100246789880178271925953381282421954028302759408448955014676668389697996886241636313376393903373455801407636741877711055384225739499110186468219696581651485130494222369947714763069155468217682876200362777257723781365331611196811280792669481887201298643660768551639860534602297871557517947385246369446923087894265948217008051120322365496288169035739121368338393591756418733850510970271613915439590991598154654417336311656936031122249937969999226781732358023111862644575299135758175008199839236284615249881088960232244362173771618086357015468484058622329792853875623486556440536962622018963571028812361567512543338303270029097668650568557157505516727518899194129711337690149916181315171544007728650573189557450920330185304847113818315407324053319038462084036421763703911550639789000742853672196280903477974533320468368795868580237952218629120080742819551317948157624448298518461509704888027274721574688131594750409732115080498190455803416826949787141316063210686391511681774304792596709376

Base Case: if b is 0, return 1

Recursive Step1: if b is even, return fast\_power(a, b/2) \*\* 2

Recursive Step2: if b is odd, return fast\_power(a, b-1)

* Time complexity of the standard recursive method is O(b) because it performs b recursive calls.
* Time complexity of the optimized exponentiation by squaring method is O(log b) because it reduces the number of recursive calls significantly.

time complexity of fast\_power\_iterative(): O(log b) because it performs log b iterations in the loop.

Performance of the iterative method is better than the recursive method because it does not create new stack frames for each iteration.

**Phantom\_Tracking**

**OUTPUT: -**

Name: Sidhanta Barik, RegNo: 2241002049

-------------------------------------------------------

High Value Transactions: [300, 600, 800, 500, 350]

Sorted Transactions: [300, 350, 500, 600, 800]

500 is found

Sorted Transaction History: [40, 45, 75, 90, 120, 130, 220, 300, 350, 500, 600, 800]