УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Системы искусственного интеллекта»

**Лабораторная работа №1**

Студент

*Митрофанов Е. Ю.*

*P33101*

Преподаватель

*Бессмертный И. А.*

Санкт-Петербург, 2021 г.

Описание задания

Изучение работы Prolog программы на примере решения задачи сортировки методами вставки и методом «быстрой сортировки».

Входные данные: 2, 5, 0, 7, 2, 0, 0, 1

Требуется:

1) запустить каждую из программ на выполнение в режиме

трассировки

2) записать структурированную трассу программы, отражающую

рекурсивную вложенность предикатов

3) описать логику предикатов в каждой из программ

4) выбрать критерии и сравнить два метода сортировки

Сортировка вставкой

insrtsort([],[]). *%Выход из рекурсии*

insrtsort([Head|Tail], ListSorted) :- insrtsort(Tail,TailSorted), insrt(Head,TailSorted,ListSorted). *%Сортируем все кроме первого элемента, вставляем элемент на нужную позицию в отсортированном списке*

insrt(X, [Y | ListSorted], [Y | ListSorted1]) :- X > Y, !, insrt(X, ListSorted, ListSorted1). *%Ищем место для элемента, после первого подходящего места, завершаем рекурсию*

insrt(X,ListSorted, [X | ListSorted]). *%Вставляем элемент в начало*

**Call:***insrtsort*([2, 5, 0, 7, 2, 0, 0, 1], \_5104)

**Call:***insrtsort*([5, 0, 7, 2, 0, 0, 1], \_5490)

**Call:***insrtsort*([0, 7, 2, 0, 0, 1], \_5492)

**Call:***insrtsort*([7, 2, 0, 0, 1], \_5494)

**Call:***insrtsort*([2, 0, 0, 1], \_5496)

**Call:***insrtsort*([0, 0, 1], \_5498)

**Call:***insrtsort*([0, 1], \_5500)

**Call:***insrtsort*([1], \_5502)

**Call:***insrtsort*([], \_5504)

**Exit:***insrtsort*([], [])

**Call:***insrt*(1, [], \_5502)

**Exit:***insrt*(1, [], [1])

**Exit:***insrtsort*([1], [1])

**Call:***insrt*(0, [1], \_5500)

**Call:**0>1

**Fail:**0>1

**Redo:***insrt*(0, [1], \_5500)

**Exit:***insrt*(0, [1], [0, 1])

**Exit:***insrtsort*([0, 1], [0, 1])

**Call:***insrt*(0, [0, 1], \_5498)

**Call:**0>0

**Fail:**0>0

**Redo:***insrt*(0, [0, 1], \_5498)

**Exit:***insrt*(0, [0, 1], [0, 0, 1])

**Exit:***insrtsort*([0, 0, 1], [0, 0, 1])

**Call:***insrt*(2, [0, 0, 1], \_5496)

**Call:**2>0

**Exit:**2>0

**Call:***insrt*(2, [0, 1], \_5528)

**Call:**2>0

**Exit:**2>0

**Call:***insrt*(2, [1], \_5534)

**Call:**2>1

**Exit:**2>1

**Call:***insrt*(2, [], \_5540)

**Exit:***insrt*(2, [], [2])

**Exit:***insrt*(2, [1], [1, 2])

**Exit:***insrt*(2, [0, 1], [0, 1, 2])

**Exit:***insrt*(2, [0, 0, 1], [0, 0, 1, 2])

**Exit:***insrtsort*([2, 0, 0, 1], [0, 0, 1, 2])

**Call:***insrt*(7, [0, 0, 1, 2], \_5494)

**Call:**7>0

**Exit:**7>0

**Call:***insrt*(7, [0, 1, 2], \_5552)

**Call:**7>0

**Exit:**7>0

**Call:***insrt*(7, [1, 2], \_5558)

**Call:**7>1

**Exit:**7>1

**Call:***insrt*(7, [2], \_5564)

**Call:**7>2

**Exit:**7>2

**Call:***insrt*(7, [], \_5570)

**Exit:***insrt*(7, [], [7])

**Exit:***insrt*(7, [2], [2, 7])

**Exit:***insrt*(7, [1, 2], [1, 2, 7])

**Exit:***insrt*(7, [0, 1, 2], [0, 1, 2, 7])

**Exit:***insrt*(7, [0, 0, 1, 2], [0, 0, 1, 2, 7])

**Exit:***insrtsort*([7, 2, 0, 0, 1], [0, 0, 1, 2, 7])

**Call:***insrt*(0, [0, 0, 1, 2, 7], \_5492)

**Call:**0>0

**Fail:**0>0

**Redo:***insrt*(0, [0, 0, 1, 2, 7], \_5492)

**Exit:***insrt*(0, [0, 0, 1, 2, 7], [0, 0, 0, 1, 2, 7])

**Exit:***insrtsort*([0, 7, 2, 0, 0, 1], [0, 0, 0, 1, 2, 7])

**Call:***insrt*(5, [0, 0, 0, 1, 2, 7], \_5490)

**Call:**5>0

**Exit:**5>0

**Call:***insrt*(5, [0, 0, 1, 2, 7], \_5588)

**Call:**5>0

**Exit:**5>0

**Call:***insrt*(5, [0, 1, 2, 7], \_5594)

**Call:**5>0

**Exit:**5>0

**Call:***insrt*(5, [1, 2, 7], \_5600)

**Call:**5>1

**Exit:**5>1

**Call:***insrt*(5, [2, 7], \_5606)

**Call:**5>2

**Exit:**5>2

**Call:***insrt*(5, [7], \_5612)

**Call:**5>7

**Fail:**5>7

**Redo:***insrt*(5, [7], \_5612)

**Exit:***insrt*(5, [7], [5, 7])

**Exit:***insrt*(5, [2, 7], [2, 5, 7])

**Exit:***insrt*(5, [1, 2, 7], [1, 2, 5, 7])

**Exit:***insrt*(5, [0, 1, 2, 7], [0, 1, 2, 5, 7])

**Exit:***insrt*(5, [0, 0, 1, 2, 7], [0, 0, 1, 2, 5, 7])

**Exit:***insrt*(5, [0, 0, 0, 1, 2, 7], [0, 0, 0, 1, 2, 5, 7])

**Exit:***insrtsort*([5, 0, 7, 2, 0, 0, 1], [0, 0, 0, 1, 2, 5, 7])

**Call:***insrt*(2, [0, 0, 0, 1, 2, 5, 7], \_5104)

**Call:**2>0

**Exit:**2>0

**Call:***insrt*(2, [0, 0, 1, 2, 5, 7], \_5624)

**Call:**2>0

**Exit:**2>0

**Call:***insrt*(2, [0, 1, 2, 5, 7], \_5630)

**Call:**2>0

**Exit:**2>0

**Call:***insrt*(2, [1, 2, 5, 7], \_5636)

**Call:**2>1

**Exit:**2>1

**Call:***insrt*(2, [2, 5, 7], \_5642)

**Call:**2>2

**Fail:**2>2

**Redo:***insrt*(2, [2, 5, 7], \_5642)

**Exit:***insrt*(2, [2, 5, 7], [2, 2, 5, 7])

**Exit:***insrt*(2, [1, 2, 5, 7], [1, 2, 2, 5, 7])

**Exit:***insrt*(2, [0, 1, 2, 5, 7], [0, 1, 2, 2, 5, 7])

**Exit:***insrt*(2, [0, 0, 1, 2, 5, 7], [0, 0, 1, 2, 2, 5, 7])

**Exit:***insrt*(2, [0, 0, 0, 1, 2, 5, 7], [0, 0, 0, 1, 2, 2, 5, 7])

**Exit:***insrtsort*([2, 5, 0, 7, 2, 0, 0, 1], [0, 0, 0, 1, 2, 2, 5, 7])

Быстрая сортировка

fastsort([],[]). *%Выход из рекурсии*

fastsort([Head | Tail], ListSorted) :- split(Head, Tail, TailLess, TailGreater),

 fastsort(TailLess,TailLessSorted), fastsort(TailGreater,TailGreaterSorted),

 append(TailLessSorted, [Head | TailGreaterSorted], ListSorted).

%Разделение на два подмассива, сортировка каждого, соединение левого + голова + правого

split(\_, [], [], []). *%Выход из рекурсии*

split(X, [H | T], [H | TL], TG) :- H < X, !, split(X, T, TL, TG).

split(X, [H | T], TL, [H | TG]) :- split(X, T, TL, TG).

*% Отделение первого элемента. Если он меньше опорного, в левую часть. Остальное включая опорный в правую.*

**Call:***fastsort*([2, 5, 0, 7, 2, 0, 0, 1], \_6326)

**Call:***split*(2, [5, 0, 7, 2, 0, 0, 1], \_6712, \_6714)

**Call:**5<2

**Fail:**5<2

**Redo:***split*(2, [5, 0, 7, 2, 0, 0, 1], \_6712, \_6714)

**Call:***split*(2, [0, 7, 2, 0, 0, 1], \_6712, \_6720)

**Call:**0<2

**Exit:**0<2

**Call:***split*(2, [7, 2, 0, 0, 1], \_6726, \_6720)

**Call:**7<2

**Fail:**7<2

**Redo:***split*(2, [7, 2, 0, 0, 1], \_6726, \_6720)

**Call:***split*(2, [2, 0, 0, 1], \_6726, \_6732)

**Call:**2<2

**Fail:**2<2

**Redo:***split*(2, [2, 0, 0, 1], \_6726, \_6732)

**Call:***split*(2, [0, 0, 1], \_6726, \_6738)

**Call:**0<2

**Exit:**0<2

**Call:***split*(2, [0, 1], \_6744, \_6738)

**Call:**0<2

**Exit:**0<2

**Call:***split*(2, [1], \_6750, \_6738)

**Call:**1<2

**Exit:**1<2

**Call:***split*(2, [], \_6756, \_6738)

**Exit:***split*(2, [], [], [])

**Exit:***split*(2, [1], [1], [])

**Exit:***split*(2, [0, 1], [0, 1], [])

**Exit:***split*(2, [0, 0, 1], [0, 0, 1], [])

**Exit:***split*(2, [2, 0, 0, 1], [0, 0, 1], [2])

**Exit:***split*(2, [7, 2, 0, 0, 1], [0, 0, 1], [7, 2])

**Exit:***split*(2, [0, 7, 2, 0, 0, 1], [0, 0, 0, 1], [7, 2])

**Exit:***split*(2, [5, 0, 7, 2, 0, 0, 1], [0, 0, 0, 1], [5, 7, 2])

**Call:***fastsort*([0, 0, 0, 1], \_6758)

**Call:***split*(0, [0, 0, 1], \_6760, \_6762)

**Call:**0<0

**Fail:**0<0

**Redo:***split*(0, [0, 0, 1], \_6760, \_6762)

**Call:***split*(0, [0, 1], \_6760, \_6768)

**Call:**0<0

**Fail:**0<0

**Redo:***split*(0, [0, 1], \_6760, \_6768)

**Call:***split*(0, [1], \_6760, \_6774)

**Call:**1<0

**Fail:**1<0

**Redo:***split*(0, [1], \_6760, \_6774)

**Call:***split*(0, [], \_6760, \_6780)

**Exit:***split*(0, [], [], [])

**Exit:***split*(0, [1], [], [1])

**Exit:***split*(0, [0, 1], [], [0, 1])

**Exit:***split*(0, [0, 0, 1], [], [0, 0, 1])

**Call:***fastsort*([], \_6782)

**Exit:***fastsort*([], [])

**Call:***fastsort*([0, 0, 1], \_6784)

**Call:***split*(0, [0, 1], \_6786, \_6788)

**Call:**0<0

**Fail:**0<0

**Redo:***split*(0, [0, 1], \_6786, \_6788)

**Call:***split*(0, [1], \_6786, \_6794)

**Call:**1<0

**Fail:**1<0

**Redo:***split*(0, [1], \_6786, \_6794)

**Call:***split*(0, [], \_6786, \_6800)

**Exit:***split*(0, [], [], [])

**Exit:***split*(0, [1], [], [1])

**Exit:***split*(0, [0, 1], [], [0, 1])

**Call:***fastsort*([], \_6802)

**Exit:***fastsort*([], [])

**Call:***fastsort*([0, 1], \_6804)

**Call:***split*(0, [1], \_6806, \_6808)

**Call:**1<0

**Fail:**1<0

**Redo:***split*(0, [1], \_6806, \_6808)

**Call:***split*(0, [], \_6806, \_6814)

**Exit:***split*(0, [], [], [])

**Exit:***split*(0, [1], [], [1])

**Call:***fastsort*([], \_6816)

**Exit:***fastsort*([], [])

**Call:***fastsort*([1], \_6818)

**Call:***split*(1, [], \_6820, \_6822)

**Exit:***split*(1, [], [], [])

**Call:***fastsort*([], \_6824)

**Exit:***fastsort*([], [])

**Call:***fastsort*([], \_6826)

**Exit:***fastsort*([], [])

**Call:**lists:*append*([], [1], \_6818)

**Exit:**lists:*append*([], [1], [1])

**Exit:***fastsort*([1], [1])

**Call:**lists:*append*([], [0, 1], \_6804)

**Exit:**lists:*append*([], [0, 1], [0, 1])

**Exit:***fastsort*([0, 1], [0, 1])

**Call:**lists:*append*([], [0, 0, 1], \_6784)

**Exit:**lists:*append*([], [0, 0, 1], [0, 0, 1])

**Exit:***fastsort*([0, 0, 1], [0, 0, 1])

**Call:**lists:*append*([], [0, 0, 0, 1], \_6758)

**Exit:**lists:*append*([], [0, 0, 0, 1], [0, 0, 0, 1])

**Exit:***fastsort*([0, 0, 0, 1], [0, 0, 0, 1])

**Call:***fastsort*([5, 7, 2], \_6852)

**Call:***split*(5, [7, 2], \_6854, \_6856)

**Call:**7<5

**Fail:**7<5

**Redo:***split*(5, [7, 2], \_6854, \_6856)

**Call:***split*(5, [2], \_6854, \_6862)

**Call:**2<5

**Exit:**2<5

**Call:***split*(5, [], \_6868, \_6862)

**Exit:***split*(5, [], [], [])

**Exit:***split*(5, [2], [2], [])

**Exit:***split*(5, [7, 2], [2], [7])

**Call:***fastsort*([2], \_6870)

**Call:***split*(2, [], \_6872, \_6874)

**Exit:***split*(2, [], [], [])

**Call:***fastsort*([], \_6876)

**Exit:***fastsort*([], [])

**Call:***fastsort*([], \_6878)

**Exit:***fastsort*([], [])

**Call:**lists:*append*([], [2], \_6870)

**Exit:**lists:*append*([], [2], [2])

**Exit:***fastsort*([2], [2])

**Call:***fastsort*([7], \_6886)

**Call:***split*(7, [], \_6888, \_6890)

**Exit:***split*(7, [], [], [])

**Call:***fastsort*([], \_6892)

**Exit:***fastsort*([], [])

**Call:***fastsort*([], \_6894)

**Exit:***fastsort*([], [])

**Call:**lists:*append*([], [7], \_6886)

**Exit:**lists:*append*([], [7], [7])

**Exit:***fastsort*([7], [7])

**Call:**lists:*append*([2], [5, 7], \_6852)

**Exit:**lists:*append*([2], [5, 7], [2, 5, 7])

**Exit:***fastsort*([5, 7, 2], [2, 5, 7])

**Call:**lists:*append*([0, 0, 0, 1], [2, 2, 5, 7], \_6326)

**Exit:**lists:*append*([0, 0, 0, 1], [2, 2, 5, 7], [0, 0, 0, 1, 2, 2, 5, 7])

**Exit:***fastsort*([2, 5, 0, 7, 2, 0, 0, 1], [0, 0, 0, 1, 2, 2, 5, 7])

Логика сортировки вставкой

Сортировка вставкой предполагает последовательный перебор списка. Каждую итерацию берется голова списка и для нее выбирается место вставки в отсортированной части списка, с условием что предыдущий элемент меньше, а следующий больше. Голова отсортированной части отделяется, после чего вставляется в начало отсортированной части, если она больше выбранного элемента.

Логика быстрой сортировки

Опорным элементом выбирается первый элемент списка. Элементы, меньшие выборного распределяются в первый подмассив, элементы равные и больше – во второй. Каждый из подмассивов подобным образом сортируется рекурсивно.

Сравнение алгоритмов

Среднее время работы быстрой сортировки – nlogn. Сортировка вставкой – n^2. Но худшее время для быстрой сортировки может быть так жe N^2. На малый выборке и при частично отсортированном списке сортировка вставкой может работать быстрее.

Глубина рекурсии у двух методов одинаковая и зависит только от количества элементов.

Выводы

При выполнении лабораторной работы я изучил работу языка Prolog на примере двух алгоритмов сортировки.