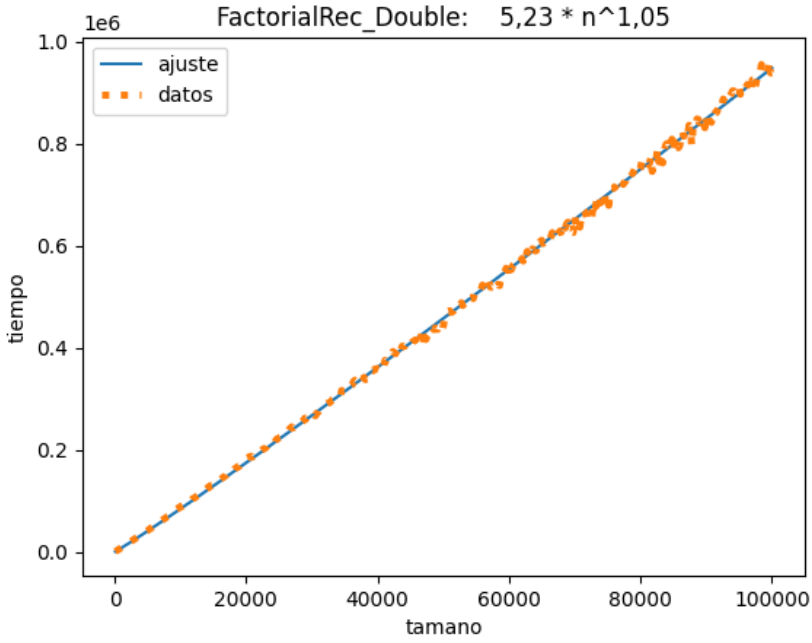
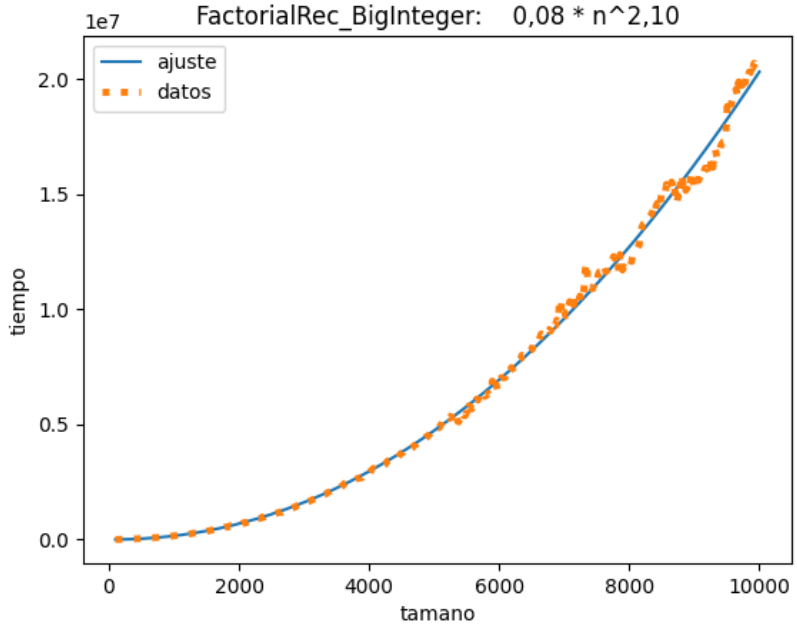
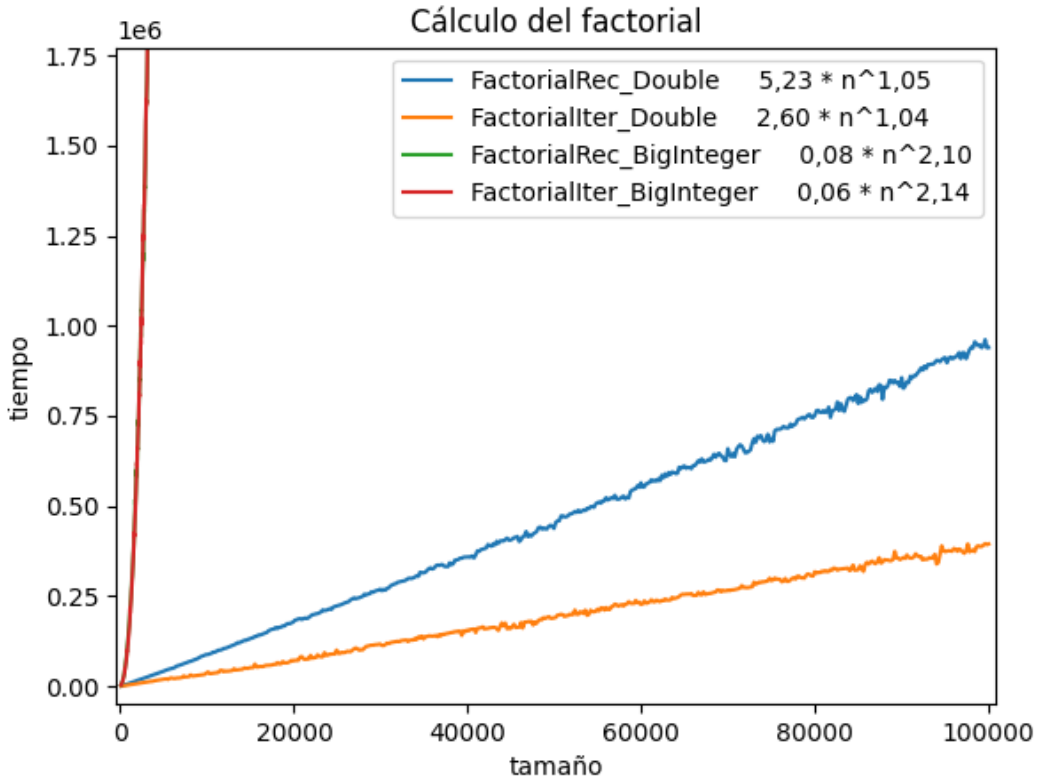
Ejercicio 1.

En este ejercicio se debe analizar la complejidad del cálculo del factorial de un número en versiones recursiva e iterativa, usando tres tipos distintos: Long, Double y Big Integer.

Los resultados deben mostrar que la complejidad de la función es de orden lineal al usar tipo Double o Long, y cuadrático al usar BigInteger. Se muestra un ejemplo de la versión recursiva para Double y BigInteger. La comparativa entre los distintos tipos mostrará la mayor complejidad al usar BigInteger.

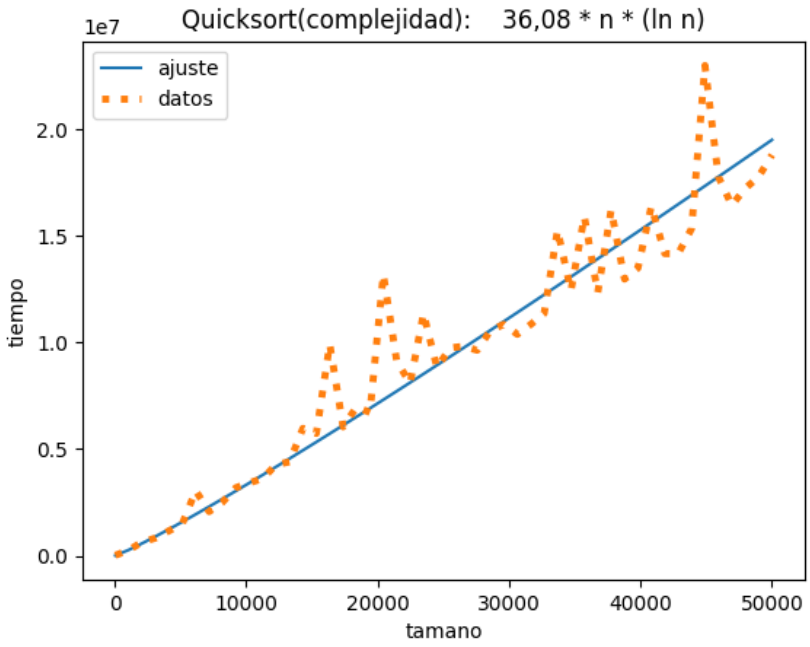




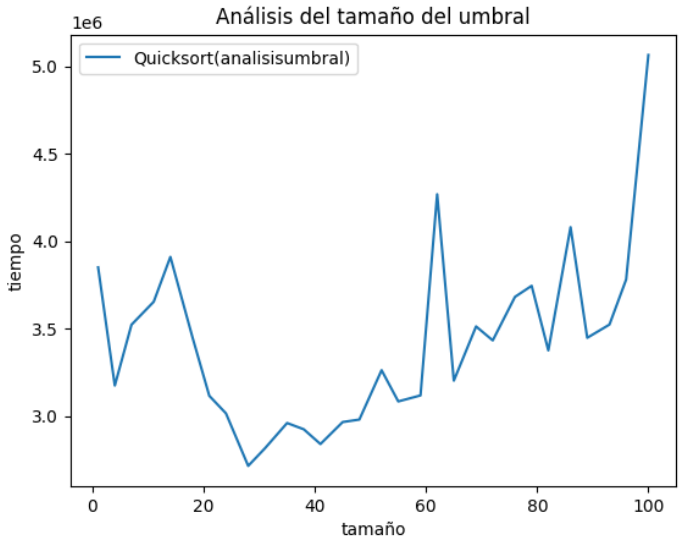

Cálculo del factorial
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
tiempo
0
20000
40000
60000
80000
100000
tamaño
FactorialRec_Double 5,23 * n^1,05
FactorialIter_Double 2,60 * n^1,04
FactorialRec_BigInteger 0,08 * n^2,10
FactorialIter_BigInteger 0,06 * n^2,14

Ejercicio 2.

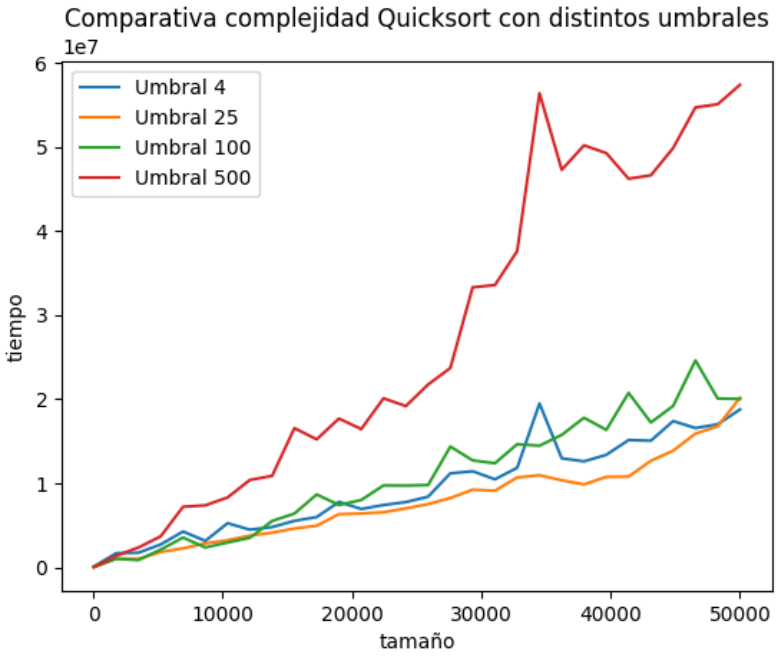
El algoritmo Quicksort tiene complejidad n log(n) en el caso medio. Se muestra un ejemplo del resultado esperado usando distintos tamaños de listas con un umbral de caso base fijo (4):



Por otro lado, dado un tamaño fijo de lista, se debe analizar como varía el tiempo de ejecución dependiendo del tamaño umbral. Se debe observar cómo a partir de un cierto umbral, el tiempo de ejecución aumenta. El punto mínimo de la curva determina el umbral más adecuado para un determinado tamaño de lista. Se muestra un ejemplo probando distintos umbrales para una lista de tamaño 10000.



Se deben también analizar las curvas de complejidad con algunos umbrales para distintos tamaños de listas. Se muestra un ejemplo de Quicksort con cuatro umbrales distintos:



Ejercicio 3.

Utilizando los datos de entrada de los ficheros Ejercicio3DatosEntradaBinario.txt y Ejercicio3DatosEntradaNario.txt, los resultados de los diferentes tests deben ser:

Árboles binarios

Arbol: A(B,C) Caracter: D [[AB, AC]]
Arbol: A(B,C) Caracter: C [[AB]]
Arbol: A(B,C) Caracter: A [[]]
Arbol: A(B(C,D),E(F,_)) Caracter: H [[ABC, ABD, AEF]]
Arbol: A(B(C,D),E(F,_)) Caracter: D [[ABC, AEF]]
Arbol: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: H [[ABC, ABDE, ABDFG, AIJ, AIK]]
Arbol: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: C [[ABDE, ABDFG, ABDFH, AIJ, AIK]]

Árboles n-arios

Arbol: A(B,C,D) Caracter: A [[]]
Arbol: A(B,C,D) Caracter: C [[AB, AD]]
Arbol: A(B,C,D) Caracter: D [[AB, AC]]
Arbol: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: F [[ABC, ABD, ABE, AJK, AJL]]
Arbol: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: K [[ABC, ABD, ABE, AFG, AFH, AFI, AJL]]
Arbol: A(B(C,D(E,F(G,H,I),J),K)) Caracter: D [[ABC, ABK]]
Arbol: A(B(C,D(E,F(G,H,I),J),K)) Caracter: I [[ABC, ABDE, ABDFG, ABDFH, ABDJ, ABK]]

Ejercicio 4.

Utilizando los datos de entrada de los ficheros Ejercicio4DatosEntradaBinario.txt y Ejercicio4DatosEntradaNario.txt, los resultados de los diferentes tests deben ser:

Árboles binarios

pepe(pepa,pepe): true
pepe(pepa,pep): false
ada(eda(ola,ale),eda(ele,ale)): true
ada(eda(ola,ale),eda(ele,al)): false
cafe(taza(bote,bolsa),perro(gato,leon)): true
cafe(taza(bote,bolsa),perro(gato,_)): false
cafe(taza(bote,bolsa),perro(gato,tortuga)): false

Árboles n-arios

pepe(pepa,pepe,pepo): true
pepe(pepa,pepe,pep): false
ada(eda(ola,ale,elo),eda(ele,ale,alo)): true
ada(eda(ola,ale,elo),eda(ele,ale,al)): false
cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre)): true
cafe(taza(bote,bolsa,vaso),perro(gato,leon)): false
cafe(taza(bote,bolsa,vaso),perro(gato,tortuga)): false
