

Санкт-Петербургский Национальный Исследовательский Университет  
Информационных Технологий, Механики и Оптики

Факультет программной инженерии и компьютерной техники

Домашнее задание №1  
по дисциплине  
«Системы искусственного интеллекта»

Выполнил:  
Студент группы Р33101,  
Патулин Владимир

Преподаватель: Бессмертный И.А.

Санкт-Петербург  
2021г.

## 1. Исходный текст программы с комментариями:

- Сортировка методом вставки:

```
% выход из рекурсии
insrtsort([], []).
% Рекурсивно отделяем первый элемент массива(голову), потом
% вызываем функцию вставки в отсортированный массив.
insrtsort([Head|Tail], ListSorted) :- insrtsort(Tail, TailSorted),
insrt(Head, TailSorted, ListSorted).
% Проверяем больше ли вставляемое число первого элемента
%отсортированного списка, если так, то рекурсивно вызываем %функцию,
%пока не получим fail в условии и не откатимся на
%другой предикат, в противном случае откатываемся на другой
%предикат.
insrt(X, [Y | ListSorted], [Y | ListSorted1]) :- X > Y, !,
insrt(X, ListSorted, ListSorted1).
%Вставка в начало отсортированного массива
insrt(X, ListSorted, [X | ListSorted]).
```

- Быстрая сортировка:

```
% выход из рекурсии
fastsort([], []).
% Дробление массива на массивы больше и меньше головы
%и дальнейшее заикливание для сортировки полученных
%массивов. В итоге соединяем массив: меньше головы+ голова
%+ массив больше головы.
fastsort([Head | Tail], ListSorted) :-
split(Head, Tail, TailLess, TailGreater),
fastsort(TailLess, TailLessSorted),
fastsort(TailGreater, TailGreaterSorted),
append(TailLessSorted, [Head | TailGreaterSorted], ListSorted).
% выход из рекурсии
split(_, [], [], []).
% Предикат, который в итоге вернет массив меньше X
split(X, [H | T], [H | TL], TG) :- H < X, !, split(X, T, TL, TG).
% Предикат, который в итоге вернет массив больше X
split(X, [H | T], TL, [H | TG]) :- split(X, T, TL, TG).
```

## 2. Структурированная трасса хода выполнения каждой из двух программы:

- Исходные данные: [1,5,0,3,2,0,0,2]

- Сортировка методом вставки:

Call: *insrtsort*([1, 5, 0, 3, 2, 0, 0, 2], \_4880)

Call: *insrtsort*([5, 0, 3, 2, 0, 0, 2], \_5266)

Call: *insrtsort*([0, 3, 2, 0, 0, 2], \_5268)

Call: *insrtsort*([3, 2, 0, 0, 2], \_5270)

Call: *insrtsort*([2, 0, 0, 2], \_5272)

Call: *insrtsort*([0, 0, 2], \_5274)

Call: *insrtsort*([0, 2], \_5276)

Call: *insrtsort*([2], \_5278)

Call: *insrtsort*([], \_5280)

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

Call: *insrt*(2, [], \_5278)

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

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

Call: *insrt*(0, [2], \_5276)

Call: 0>2

Fail: 0>2

Redo: *insrt*(0, [2], \_5276)

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

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

Call: *insrt*(0, [0, 2], \_5274)

Call: 0>0

Fail: 0>0

Redo: *insrt*(0, [0, 2], \_5274)

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

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

Call: *insrt*(2, [0, 0, 2], \_5272)

Call: 2>0

Exit: 2>0

Call: *insrt*(2, [0, 2], \_5304)

Call: 2>0

Exit: 2>0

Call: *insrt*(2, [2], \_5310)

Call: 2>2

Fail: 2>2

Redo: *insrt*(2, [2], \_5310)

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

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

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

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

Call: *insrt*(3, [0, 0, 2, 2], \_5270)

Call: 3>0

Exit: 3>0

Call: *insrt*(3, [0, 2, 2], \_5322)

Call: 3>0

Exit: 3>0

Call: *insrt*(3, [2, 2], \_5328)

Call: 3>2

Exit: 3>2

Call: *insrt*(3, [2], \_5334)

Call: 3>2

Exit: 3>2

Call: *insrt*(3, [], \_5340)

Exit: *insrt*(3, [], [3])

Exit: *insrt*(3, [2], [2, 3])

Exit: *insrt*(3, [2, 2], [2, 2, 3])

Exit: *insrt*(3, [0, 2, 2], [0, 2, 2, 3])

Exit: *insrt*(3, [0, 0, 2, 2], [0, 0, 2, 2, 3])

Exit: *insrtsort*([3, 2, 0, 0, 2], [0, 0, 2, 2, 3])

**Call:** *insrt*(0, [0, 0, 2, 2, 3], \_5268)  
**Call:** 0>0  
**Fail:** 0>0  
**Redo:** *insrt*(0, [0, 0, 2, 2, 3], \_5268)  
**Exit:** *insrt*(0, [0, 0, 2, 2, 3], [0, 0, 0, 2, 2, 3])  
**Exit:** *insrtsort*([0, 3, 2, 0, 0, 2], [0, 0, 0, 2, 2, 3])  
**Call:** *insrt*(5, [0, 0, 0, 2, 2, 3], \_5266)  
**Call:** 5>0  
**Exit:** 5>0  
**Call:** *insrt*(5, [0, 0, 2, 2, 3], \_5358)  
**Call:** 5>0  
**Exit:** 5>0  
**Call:** *insrt*(5, [0, 2, 2, 3], \_5364)  
**Call:** 5>0  
**Exit:** 5>0  
**Call:** *insrt*(5, [2, 2, 3], \_5370)  
**Call:** 5>2  
**Exit:** 5>2  
**Call:** *insrt*(5, [2, 3], \_5376)  
**Call:** 5>2  
**Exit:** 5>2  
**Call:** *insrt*(5, [3], \_5382)  
**Call:** 5>3  
**Exit:** 5>3  
**Call:** *insrt*(5, [], \_5388)  
**Exit:** *insrt*(5, [], [5])  
**Exit:** *insrt*(5, [3], [3, 5])  
**Exit:** *insrt*(5, [2, 3], [2, 3, 5])  
**Exit:** *insrt*(5, [2, 2, 3], [2, 2, 3, 5])  
**Exit:** *insrt*(5, [0, 2, 2, 3], [0, 2, 2, 3, 5])  
**Exit:** *insrt*(5, [0, 0, 2, 2, 3], [0, 0, 2, 2, 3, 5])  
**Exit:** *insrt*(5, [0, 0, 0, 2, 2, 3], [0, 0, 0, 2, 2, 3, 5])  
**Exit:** *insrtsort*([5, 0, 3, 2, 0, 0, 2], [0, 0, 0, 2, 2, 3, 5])  
**Call:** *insrt*(1, [0, 0, 0, 2, 2, 3, 5], \_4880)  
**Call:** 1>0  
**Exit:** 1>0  
**Call:** *insrt*(1, [0, 0, 2, 2, 3, 5], \_5400)  
**Call:** 1>0  
**Exit:** 1>0  
**Call:** *insrt*(1, [0, 2, 2, 3, 5], \_5406)  
**Call:** 1>0  
**Exit:** 1>0  
**Call:** *insrt*(1, [2, 2, 3, 5], \_5412)  
**Call:** 1>2  
**Fail:** 1>2  
**Redo:** *insrt*(1, [2, 2, 3, 5], \_5412)  
**Exit:** *insrt*(1, [2, 2, 3, 5], [1, 2, 2, 3, 5])  
**Exit:** *insrt*(1, [0, 2, 2, 3, 5], [0, 1, 2, 2, 3, 5])  
**Exit:** *insrt*(1, [0, 0, 2, 2, 3, 5], [0, 0, 1, 2, 2, 3, 5])  
**Exit:** *insrt*(1, [0, 0, 0, 2, 2, 3, 5], [0, 0, 0, 1, 2, 2, 3, 5])  
**Exit:** *insrtsort*([1, 5, 0, 3, 2, 0, 0, 2], [0, 0, 0, 1, 2, 2, 3, 5])  
**X** = [0, 0, 0, 1, 2, 2, 3, 5]

- Быстрая сортировка:

**Call:** *fastsort*([1, 5, 0, 3, 2, 0, 0, 2], \_6310)  
**Call:** *split*(1, [5, 0, 3, 2, 0, 0, 2], \_6696, \_6698)  
**Call:** 5<1  
**Fail:** 5<1  
**Redo:** *split*(1, [5, 0, 3, 2, 0, 0, 2], \_6696, \_6698)  
**Call:** *split*(1, [0, 3, 2, 0, 0, 2], \_6696, \_6704)

**Call:**0<1  
**Exit:**0<1  
**Call:***split*(1, [3, 2, 0, 0, 2], \_6710, \_6704)  
**Call:**3<1  
**Fail:**3<1  
**Redo:***split*(1, [3, 2, 0, 0, 2], \_6710, \_6704)  
**Call:***split*(1, [2, 0, 0, 2], \_6710, \_6716)  
**Call:**2<1  
**Fail:**2<1  
**Redo:***split*(1, [2, 0, 0, 2], \_6710, \_6716)  
**Call:***split*(1, [0, 0, 2], \_6710, \_6722)  
**Call:**0<1  
**Exit:**0<1  
**Call:***split*(1, [0, 2], \_6728, \_6722)  
**Call:**0<1  
**Exit:**0<1  
**Call:***split*(1, [2], \_6734, \_6722)  
**Call:**2<1  
**Fail:**2<1  
**Redo:***split*(1, [2], \_6734, \_6722)  
**Call:***split*(1, [], \_6734, \_6740)  
**Exit:***split*(1, [], [], [])  
**Exit:***split*(1, [2], [], [2])  
**Exit:***split*(1, [0, 2], [0], [2])  
**Exit:***split*(1, [0, 0, 2], [0, 0], [2])  
**Exit:***split*(1, [2, 0, 0, 2], [0, 0], [2, 2])  
**Exit:***split*(1, [3, 2, 0, 0, 2], [0, 0], [3, 2, 2])  
**Exit:***split*(1, [0, 3, 2, 0, 0, 2], [0, 0, 0], [3, 2, 2])  
**Exit:***split*(1, [5, 0, 3, 2, 0, 0, 2], [0, 0, 0], [5, 3, 2, 2])  
**Call:***fastsort*([0, 0, 0], \_6742)  
**Call:***split*(0, [0, 0], \_6744, \_6746)  
**Call:**0<0  
**Fail:**0<0  
**Redo:***split*(0, [0, 0], \_6744, \_6746)  
**Call:***split*(0, [0], \_6744, \_6752)  
**Call:**0<0  
**Fail:**0<0  
**Redo:***split*(0, [0], \_6744, \_6752)  
**Call:***split*(0, [], \_6744, \_6758)  
**Exit:***split*(0, [], [], [])  
**Exit:***split*(0, [0], [], [0])  
**Exit:***split*(0, [0, 0], [], [0, 0])  
**Call:***fastsort*([], \_6760)  
**Exit:***fastsort*([], [])  
**Call:***fastsort*([0, 0], \_6762)  
**Call:***split*(0, [0], \_6764, \_6766)  
**Call:**0<0  
**Fail:**0<0  
**Redo:***split*(0, [0], \_6764, \_6766)  
**Call:***split*(0, [], \_6764, \_6772)  
**Exit:***split*(0, [], [], [])  
**Exit:***split*(0, [0], [], [0])  
**Call:***fastsort*([], \_6774)  
**Exit:***fastsort*([], [])  
**Call:***fastsort*([0], \_6776)  
**Call:***split*(0, [], \_6778, \_6780)  
**Exit:***split*(0, [], [], [])  
**Call:***fastsort*([], \_6782)  
**Exit:***fastsort*([], [])  
**Call:***fastsort*([], \_6784)

**Exit:** *fastsort*([], [])  
**Call:** *lists.append*([], [0], \_6776)  
**Exit:** *lists.append*([], [0], [0])  
**Exit:** *fastsort*([0], [0])  
**Call:** *lists.append*([], [0, 0], \_6762)  
**Exit:** *lists.append*([], [0, 0], [0, 0])  
**Exit:** *fastsort*([0, 0], [0, 0])  
**Call:** *lists.append*([], [0, 0, 0], \_6742)  
**Exit:** *lists.append*([], [0, 0, 0], [0, 0, 0])  
**Exit:** *fastsort*([0, 0, 0], [0, 0, 0])  
**Call:** *fastsort*([5, 3, 2, 2], \_6804)  
**Call:** *split*(5, [3, 2, 2], \_6806, \_6808)  
**Call:** 3<5  
**Exit:** 3<5  
**Call:** *split*(5, [2, 2], \_6814, \_6808)  
**Call:** 2<5  
**Exit:** 2<5  
**Call:** *split*(5, [2], \_6820, \_6808)  
**Call:** 2<5  
**Exit:** 2<5  
**Call:** *split*(5, [], \_6826, \_6808)  
**Exit:** *split*(5, [], [], [])  
**Exit:** *split*(5, [2], [2], [])  
**Exit:** *split*(5, [2, 2], [2, 2], [])  
**Exit:** *split*(5, [3, 2, 2], [3, 2, 2], [])  
**Call:** *fastsort*([3, 2, 2], \_6828)  
**Call:** *split*(3, [2, 2], \_6830, \_6832)  
**Call:** 2<3  
**Exit:** 2<3  
**Call:** *split*(3, [2], \_6838, \_6832)  
**Call:** 2<3  
**Exit:** 2<3  
**Call:** *split*(3, [], \_6844, \_6832)  
**Exit:** *split*(3, [], [], [])  
**Exit:** *split*(3, [2], [2], [])  
**Exit:** *split*(3, [2, 2], [2, 2], [])  
**Call:** *fastsort*([2, 2], \_6846)  
**Call:** *split*(2, [2], \_6848, \_6850)  
**Call:** 2<2  
**Fail:** 2<2  
**Redo:** *split*(2, [2], \_6848, \_6850)  
**Call:** *split*(2, [], \_6848, \_6856)  
**Exit:** *split*(2, [], [], [])  
**Exit:** *split*(2, [2], [], [2])  
**Call:** *fastsort*([], \_6858)  
**Exit:** *fastsort*([], [])  
**Call:** *fastsort*([2], \_6860)  
**Call:** *split*(2, [], \_6862, \_6864)  
**Exit:** *split*(2, [], [], [])  
**Call:** *fastsort*([], \_6866)  
**Exit:** *fastsort*([], [])  
**Call:** *fastsort*([], \_6868)  
**Exit:** *fastsort*([], [])  
**Call:** *lists.append*([], [2], \_6860)  
**Exit:** *lists.append*([], [2], [2])  
**Exit:** *fastsort*([2], [2])  
**Call:** *lists.append*([], [2, 2], \_6846)  
**Exit:** *lists.append*([], [2, 2], [2, 2])  
**Exit:** *fastsort*([2, 2], [2, 2])  
**Call:** *fastsort*([], \_6882)

```

Exit:fastsort([], [])
Call:lists:append([2, 2], [3], _6828)
Exit:lists:append([2, 2], [3], [2, 2, 3])
Exit:fastsort([3, 2, 2], [2, 2, 3])
Call:fastsort([], _6902)
Exit:fastsort([], [])
Call:lists:append([2, 2, 3], [5], _6804)
Exit:lists:append([2, 2, 3], [5], [2, 2, 3, 5])
Exit:fastsort([5, 3, 2, 2], [2, 2, 3, 5])
Call:lists:append([0, 0, 0], [1, 2, 2, 3, 5], _6310)
Exit:lists:append([0, 0, 0], [1, 2, 2, 3, 5], [0, 0, 0, 1, 2, 2, 3, 5])
Exit:fastsort([1, 5, 0, 3, 2, 0, 0, 2], [0, 0, 0, 1, 2, 2, 3, 5])
X = [0, 0, 0, 1, 2, 2, 3, 5]

```

### 3. Описание логики каждого из методов сортировки:

- Логика метода вставки:

Запускаем программу. «insrtsort([1,5,0,3,2,0,0,2],X)», попадаем на соответствующий предикат, попадая в рекурсию, и каждой итерацией снимаем голову нашего массива, пока не дойдем до вида «insrtsort([],\_)», в этот момент мы выходим из рекурсии. Вызывается новый предикат «insrt(X, [Y | ListSorted], [Y | ListSorted1])», который является неоднозначным, проверяем условие «X > Y», если оно не выполняется, то попадаем на предикат «insrt(X,ListSorted, [X | ListSorted])», который отвечает за вставку элемента в начало отсортированного массива(делает его головой отсортированного массива), в противном случае уходим на рекурсию данного предиката, пока условие не получим fail.

- Логика быстро сортировки:

Запускаем программу. «fastsort([1,5,0,3,2,0,0,2],X)», Попадаем на соответствующий предикат, который перекидывает нас на «split(X, [H | T], [H | TL], TG)», где происходит проверка головы оставшегося массива относительно изначальной головы, если изначальная голова меньше, то откат и идем в «split(X, [H | T], TL, [H | TG])», где будет храниться числа меньше изначальной головы, в противном случае рекурсия и получаем массив из элементов, которые больше изначальной головы. Выйдя из «split(X, [H | T], [H | TL], TG)» происходит сортировка подмассива с меньшими и большими числами, и в итоге соединяем в последовательности «массив

меньше+ изначальная голова + массив больше» в одну переменную ListSorted.

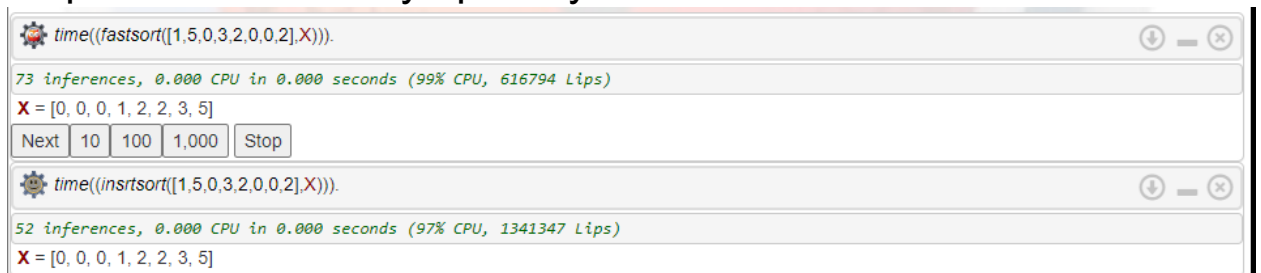
#### 4. Сравнение рассмотренных методов сортировки:

Начать стоит со сложности, которой обладают данные методы, метод вставки обладает сложностью  $O(N^2)$ , а метод быстрой сортировки обладает сложностью  $O(N \cdot \log(N))$   $\Rightarrow$  лучше использовать метод быстрой сортировки.

Глубина погружения у обоих методов одинаковая, потому что зависит от количества элементов.

Метод вставки обладает меньшим количеством правил, что ускоряет время его написания и его просто легче понимать.

Обратимся к самому прологу:



The screenshot shows two code cells in a Jupyter Notebook. The first cell runs `time((fastsort([1,5,0,3,2,0,0,2],X)))` and reports 73 inferences, 0.000 CPU in 0.000 seconds (99% CPU, 616794 Lips). The second cell runs `time((insrtsort([1,5,0,3,2,0,0,2],X)))` and reports 52 inferences, 0.000 CPU in 0.000 seconds (97% CPU, 1341347 Lips). Both cells use the same input array `X = [0, 0, 0, 1, 2, 2, 3, 5]`. The first cell has a 'Next' button and a 'Stop' button.

Method	Inferences	CPU	Time	Percentage	Lips
fastsort	73	0.000	0.000 seconds	99%	616794
insrtsort	52	0.000	0.000 seconds	97%	1341347