UCS1524 – Logic Programming

Operations on Data structures - Sorting

Session Meta Data

Author	**Dr. D. Thenmozhi**
Reviewer	
Version Number	1.2
Release Date	4 October 2020

Session Objectives

- Understanding the operations on data structure namely sorting.
- Learn about different sorting techniques namely bubble sort, insertion sort and quick sort.

Session Outcomes

- At the end of this session, participants will be able to
  - Understand the operations on data structure with different sorting techniques.

Agenda

- Operations on data structure
- Sorting
  - Bubble sort
  - Insertion sort
  - Quick sort

Sorting lists

- A list can be sorted if there is an ordering relation between the items in the list.
- Assume that there is an ordering relation

  gt(X, Y)

  meaning the X is greater than Y.

  – If our items are numbers then the **gt** relation is defined as:

    gt(X, Y) :- X > Y.

  – If the items are atoms then we can define:

    gt(X, Y) :- X @> Y.

  – Remember that this relation also orders compound terms.

Sorting lists

- Let

  sort(List, Sorted)

  denote a relation where **List** is a list of items and **Sorted** is a list of the same items sorted in the ascending order according to the **gt** relation.

Sorting lists

- To sort a list, **List**:
  - Find two adjacent elements, X and Y, in **List** such that **gt(X, Y)** and swap X and Y in **List**, obtaining **List1**; then sort **List1**.
  - If there is no pair of adjacent elements, X and Y, in **List** such that **gt(X, Y),** then **List** is already sorted.

- Bubble sort:

```
bubblesort( List, Sorted) :-
    swap( List, List1), !,                    %swap 2 elements
    bubblesort( List1, Sorted).
bubblesort( Sorted, Sorted).                  %list is sorted
swap([X, Y| Rest], [Y, X| Rest]) :- gt( X, Y).    % swap 1st 2 ele
swap([Z|Rest],[Z|Rest1]) :- swap( Rest, Rest1).  %swap tail
```

Sorting lists

```
| ?- bubblesort([3,5,2,4], L).
    1  1  Call: bubblesort([3,5,2,4],_24) ?
    2  2  Call: swap([3,5,2,4],_93) ?
    3  3  Call: gt(3,5) ?
    4  4  Call: 3>5 ?
    4  4  Fail: 3>5 ?
    3  3  Fail: gt(3,5) ?
    3  3  Call: swap([5,2,4],_80) ?
    4  4  Call: gt(5,2) ?
    5  5  Call: 5>2 ?
    5  5  Exit: 5>2 ?
    4  4  Exit: gt(5,2) ?
    3  3  Exit: swap([5,2,4],[2,5,4]) ?
    2  2  Exit: swap([3,5,2,4],[3,2,5,4]) ?
    6  2  Call: bubblesort([3,2,5,4],_24) ?
    7  3  Call: swap([3,2,5,4],_223) ?
    8  4  Call: gt(3,2) ?
    9  5  Call: 3>2 ?
    9  5  Exit: 3>2 ?
    8  4  Exit: gt(3,2) ?
    7  3  Exit: swap([3,2,5,4],[2,3,5,4]) ?
   10  3  Call: bubblesort([2,3,5,4],_24) ?
   11  4  Call: swap([2,3,5,4],_326) ?
   12  5  Call: gt(2,3) ?
   13  6  Call: 2>3 ?
   13  6  Fail: 2>3 ?
   12  5  Fail: gt(2,3) ?
   12  5  Call: swap([3,5,4],_313) ?
   13  6  Call: gt(3,5) ?
   14  7  Call: 3>5 ?
   14  7  Fail: 3>5 ?
   13  6  Fail: gt(3,5) ?
   13  6  Call: swap([5,4],_339) ?
   14  7  Call: gt(5,4) ?
   15  8  Call: 5>4 ?
   15  8  Exit: 5>4 ?
   14  7  Exit: gt(5,4) ?
   13  6  Exit: swap([5,4],[4,5]) ?
   12  5  Exit: swap([3,5,4],[3,4,5]) ?
   11  4  Exit: swap([2,3,5,4],[2,3,4,5]) ?
   16  4  Call: bubblesort([2,3,4,5],_24) ?
   17  5  Call: swap([2,3,4,5],_483) ?
   18  6  Call: gt(2,3) ?
   19  7  Call: 2>3 ?
   19  7  Fail: 2>3 ?
   18  6  Fail: gt(2,3) ?
   18  6  Call: swap([3,4,5],_470) ?
   19  7  Call: gt(3,4) ?
   20  8  Call: 3>4 ?
   20  8  Fail: 3>4 ?
   19  7  Fail: gt(3,4) ?
   19  7  Call: swap([4,5],_496) ?
   20  8  Call: gt(4,5) ?
   21  9  Call: 4>5 ?
   21  9  Fail: 4>5 ?
   20  8  Fail: gt(4,5) ?
   20  8  Call: swap([5],_522) ?
   21  9  Call: swap([],_548) ?
   21  9  Fail: swap([],_548) ?
   20  8  Fail: swap([5],_522) ?
   19  7  Fail: swap([4,5],_496) ?
   18  6  Fail: swap([3,4,5],_470) ?
   17  5  Fail: swap([2,3,4,5],_471) ?
   16  4  Exit: bubblesort([2,3,4,5],[2,3,4,5]) ?
   10  3  Exit: bubblesort([2,3,5,4],[2,3,4,5]) ?
    6  2  Exit: bubblesort([3,2,5,4],[2,3,4,5]) ?
    1  1  Exit: bubblesort([3,5,2,4],[2,3,4,5]) ?

L = [2,3,4,5]

(94 ms) yes
{trace}
```

Sorting lists

- To sort a non-empty list, **L = [X|T]**:
  - Sort the tail **T** of **L**.
  - Insert the head, **X**, of **L** into the sorted tail at such a position that the resulting list is sorted. The result is the whole sorted list.

- Insertion sort:

```
insertsort( [], []).
insertsort( [X|Tail], Sorted) :-
    insertsort( Tail, SortedTail),
    insert( X, SortedTail, Sorted).
insert(X, [Y| Sorted], [Y| Sorted1]) :-
    gt( X, Y), !, insert( X, Sorted, Sorted1).
insert(X, Sorted, [X|Sorted]).
```

Sorting lists

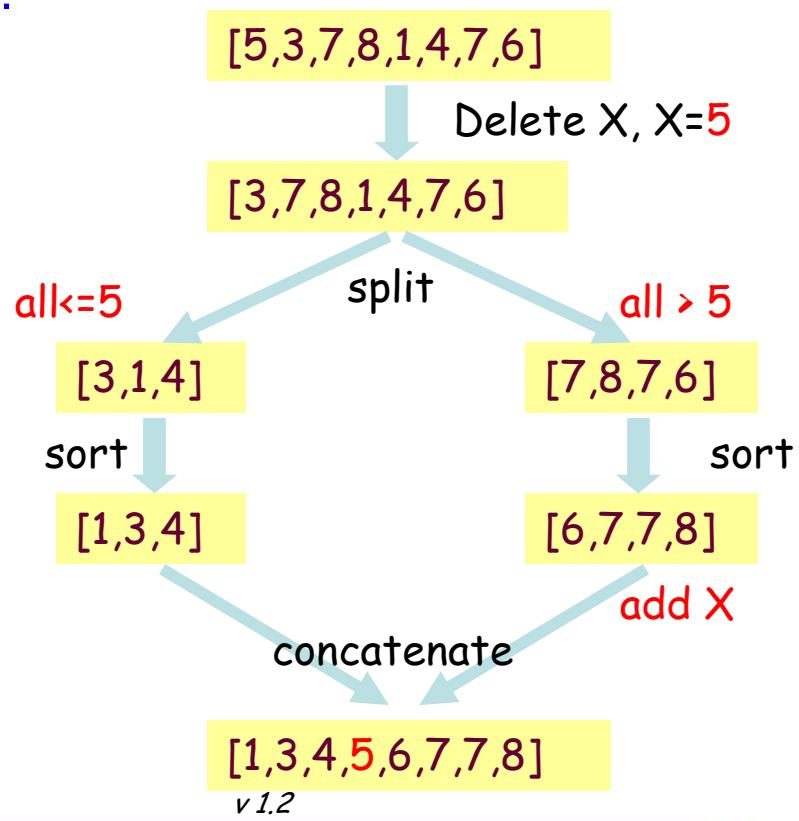
```
| ?- insertsort([3,5,2,4], L).
    1  1  Call: insertsort([3,5,2,4],_24) ?
    2  2  Call: insertsort([5,2,4],_93) ?
    3  3  Call: insertsort([2,4],_117) ?
    4  4  Call: insertsort([4],_141) ?
    5  5  Call: insertsort([],_165) ?
    5  5  Exit: insertsort([],[]) ?
    6  5  Call: insert(4,[],_191) ?
    6  5  Exit: insert(4,[],[4]) ?
    4  4  Exit: insertsort([4],[4]) ?
    7  4  Call: insert(2,[4],_220) ?
    8  5  Call: gt(2,4) ?
    9  6  Call: 2>4 ?
    9  6  Fail: 2>4 ?
    8  5  Fail: gt(2,4) ?
    7  4  Exit: insert(2,[4],[2,4]) ?
    3  3  Exit: insertsort([2,4],[2,4]) ?
    8  3  Call: insert(5,[2,4],_249) ?
    9  4  Call: gt(5,2) ?
   10  5  Call: 5>2 ?
   10  5  Exit: 5>2 ?
    9  4  Exit: gt(5,2) ?
   11  4  Call: insert(5,[4],_236) ?
   12  5  Call: gt(5,4) ?
   13  6  Call: 5>4 ?
   13  6  Exit: 5>4 ?
   12  5  Exit: gt(5,4) ?
   14  5  Call: insert(5,[],_313) ?
   14  5  Exit: insert(5,[],[5]) ?
   11  4  Exit: insert(5,[4],[4,5]) ?
    8  3  Exit: insert(5,[2,4],[2,4,5]) ?
    2  2  Exit: insertsort([5,2,4],[2,4,5]) ?
   15  2  Call: insert(3,[2,4,5],_24) ?
   16  3  Call: gt(3,2) ?
   17  4  Call: 3>2 ?
   17  4  Exit: 3>2 ?
   16  3  Exit: gt(3,2) ?
   18  3  Call: insert(3,[4,5],_421) ?
   19  4  Call: gt(3,4) ?
   20  5  Call: 3>4 ?
   20  5  Fail: 3>4 ?
   19  4  Fail: gt(3,4) ?
   18  3  Exit: insert(3,[4,5],[3,4,5]) ?
   15  2  Exit: insert(3,[2,4,5],[2,3,4,5]) ?
    1  1  Exit: insertsort([3,5,2,4],[2,3,4,5]) ?

L = [2,3,4,5]

(94 ms) yes
{trace}
```

Sorting lists

- The sorting procedures **bubblesort** and **insertsort** are simple, but inefficient. (time complexity is n^2).
- A much better sorting algorithm is **quicksort**.
- For example:



Sorting lists

- To sort a non-empty list, **L**:
  - Delete some element **X** from **L** and split the rest of **L** into two lists, called **Small** and **Big**, as follows:
    - All elements in L that are greater then X belong to **Big,**
    - And all others to **Small**.
  - Sort **Small** obtaining **SortedSmall**.
  - Sort **Big** obtaining **SortedBig**.
  - The whole sorted list is the concatenation of **SortedSmall** and **[X| SortedBig].**

Sorting lists

- Quick sort:

```
% Figure 9.2  Quicksort.
quicksort( [], []).
quicksort( [X|Tail], Sorted)  :-
     split( X, Tail, Small, Big),
     quicksort( Small, SortedSmall),
     quicksort( Big, SortedBig),
     conc( SortedSmall, [X|SortedBig], Sorted).

split( X, [], [], []).
split( X, [Y|Tail], [Y|Small], Big)  :-
     gt( X, Y), !, split( X, Tail, Small, Big).
split( X, [Y|Tail], Small, [Y|Big])  :-
     split( X, Tail, Small, Big).
```

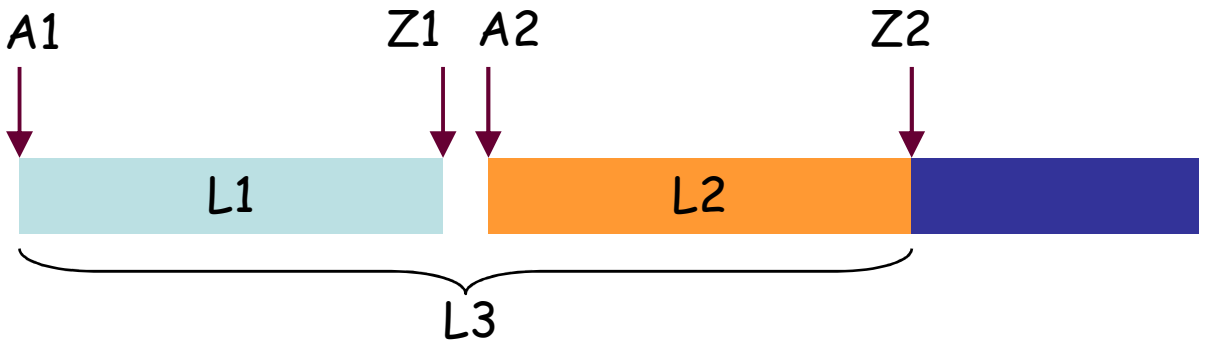
Sorting lists

```
| ?- quicksort([3,5,2,4], L).
    1  1  Call: quicksort([3,5,2,4],_24) ?
    2  2  Call: split(3,[5,2,4],_95,_96) ?
    3  3  Call: gt(3,5) ?
    4  4  Call: 3>5 ?
    4  4  Fail: 3>5 ?
    3  3  Fail: gt(3,5) ?
    3  3  Call: split(3,[2,4],_123,_82) ?
    4  4  Call: gt(3,2) ?
    5  5  Call: 3>2 ?
    5  5  Exit: 3>2 ?
    4  4  Exit: gt(3,2) ?
    6  4  Call: split(3,[4],_110,_82) ?
    7  5  Call: gt(3,4) ?
    8  6  Call: 3>4 ?
    8  6  Fail: 3>4 ?
    7  5  Fail: gt(3,4) ?
    7  5  Call: split(3,[],_110,_188) ?
    7  5  Exit: split(3,[],[],[]) ?
    6  4  Exit: split(3,[4],[],[4]) ?
    3  3  Exit: split(3,[2,4],[2],[4]) ?
    2  2  Exit: split(3,[5,2,4],[2],[5,4]) ?
    8  2  Call: quicksort([2],_257) ?
    9  3  Call: split(2,[],_283,_284) ?
    9  3  Exit: split(2,[],[],[]) ?
   10  3  Call: quicksort([],_308) ?
   10  3  Exit: quicksort([],[]) ?
   11  3  Call: quicksort([],_333) ?
   11  3  Exit: quicksort([],[]) ?
   12  3  Call: conc([],[2],_361) ?
   12  3  Exit: conc([],[2],[2]) ?
    8  2  Exit: quicksort([2],[2]) ?
   13  2  Call: quicksort([5,4],_387) ?
   14  3  Call: split(5,[4],_413,_414) ?
   15  4  Call: gt(5,4) ?
   16  5  Call: 5>4 ?
   16  5  Exit: 5>4 ?
   15  4  Exit: gt(5,4) ?
   17  4  Call: split(5,[],_400,_491) ?
   17  4  Exit: split(5,[],[],[]) ?
   14  3  Exit: split(5,[4],[4],[]) ?
   18  3  Call: quicksort([4],_517) ?
   19  4  Call: split(4,[],_543,_544) ?
   19  4  Exit: split(4,[],[],[]) ?
   20  4  Call: quicksort([],_568) ?
   20  4  Exit: quicksort([],[]) ?
   21  4  Call: quicksort([],_593) ?
   21  4  Exit: quicksort([],[]) ?
   22  4  Call: conc([],[4],_621) ?
   22  4  Exit: conc([],[4],[4]) ?
   18  3  Exit: quicksort([4],[4]) ?
   23  3  Call: quicksort([],_647) ?
   23  3  Exit: quicksort([],[]) ?
   24  3  Call: conc([4],[5],_675) ?
   25  4  Call: conc([],[5],_662) ?
   25  4  Exit: conc([],[5],[5]) ?
   24  3  Exit: conc([4],[5],[4,5]) ?
   13  2  Exit: quicksort([5,4],[4,5]) ?
   26  2  Call: conc([2],[3,4,5],_24) ?
   27  3  Call: conc([],[3,4,5],_719) ?
   27  3  Exit: conc([],[3,4,5],[3,4,5]) ?
   26  2  Exit: conc([2],[3,4,5],[2,3,4,5]) ?
    1  1  Exit: quicksort([3,5,2,4],[2,3,4,5]) ?
L = [2,3,4,5] ?
(78 ms) yes
{trace}
```

Sorting lists

- Quick sort:
  - If the list is split into two lists of approximately equal lengths then the time complexity of this sorting procedure is of the order $n\log n$, where n is the length of the list to be sorted.
  - If splitting always results in one list far bigger than the other, then the complexity is in the order of n^2.
  - The program in Figure 9.2 is not a good implementation because using the concatenation operation.
  - The program in Figure 9.3 is a more efficient implementation of **quicksort** using difference-pair representation for list.

Sorting lists



- To use the difference-pair representation in the sorting procedure, the list in the program of Figure 9.2 can be represented by pairs of lists of the form **A-Z** as follows (see Page 186):

  SortedSmall is represented by **A1 – Z1**

  SortedBig is represented by **A2 – Z2**

- The resulting concatenated list is represented by

  A1 – Z2 (and **Z1 = [X|A2])**

Sorting lists

```
% Figure 9.3  A more efficient implementation of quicksort using
     difference-pair representation for lists.

quicksort( List, Sorted)  :-
      quicksort2( List, Sorted - [] ).

quicksort2( [], Z - Z).
quicksort2( [X | Tail], A1 - Z2)  :-
     split( X, Tail, Small, Big),
     quicksort2( Small, A1 - [X | A2] ),
     quicksort2( Big, A2 - Z2).


split( X, [], [], []).
split( X, [Y|Tail], [Y|Small], Big)  :-
     gt( X, Y), !, split( X, Tail, Small, Big).
split( X, [Y|Tail], Small, [Y|Big])  :-
     split( X, Tail, Small, Big).
```

Sorting lists

```
|?- quicksort([3,5,2,4], L).

 1  1  Call: quicksort([3,5,2,4],_24) ?
 2  2  Call: quicksort2([3,5,2,4],_24-[]) ?
 3  3  Call: split(3,[5,2,4],_122,_123) ?
 4  4  Call: gt(3,5) ?
 5  5  Call: 3>5 ?
 5  5  Fail: 3>5 ?
 4  4  Fail: gt(3,5) ?
 4  4  Call: split(3,[2,4],_150,_109) ?
 5  5  Call: gt(3,2) ?
 6  6  Call: 3>2 ?
 6  6  Exit: 3>2 ?
 5  5  Exit: gt(3,2) ?
 7  5  Call: split(3,[4],_137,_109) ?
 8  6  Call: gt(3,4) ?
 9  7  Call: 3>4 ?
 9  7  Fail: 3>4 ?
 8  6  Fail: gt(3,4) ?
 8  6  Call: split(3,[],_137,_215) ?
 8  6  Exit: split(3,[],[],[]) ?
 7  5  Exit: split(3,[4],[],[4]) ?
 4  4  Exit: split(3,[2,4],[2],[4]) ?
 3  3  Exit: split(3,[5,2,4],[2],[5,4]) ?
 9  3  Call: quicksort2([2],_24-[3|_250]) ?
10  4  Call: split(2,[],_315,_316) ?
10  4  Exit: split(2,[],[],[]) ?
11  4  Call: quicksort2([],_24-[2|_306]) ?
11  4  Exit: quicksort2([],[2|_306]-[2|_306]) ?
12  4  Call: quicksort2([],_306-[3|_250]) ?
12  4  Exit: quicksort2([],[3|_250]-[3|_250]) ?
 9  3  Exit: quicksort2([2],[2,3|_250]-[3|_250]) ?
13  3  Call: quicksort2([5,4],_250-[]) ?
14  4  Call: split(5,[4],_428,_429) ?
15  5  Call: gt(5,4) ?
16  6  Call: 5>4 ?
16  6  Exit: 5>4 ?
15  5  Exit: gt(5,4) ?
17  5  Call: split(5,[],_415,_506) ?
17  5  Exit: split(5,[],[],[]) ?
14  4  Exit: split(5,[4],[4],[]) ?
18  4  Call: quicksort2([4],_250-[5|_498]) ?
19  5  Call: split(4,[],_563,_564) ?
19  5  Exit: split(4,[],[],[]) ?
20  5  Call: quicksort2([],_250-[4|_554]) ?
20  5  Exit: quicksort2([],[4|_554]-[4|_554]) ?
21  5  Call: quicksort2([],_554-[5|_498]) ?
21  5  Exit: quicksort2([],[5|_498]-[5|_498]) ?
18  4  Exit: quicksort2([4],[4,5|_498]-[5|_498]) ?
22  4  Call: quicksort2([],_498-[]) ?
22  4  Exit: quicksort2([],[]-[]) ?
13  3  Exit: quicksort2([5,4],[4,5]-[]) ?
 2  2  Exit: quicksort2([3,5,2,4],[2,3,4,5]-[]) ?
 1  1  Exit: quicksort([3,5,2,4],[2,3,4,5]) ?

L = [2,3,4,5] ?

(31 ms) yes
{trace}
```

Summary

- Operations on data structure
- Sorting
  - Bubble sort
  - Insertion sort
  - Quick sort

Check your understanding

- Write a procedure to merge two sorted lists producing a third list. For example:
- ?- merge([2,5,6,6,8], [1,3,5,9],L).

  L : [1,2,3,5,5,6,6,8,9