

1. Solve exercises 3.2.1 (i), (ii), (iii), and (iv).

i. A variant of Program 3.14 for sublist is defined by the following three rules:
`subsequence([X|Xs],[X|Ys]) :- subsequence(Xs,Ys).`
`subsequence(Xs,[Y|Ys]) :- subsequence(Xs,Ys).`
`subsequence([],Ys).`

Explain why this program has different meaning from Program 3.14

```
% sublist(Sub,List): Sub is a sublist of List.

% a: suffix of a prefix
sublist(Xs,Ys) :- prefix(Ps,Ys), suffix(Xs,Ps).

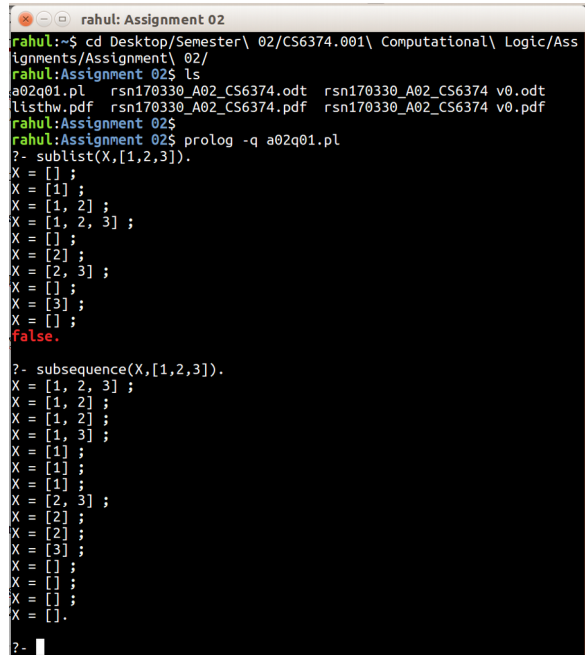
% b: prefix of a suffix
sublist(Xs,Ys) :- prefix(Xs,Ss), suffix(Ss,Ys).

% c: recursive definition of a sublist
sublist(Xs,Ys) :- prefix(Xs,Ys).
sublist(Xs,[Y|Ys]) :- sublist(Xs,Ys).
```

Program 3.14 Determining sublist of lists

Ans.

For sublist, the list is broken into all possible connected subsets.
 Whereas for subsequence, it is broken into all possible disconnected/connected subsets.



```

rahul: Assignment 02
rahul:~$ cd Desktop/Semester\ 02\CS6374.001\ Computational\ Logic\Assignments\Assignment\ 02/
rahul:Assignment 02$ ls
a02q01.pl  rsn170330_A02_CS6374.odt  rsn170330_A02_CS6374 v0.odt
listhw.pdf rsn170330_A02_CS6374.pdf  rsn170330_A02_CS6374 v0.pdf
rahul:Assignment 02$
rahul:Assignment 02$ prolog -q a02q01.pl
?- sublist(X,[1,2,3]).
X = [] ;
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
X = [] ;
X = [2] ;
X = [2, 3] ;
X = [] ;
X = [3] ;
X = [] ;
false.

?- subsequence(X,[1,2,3]).
X = [1, 2, 3] ;
X = [1, 2] ;
X = [1, 2] ;
X = [1, 3] ;
X = [1] ;
X = [1] ;
X = [1] ;
X = [2, 3] ;
X = [2] ;
X = [2] ;
X = [3] ;
X = [] ;
X = [] ;
X = [] ;
X = [] ;
?-

```

ii. Write recursive programs for adjacent and last that have the same meaning as the predicates defined in the text in terms of append.

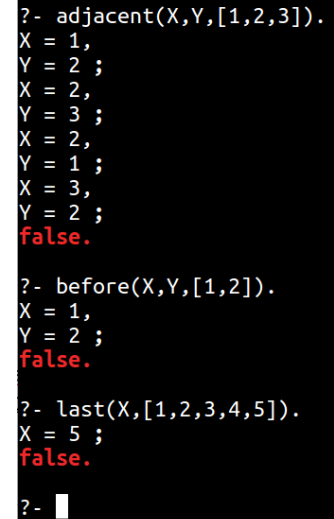
Ans.

```
% before(X,Y,List): X is before Y in List
before(X,Y,[X,Y|_]).
before(X,Y,[_|T]) :-
    before(X,Y,T).

% after(X,Y,List): X is after Y in List
after(X,Y,[Y,X|_]).
after(X,Y,[_|T]) :-
    after(X,Y,T).

% adjacent(X,Y,List): X is adjacent to Y in List
adjacent(X,Y,List) :-
    before(X,Y,List);
    after(X,Y,List).

% last(X,List): X is last element in List
last(X,[X]).
last(X,[_|T]) :-
    last(X,T).
```



```

?- adjacent(X,Y,[1,2,3]).
X = 1,
Y = 2 ;
X = 2,
Y = 3 ;
X = 2,
Y = 1 ;
X = 3,
Y = 2 ;
false.

?- before(X,Y,[1,2]).
X = 1,
Y = 2 ;
false.

?- last(X,[1,2,3,4,5]).
X = 5 ;
false.

?-

```

iii. Write a program for `double(List,ListList)`, where every element in the List appears twice in ListList, e.g., `double([1,2,3],[1,1,2,2,3,3])` is true.

Ans.

```
% double(L,LL): every element of L occurs twice in LL
double([],[]).
double([H|T1],[H,H|T2]) :-
    double(T1,T2).
```

```
?- double([1,2,3],X).
X = [1, 1, 2, 2, 3, 3].

?- double(X,[1,1,2,3,4,5]).
X = [1, 2, 3, 4, 5].

?-
```

iv. Compute the size of the proof tree as a function of the size of the input list for Programs 3.16a and 3.16b defining reverse.

```
% reverse(List,Tsil): Tsil is the result
of reversing the list List.

% a: Naive reverse

reverse([],[]).
reverse([X|Xs],Zs) :-
    reverse(Xs,Ys),
    append(Ys,[X],Zs).

% b: Reverse-accumulate
reverse(Xs,Ys) :-
    reverse(Xs,[],Ys).

reverse([X|Xs],Acc,Ys) :-
    reverse(Xs,[X|Acc],Ys).
reverse([],Ys,Ys).

Program 3.16 Reversing a list
```

Ans.

Lets check with the trace command how many times we need to call (calls in the proof tree) for different values of list size n.

For naive reverse:

n	1	2	3	4	5
calls	6	12	20	30	42

Hence, $\text{calls}(n) = n^2 + 3n + 2$

For reverse-accumulate

n	1	2	3	4	5
calls	6	8	10	12	14

Hence, $\text{calls}(n) = 2n + 4$

So, naive reverse/3.16a is $O(n^2)$ whereas reverse-accumulate/3.16b is $O(n)$, which is far better than $O(n^2)$.

*You can refer from this image, where left is for naive and right for rev-acc.

```
Terminal
x - rahul: Assignment 02

[trace] ?- reverse([1,2],X).
Call: (8) reverse([1, 2], _5752)
Call: (9) reverse([2], _5984)
Call: (10) reverse([], _5984)
Exit: (10) reverse([], [])
Call: (10) append([], [2], _5992)
Exit: (10) append([], [2], [2])
Exit: (9) reverse([2], [2])
Call: (9) append([2], [1], _5752)
Call: (10) append([1], [1], _5982)
Exit: (10) append([1], [1], [1])
Exit: (9) append([2], [1], [2, 1])
Exit: (8) reverse([1, 2], [2, 1])
X = [2, 1].

[trace] ?- reverse([1,2,3],X).
Call: (8) reverse([1, 2, 3], _5408)
Call: (9) reverse([2, 3], _5650)
Call: (10) reverse([3], _5650)
Call: (11) reverse([], _5650)
Exit: (11) reverse([], [])
Call: (11) append([], [3], _5658)
Exit: (11) append([], [3], [3])
Exit: (10) reverse([3], [3])
Call: (10) append([3], [2], _5664)
Call: (11) append([1], [2], _5648)
Exit: (11) append([1], [2], [2])
Exit: (10) append([3], [2], [3, 2])
Call: (9) reverse([2, 3], [3, 2])
Call: (9) append([3, 2], [1], _5408)
Call: (10) append([2], [1], _5660)
Call: (11) append([1], [1], _5666)
Exit: (11) append([1], [1], [1])
Exit: (10) append([2], [1], [2, 1])
Exit: (9) append([3, 2], [1], [3, 2, 1])
Exit: (8) reverse([1, 2, 3], [3, 2, 1])
X = [3, 2, 1].

Terminal
x - rahul: Assignment 02

[trace] ?- reverse([1,2],X).
Call: (8) reverse([1, 2], _7156)
Call: (9) reverse([1, 2], [], _7156)
Call: (10) reverse([2], [1], _7156)
Call: (11) reverse([], [2, 1], _7156)
Exit: (11) reverse([], [2, 1], [2, 1])
Exit: (10) reverse([2], [1], [2, 1])
Exit: (9) reverse([1, 2], [1], [2, 1])
Exit: (8) reverse([1, 2], [2, 1])
X = [2, 1].

[trace] ?- reverse([1,2,3],X).
Call: (8) reverse([1, 2, 3], _7162)
Call: (9) reverse([1, 2, 3], [], _7162)
Call: (10) reverse([2, 3], [1], _7162)
Call: (11) reverse([3], [2, 1], _7162)
Call: (12) reverse([], [3, 2, 1], _7162)
Exit: (12) reverse([], [3, 2, 1], [3, 2, 1])
Exit: (11) reverse([3], [2, 1], [3, 2, 1])
Exit: (10) reverse([2, 3], [1], [3, 2, 1])
Exit: (9) reverse([1, 2, 3], [1], [3, 2, 1])
Exit: (8) reverse([1, 2, 3], [3, 2, 1])
X = [3, 2, 1].

[trace] ?- reverse([1,2,3,4],X).
Call: (8) reverse([1, 2, 3, 4], _7168)
Call: (9) reverse([1, 2, 3, 4], [], _7168)
Call: (10) reverse([2, 3, 4], [1], _7168)
Call: (11) reverse([3, 4], [2, 1], _7168)
Call: (12) reverse([4], [3, 2, 1], _7168)
Call: (13) reverse([], [4, 3, 2, 1], _7168)
Exit: (13) reverse([], [4, 3, 2, 1], [4, 3, 2, 1])
Exit: (12) reverse([4], [3, 2, 1], [4, 3, 2, 1])
Exit: (11) reverse([3, 4], [2, 1], [4, 3, 2, 1])
Exit: (10) reverse([2, 3, 4], [1], [4, 3, 2, 1])
Exit: (9) reverse([1, 2, 3, 4], [1], [4, 3, 2, 1])
Exit: (8) reverse([1, 2, 3, 4], [4, 3, 2, 1])
X = [4, 3, 2, 1].
```

2. Solve exercises 3.3.1 (i), (ii), (iii), (v), (vi), (vii).

i. Write a program for `substitute(X,Y,L1,L2)`, where `L2` is the result of substituting `Y` for all occurrences of `X` in `L1`, e.g., `substitute(a,x,[a,b,a,c],[x,b,x,c])` is true, whereas `substitute(a,x,[a,b,a,c],[a,b,x,c])` is false.

Ans.

```
% substitute(X,Y,L1,L2): where L2 is the result of
% substituting Y for all occurrences of X in L1
substitute(X,Y,[],[]).
```

```
substitute(X,Y,[X|T1],[Y|T2]) :-
    substitute(X,Y,T1,T2).
```

```
substitute(X,Y,[Z|T1],[Z|T2]) :-
    Z \= X,
    substitute(X,Y,T1,T2).
```

```
?- substitute(a,x,[a,b,a,c],X).
X = [x, b, x, c] .

?- substitute(a,x,[a,b,a,c],[x,b,x,c]).
true .

?- substitute(a,x,[a,b,a,c],[a,b,x,c]).
false.

?- 
```

ii. What is the meaning of the variant of the select:

```
select(X,[X|Xs],Xs).
```

```
select(X,[Y|Ys],[Y|Zs]) :- X != Y, select(X,Ys,Zs).
```

Ans.

From fig below, first output of `select()` corresponds to above variant with `X!=Y`, whereas second `select()` output is without it (commented one).

Hence, above variant only outputs selecting first instance [Deterministic] whereas, original variant of `select` outputs all possible solutions, selecting every instances at a time [Non-Deterministic].

```
a02q01.pl
1 % substitute(X,Y,L1,L2): where L2 is the result of
2 % substituting Y for all occurrences of X in L1
3
4 substitute(X,Y,[X|T1],[Y|T1]).
5 substitute(X,Y,[H1|T1],[H1|T2]) :-
6     %substitute(X,Y,T1,T2),
7     append(L1,[X|L2],T1),
8     append(L1,[Y|L3],T2),
9     append(M1,[X|R1],L2),
10    append(M1,[Y|R2],L3).
11
12 % select(X,List,R): R is a List with one removed X
13 select(X,[X|T],T).
14 select(X,[H|T],[H|R]) :-
15     %X \= H,
16     select(X,T,R).
17

rahul: Assignment 02
?- select(s,[a,c,s,b,s,d,f],R).
R = [a, c, b, s, d, f] ;
false.

?- ['a02q02.pl'].
Warning: /home/rahul/Desktop/Semester_02/CS6374.001 Computational Log
ic/Assignments/Assignment_02/a02q02.pl:5:
Singleton variables: [R1,R2]
true.

?- select(s,[a,c,s,b,s,d,f],R).
R = [a, c, b, s, d, f] ;
R = [a, c, s, b, d, f] ;
false.

?- 
```

iii. Write a program for `no_doubles(L1,L2)`, where `L2` is the result of removing all duplicate elements from `L1`, e.g., `no_doubles([a,b,c,b],[a,c,b])` is true. (Hint: use `member`.)

Ans.

```
% no_doubles(L1,L2): L2 is list of all distinct
% values in L1, keeping last instance.
no_doubles([],[]).
```

```
no_doubles([H|T],R) :-
    member(H,T),
    no_doubles(T,R).
```

```
no_doubles([H|T],[H|R]) :-
    \+ member(H,T),
    no_doubles(T,R).
```

```
?- no_doubles([1,2,3,3,2,4,1],X).
X = [3, 2, 4, 1] ;
false.

?- no_doubles([a,b,c,b],[a,c,b]).
true .

?- 
```

v. Write a program for merge sort.

Ans.

```
% mergeSort(ListX,ListR): sorts ListX to give ListR
mergeSort([], []).
mergeSort([X],[X]).
```

```
mergeSort([X|T],ListR) :- splitHalves([X|T],L,R),
    mergeSort(L,L1),
    mergeSort(R,R1),
    merge(L1,R1,ListR).
```

```
%splitHalves(List,ListL,ListR): split List into
%      two halves ListL and ListR.
splitHalves([],[], []).
splitHalves([X],[X], []).
```

```
splitHalves([H|T1],[H|L1],R) :- removeLast(T1,T),
    splitHalves(T,L1,R1),
    last(S,T1),
    reverse([S|R1],R).
```

```
% last(X,List): X is last element in List
last(X,List) :- reverse(List,[X|_]).
```

```
%removeLast(List,R): removing last element
%      from List gives R.
removeLast([], []).
removeLast(X,R) :- reverse(X,[_|T]),
    reverse(T,R).
```

```
% b: Reverse-accumulate
reverse(Xs,Ys) :- reverse(Xs,[],Ys).
```

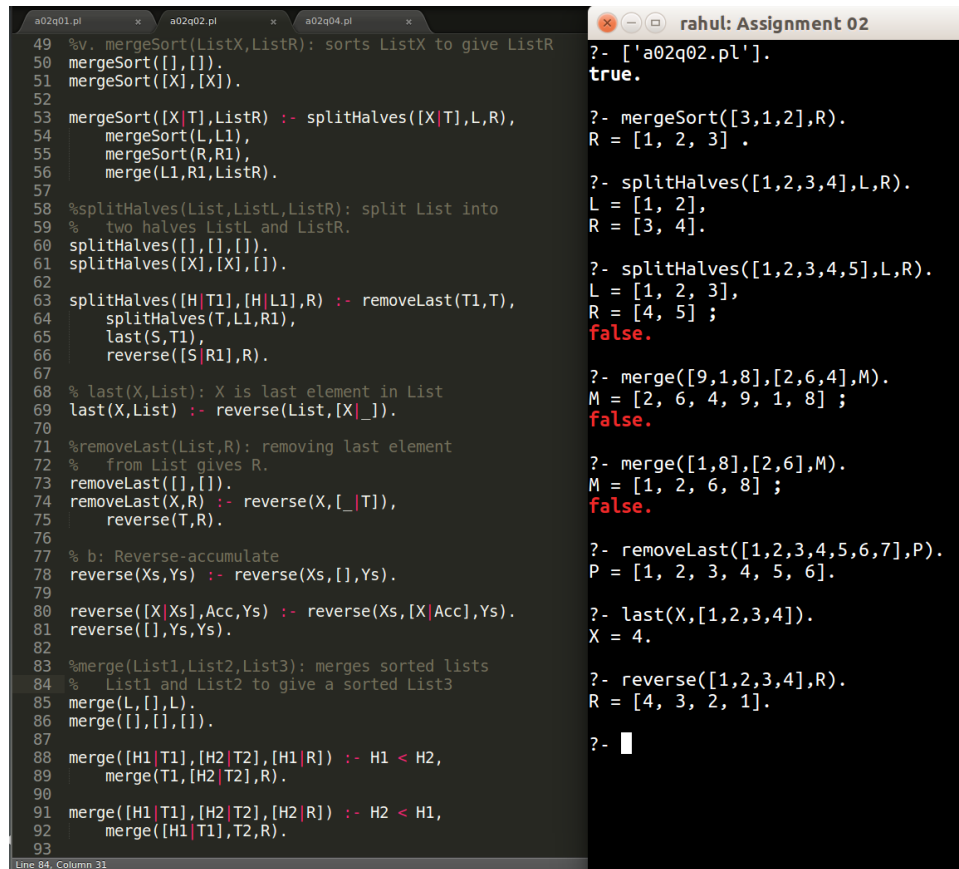
```
reverse([X|Xs],Acc,Ys) :- reverse(Xs,[X|Acc],Ys).
reverse([],Ys,Ys).
```

```
%merge(List1,List2,List3): merges sorted lists
%      List1 and List2 to give a sorted List3
merge(L,[],L).
merge([],[], []).
```

```
merge([H1|T1],[H2|T2],[H1|R]) :- H1 < H2,
    merge(T1,[H2|T2],R).
```

```
merge([H1|T1],[H2|T2],[H2|R]) :- H2 < H1,
    merge([H1|T1],T2,R).
```

Following is the screen-shot of the run.



The screenshot shows a MATLAB script editor with a file named 'a02q02.pl' and a command window titled 'rahul: Assignment 02'. The script defines several functions for merge sort, including `mergeSort`, `splitHalves`, `removeLast`, `reverse`, and `merge`. The command window shows the execution of these functions with various inputs, resulting in sorted lists and boolean values.

```
49 %v. mergeSort(ListX,ListR): sorts ListX to give ListR
50 mergeSort([],[]).
51 mergeSort([X],X).
52
53 mergeSort([X|T],ListR) :- splitHalves([X|T],L,R),
54     mergeSort(L,L1),
55     mergeSort(R,R1),
56     merge(L1,R1,ListR).
57
58 %splitHalves(List,ListL,ListR): split List into
59 % two halves ListL and ListR.
60 splitHalves([],[],[]).
61 splitHalves([X],X,[],[]).
62
63 splitHalves([H|T1],[H|L1],R) :- removeLast(T1,T),
64     splitHalves(T,L1,R1),
65     last(S,T1),
66     reverse([S|R1],R).
67
68 % last(X,List): X is last element in List
69 last(X,List) :- reverse(List,[X|_]).
70
71 %removeLast(List,R): removing last element
72 % from List gives R.
73 removeLast([],[]).
74 removeLast(X,R) :- reverse(X,[_|T]),
75     reverse(T,R).
76
77 % b: Reverse-accumulate
78 reverse(Xs,Ys) :- reverse(Xs,[],Ys).
79
80 reverse([X|Xs],Acc,Ys) :- reverse(Xs,[X|Acc],Ys).
81 reverse([],Ys,Ys).
82
83 %merge(List1,List2,List3): merges sorted lists
84 % List1 and List2 to give a sorted List3
85 merge(L,[],L).
86 merge([],L,L).
87
88 merge([H1|T1],[H2|T2],[H1|R]) :- H1 < H2,
89     merge(T1,[H2|T2],R).
90
91 merge([H1|T1],[H2|T2],[H2|R]) :- H2 < H1,
92     merge([H1|T1],T2,R).
93
```

```
?- ['a02q02.pl'].
true.

?- mergeSort([3,1,2],R).
R = [1, 2, 3] .

?- splitHalves([1,2,3,4],L,R).
L = [1, 2],
R = [3, 4].

?- splitHalves([1,2,3,4,5],L,R).
L = [1, 2, 3],
R = [4, 5] ;
false.

?- merge([9,1,8],[2,6,4],M).
M = [2, 6, 4, 9, 1, 8] ;
false.

?- merge([1,8],[2,6],M).
M = [1, 2, 6, 8] ;
false.

?- removeLast([1,2,3,4,5,6,7],P).
P = [1, 2, 3, 4, 5, 6].

?- last(X,[1,2,3,4]).
X = 4.

?- reverse([1,2,3,4],R).
R = [4, 3, 2, 1].

?-
```

vi. Write a program for `kth_largest(Xs,K)` that implements the linear algorithm for finding the kth largest element K of a list Xs. The algorithm has following steps:

1. Break the list into the groups of five elements.
2. Efficiently find the median of each of the groups, which can be done with a fixed no of comparisons.
3. Recursively find the median of the medians.
4. Partition the original list with respect to the median of medians.
5. Recursively find the kth largest element in the appropriate smaller list.

Ans.

Please, refer file `a02q02.pl`. Here is the output for `kth_largest`.

```

93 %----- V1. -----
94 % KTH_LARGEST
95
96 %kth_largest(KL,K,List): KL is Kth largest element
97 % from the List.
98
99 kth_largest(KL,K,List) :-
100
101     split5(List,List5),
102     medians(List5,MediansL),
103     medianM(MediansL,MedOfMed),
104
105     partition(List,MedOfMed,Ls,Bs),
106     lengthL(Bs,B),
107     P is B+1,
108
109     ( P == K ->
110         KL is MedOfMed;
111     ( P > K ->
112         kth_largest(KL,K,Bs);
113         KL is K-P,
114         kth_largest(KL,KL,Ls)
115     )
116     ).
117
118 % algorithm referred from:
119 % https://www.geeksforgeeks.org/kth-smallestlargest-e
120
121 %-----
122 % DONT TOUCH THIS
123
124 % given a list List, splits List into group of
125 % 5 elements as R.
126 split5(List, R) :-
127     lengthL(List,D),
128     M is mod(D,5),
129     ( M == 0 ->
130         ( 5 < D ->
131             J is D-M, slice(List,1,J,List1),
132             I is J+1, slice(List,I,D,List2),
133             addList(List2,[],List3),
134             group(List1,R1),
135             append(R1,List3,R);
136         makeList(List,R));
137
138 ?- ['a02q02.pl'].
139 true.
140
141 ?- split5([1,3,4,9,10,12,13,2,5,6,7,8,11,14],R).
142 R = [[1, 3, 4, 9, 10], [12, 13, 2, 5, 6], [7, 8, 11, 14]] .
143
144 ?- split5([1,3,4,9,10,12,13,2,5,6,7,8,11,14],R), medians(R,M).
145 R = [[1, 3, 4, 9, 10], [12, 13, 2, 5, 6], [7, 8, 11, 14]],
146 M = [4, 6, 8] .
147
148 ?- medianM([4,6,8],MOM).
149 MOM = 6 .
150
151 ?- kth_largest(KL,3,[1,3,4,9,10,12,13,2,5,6,7,8,11,14]).
152 KL = 12 .
153
154 ?- partition([1,3,4,9,10,12,13,2,5,6,7,8,11,14],6,Ls,Bs), lengthL(Bs,B).
155 Ls = [1, 3, 4, 2, 5, 6],
156 Bs = [9, 10, 12, 13, 7, 8, 11, 14],
157 B = 8.
158
159 ?- partition([1,3,4,9,10,12,13,2,5,6,7,8,11,14],6,Ls,Bs), kth_largest(KL,3,Bs).
160 Ls = [1, 3, 4, 2, 5, 6],
161 Bs = [9, 10, 12, 13, 7, 8, 11, 14],
162 KL = 12 .
163
164 ?- slice([1,3,4,9,10,12,13,2,5,6,7,8,11,14],2,6,S), group(S,L5).
165 S = [3, 4, 9, 10, 12],
166 L5 = [[3, 4, 9, 10, 12]] .
167
168 ?- slice([1,3,4,9,10,12,13,2,5,6,7,8,11,14],2,4,S1), addList(S1,[],R).
169 S1 = [3, 4, 9],
170 R = [[3, 4, 9]] .
171
172 ?-

```

vii. Write a program for the relation `better_poker_hand(Hand1,Hand2,Hand)` that succeeds when Hand is better poker hand between Hand1 and Hand2. For those unfamiliar with this card game, here are some rules of poker necessary for answering this exercise:

- The order of cards is 2,3,4,5,6,7,8,9,10,jack,queen,king,ace.
- Each hand consists of five cards.
- The rank of hands in ascending order is no pairs < one pair < two pairs < three of a kind < flush < straight < full house < four of a kind < straight flush.
- Where two cards have the same rank, the higher denomination wins, for example, a pair of kings beats a pair of 7's.

Hints:

- Represent a poker hand by a list of terms of the form `card(Suit,Value)`. For example a hand consisting of the 2 of clubs, the 5 of spades, the queen of hearts, the queen of diamonds, and the 7 of spades would be represented by the list `[card(clubs,2), card(spades,5), card(hearts,queen), card(diamonds,queen), card(spades,7)]`.
- It may be helpful to define relations such as `has-flush(Hand)`, which is true if all the cards in Hand are of the same suit; `has-full-house(Hand)`, which is true if Hand has three cards with the same value but in different suits, and the other two cards have the same different value; and `has-straight(Hand)`, which is true if Hand has cards with consecutive values.
- The number of cases to consider is reduced if the hand is first sorted.

Ans.

?- ['pokerA2.pl'].

Please, refer the file 'pokerA2.pl' attached herewith.

Following are the screen shots of the runs.

The first screenshot shows a MATLAB command window titled 'rahul: Assignment 02' with the following session:

```
?- ['pokerA2.pl'].
true.

?- card(diamonds,queen).
true.

?- card(diamonds,soldier).
false.

?- greaterValue(X,10).
X = jack ;
X = queen ;
X = king ;
X = ace ;
false.

?- greaterType(flush,Y).
Y = threeOfAKind ;
Y = twoPairs ;
Y = onePair ;
Y = noPair ;
false.

?- validHand([card(spades,2),card(spades,3),card(spades,jack),card(spades,queen),card(clubs,jack)]).
true.

?- validHand([card(spades,2),card(spades,2),card(spades,jack),card(spades,queen),card(clubs,jack)]).
false.

?- no_doubles([card(spades,2),card(spades,2),card(spades,jack),card(spades,queen),card(clubs,jack)],R).
R = [card(spades, 2), card(spades, jack), card(spades, queen), card(clubs, jack)] ;
false.

?-
```

The second screenshot shows the script file 'pokerA2.pl' with the following code:

```
63 card(spades, queen).
64 card(spades, king).
65 card(spades, ace).
66
67 % 2,3,4,5,6,7,8,9,10,jack,queen,king,ace
68
69 immediateGreaterValue(ace,king). immediateGreaterValue(king,queen).
70 immediateGreaterValue(queen,jack). immediateGreaterValue(jack,10).
71 immediateGreaterValue(10,9). immediateGreaterValue(9,8).
72 immediateGreaterValue(8,7). immediateGreaterValue(7,6).
73 immediateGreaterValue(6,5). immediateGreaterValue(5,4).
74 immediateGreaterValue(4,3). immediateGreaterValue(3,2).
75
76 greaterValue(X,Y) :- immediateGreaterValue(X,Y).
77
78 greaterValue(X,Y) :-
79     immediateGreaterValue(Z,Y), greaterValue(X,Z).
80
81 % no pairs < one pair < two pairs < three of a kind < flush <
82 % straight < full house < four of a kind < straight flush.
83
84 immediateGreaterType(straightFlush, fourOfAKind).
85 immediateGreaterType(fourOfAKind, fullHouse).
86 immediateGreaterType(fullHouse, straight).
87 immediateGreaterType(straight, flush).
88 immediateGreaterType(flush, threeOfAKind).
89 immediateGreaterType(threeOfAKind, twoPairs).
90 immediateGreaterType(twoPairs, onePair).
91 immediateGreaterType(onePair, noPair).
92
93 greaterType(X,Y) :- immediateGreaterType(X,Y).
94
95 greaterType(X,Y) :-
96     immediateGreaterType(Z,Y), greaterType(X,Z).
97
98 %----- VALIDATION -----
99
100 validHand(Hand) :-
101     length(Hand,5),
102     no_doubles(Hand,Hand).
103
104 % no doubles(L1,L2): L2 is list of all distinct values in L1.
```

The third screenshot shows a MATLAB command window titled 'rahul: Assignment 02' with the following session:

```
?- partition([card(spades,2),card(spades,3),card(spades,jack),card(spades,queen),card(clubs,jack)],card(spades,jack),L,R).
L = [card(spades, 2), card(spades, 3)],
R = [card(spades, jack), card(spades, queen), card(clubs, jack)].

?- quicksort([card(spades,2),card(spades,3),card(spades,jack),card(spades,queen),card(clubs,jack)],S).
S = [card(spades, 2), card(spades, 3), card(spades, jack), card(clubs, jack), card(spades, queen)].

?- quicksort([card(spades,2),card(spades,2),card(spades,jack),card(spades,queen),card(clubs,jack)],S).
S = [card(spades, 2), card(spades, 2), card(spades, jack), card(clubs, jack), card(spades, queen)].

?-
```

The fourth screenshot shows the script file 'pokerA2.pl' with the following code:

```
115 %----- PARTITIONING -----
116 % partitions [H|T] into Ls1 and Bs1 at P.
117 partition([],_,[],[]).
118
119 partition([card(Hs,H) | T],card(Ps,P),Ls1,Bs1) :-
120     (greaterValue(P,H) -> partition(T,card(Ps,P),Ls,Bs), Ls1 = [card(Hs,H) | Ls]) ;
121     partition(T,card(Ps,P),Ls,Bs), Ls1 = Ls, Bs1 = [card(Hs,H) | Bs].
122
123 %----- QUICKSORT -----
124 % quicksort(Hand,SortedHand): sorts Hand to SortedHand
125 quicksort([card(Xs,X) | T],Ys) :-
126     partition(T,card(Xs,X),Littles,Bigs),
127     quicksort(Littles,Ls),
128     quicksort(Bigs,Bs),
129     append(Ls,[card(Xs,X) | Bs],Ys).
130
131 quicksort([],[]).
132
```



```
raahul: Assignment 02

?- findHandType([card(spades,7),card(spades,9),card(diamonds,10),card
(spades,2),card(spades,4)],Type).
Type = noPair.

?- findHandType([card(spades,7),card(spades,9),card(diamonds,10),card
(spades,10),card(spades,4)],Type).
Type = onePair.

?- findHandType([card(spades,7),card(clubes,10),card(diamonds,10),car
d(spades,10),card(spades,4)],Type).
Type = threeOfAKind.

?- findHandType([card(spades,7),card(clubes,7),card(diamonds,10),card(
spades,10),card(spades,4)],Type).
Type = twoPairs.

?- findHandType([card(spades,7),card(clubes,8),card(diamonds,9),card(
spades,10),card(spades,jack)],Type).
Type = straight.

?- findHandType([card(spades,7),card(spades,2),card(spades,9),card(sp
ades,10),card(spades,jack)],Type).
Type = flush.

?- findHandType([card(spades,2),card(clubes,2),card(hearts,jack),card(
spades,jack),card(clubes,jack)],Type).
Type = fullHouse.

?- findHandType([card(spades,2),card(diamonds,jack),card(spades,jack)
,card(hearts,jack),card(clubes,jack)],Type).
Type = fourOfAKind.

?- findHandType([card(spades,7),card(spades,8),card(spades,9),card(sp
ades,10),card(spades,jack)],Type).
Type = straightFlush.

?- 
```

```
pokerA2.pl x scratch.pl x familyA1.pl x a02q02.pl x
203 %----- TIE-BREAKS -----
204 betterDenomination(Hand1, Hand2, Type, R) :-
205     ((Type == fullHouse; Type == fourOfAKind; Type == twoPairs;
206      Type == onePair; Type == noPair) ->
207      getDenomination5(Hand1, D1), getDenomination5(Hand2, D2),
208      (winner(D1, D2, D1) -> R = Hand1;
209       R = Hand2);
210      getDenomination3(Hand1, E1), getDenomination3(Hand2, E2),
211      (winner(E1, E2, E1) -> R = Hand1;
212       R = Hand2)).
213
214 %-----
215 % works for noPair, onePair, twoPairs, fourOfAKind, fullHouse
216 getDenomination5(Hand, D) :-
217     quicksort(Hand, S),
218     hasPair(S, D).
219
220
221 hasPair([card(_, X1), card(_, X2), card(_, X3), card(_, X4), card(_, X5)], Pair) :-
222     ( X4=X5 -> Pair = X4;
223      ( X4=X3 -> Pair = X3;
224       ( X3=X2 -> Pair = X2;
225        ( X2=X1 -> Pair = X1;
226         Pair = X5))))).
227
228 winner(D1, D2, D) :-
229     (greaterValue(D1, D2) -> D = D1;
230      D = D2).
231
232 %-----
233 % works for straight, flush, straightFlush
234 getDenomination3(Hand, D) :-
235     quicksort(Hand, S),
236     last(card(_, D), S).
237
238 %----- GAME -----
239
240 better_poker_hand(Hand1, Hand2, R) :-
241     findHandType(Hand1, Type1),
242     findHandType(Hand2, Type2),
243     (Type1 == Type2 ->
244      betterDenomination(Hand1, Hand2, Type1, R);
245      (greaterType(Type1, Type2) -> R = Hand1;
246       R = Hand2)).
247
248 %-----
249
250 ?- better_poker_hand([card(clubs,king),card(diamonds,jack),card(spade
s,king),card(clubs,3),card(hearts,10)],
[card(spades,queen),card(hearts,ace),card(diamonds,2),card(clubs,que
n),card(spades,jack)], H).
H = [card(clubs,king), card(diamonds,jack), card(spades,king), car
d(clubs,3), card(hearts,10)].
251
252 ?- findHandType([card(clubs,king),card(diamonds,jack),card(spades,kin
g),card(clubs,3),card(hearts,10)], Type1).
Type1 = onePair.
253
254 ?- findHandType([card(spades,queen),card(hearts,ace),card(diamonds,2)
,card(clubs,queen),card(spades,jack)], Type2).
Type2 = onePair.
255
256 ?- getDenomination5([card(clubs,king),card(diamonds,jack),card(spades
,king),card(clubs,3),card(hearts,10)], D1).
D1 = king.
257
258 ?- getDenomination3([card(clubs,king),card(diamonds,jack),card(spades
,king),card(clubs,3),card(hearts,10)], D1).
D1 = king.
259
260 ?- getDenomination5([card(spades,queen),card(hearts,ace),card(diamond
s,2),card(clubs,queen),card(spades,jack)], D2).
D2 = queen.
261
262 ?- getDenomination3([card(spades,queen),card(hearts,ace),card(diamond
s,2),card(clubs,queen),card(spades,jack)], D2).
D2 = ace.
263
264 ?- quicksort([card(spades,queen),card(hearts,ace),card(diamonds,2),ca
rd(clubs,queen),card(spades,jack)], S), hasPair(S, D).
S = [card(diamonds,2), card(spades,jack), card(spades,queen), card
(clubs,queen), card(hearts,ace)],
D = queen.
265
266 ?-
267
```



```

rahul: Assignment 02
?- ['pokerA2.pl'].
true.

?- better_poker_hand( [card(clubs,8),card(clubs,9),card(clubs,10),card(clubs,jack),card(clubs,queen)],
                      [card(spades,7),card(spades,8),card(spades,9),card(spades,10),card(spades,jack)], H).
H = [card(clubs, 8), card(clubs, 9), card(clubs, 10), card(clubs, jack), card(clubs, queen)].

?- better_poker_hand( [card(clubs,8),card(diamonds,8),card(spades,8),card(clubs,jack),card(spades,jack)],
                      [card(spades,7),card(spades,8),card(spades,9),card(spades,10),card(spades,jack)], H).
H = [card(spades, 7), card(spades, 8), card(spades, 9), card(spades, 10), card(spades, jack)].

?- better_poker_hand([card(clubs,king),card(diamonds,jack),card(spades,king),card(clubs,3),card(hearts,10)],
                      [card(spades,queen),card(hearts,ace),card(diamonds,2),card(clubs,queen),card(spades,jack)], H).
H = [card(clubs, king), card(diamonds, jack), card(spades, king), card(clubs, 3), card(hearts, 10)].

?- 
48
49 % straightFlush(queen) vs straightFlush(jack)
50 better_poker_hand([card(clubs,8),card(clubs,9),card(clubs,10),card(clubs,jack),card(clubs,queen)],
51 [card(spades,7),card(spades,8),card(spades,9),card(spades,10),card(spades,jack)], H).
52
53 % fullHouse(jack) vs straightFlush(jack)
54 better_poker_hand([card(clubs,8),card(diamonds,8),card(spades,8),card(clubs,jack),card(spades,jack)],
55 [card(spades,7),card(spades,8),card(spades,9),card(spades,10),card(spades,jack)], H).
56
57 % twoPairs(king) vs twoPairs(queen)
58 better_poker_hand([card(clubs,king),card(diamonds,jack),card(spades,king),card(clubs,3),card(hearts,10)],
59 [card(spades,queen),card(hearts,ace),card(diamonds,2),card(clubs,queen),card(spades,jack)], H).
60

```

*typo - last run of better_poker_hand is of type **onePair(King) vs onePair(queen)**.

3. Given the sorted binary tree (SBT) representation discussed in class, define the following functions

sumtree(T,N): N is the sum of elements in SBT T (use succ arithmetic).

delete(E,T,Tn): delete the element E from SBT T to obtain SBT Tn.

Ans.

```
%----- SumTree -----

sumTree(nil,0).

%sumTree(tree(X,nil,nil), X).

%sumTree(tree(X,nil,R),S) :-
%    sumTree(R,R1), plus(R1,X,S).

%sumTree(tree(X,L,nil),S) :-
%    sumTree(R,L1), plus(L1,X,S).

sumTree(tree(X,L,R),N) :-
    sumTree(L,N1), sumTree(R,N2),
    plus(N1,N2,N0), plus(X,N0,N).

%----- deletion -----
% delete(E,T,Tn): delete the element E from SBT T
%    to obtain SBT Tn.

% when E is leaf
deleteT(tree(X,nil,nil),T,Tn) :-
    substituteT(tree(X,nil,nil),nil,T,Tn).

% when E has a single child
deleteT(tree(X,nil,R),T,Tn) :-
    substituteT(tree(X,nil,R),R,T,Tn1),
    deleteT(R,Tn1,Tn).

deleteT(tree(X,L,nil),T,Tn) :-
    substituteT(tree(X,L,nil),L,T,Tn1),
    deleteT(L,Tn1,Tn).

% when E has two children
deleteT(tree(X,L,R),T,Tn) :-
    inOrderSuccessor(Xios,X,T),
    substitute(tree(X,L,R),tree(Xios,L,R),T,Tn1),
    deleteT(tree(Xios,L1,R1),Tn1,Tn).

% substitute(X,Y,TreeX,TreeY): TreeY is result of
% replacing all occurrences of X in TreeX with Y.
substituteT(X,Y,nil,nil).

substituteT(X,Y,tree(M,ML,MR),tree(N,NL,NR)) :-
    replace(X,Y,X,Y),
    substituteT(X,Y,ML,NL),
    substituteT(X,Y,MR,NR).

replace(X,Y,X,Y).
replace(X,Y,Z,Z) :- X \= Z.
```

```
% sublist(Sub,List): Sub is a sublist of List.
% c: recursive definition of a sublist
sublist(Xs,Ys) :- prefix(Xs,Ys).
sublist(Xs,[_:_:Ys]) :- sublist(Xs,Ys).
```

```
87 %----- SumTree -----
88
89 sumTree(nil,0).
90
91
92 %sumTree(tree(X,nil,nil), X).
93
94 %sumTree(tree(X,nil,R),S) :-
95 %   sumTree(R,R1), plus(R1,X,S).
96
97 %sumTree(tree(X,L,nil),S) :-
98 %   sumTree(R,L1), plus(L1,X,S).
99
100 sumTree(tree(X,L,R),N) :-
101   sumTree(L,N1), sumTree(R,N2),
102   plus(N1,N2,N0), plus(X,N0,N).
103
104 %----- deletion -----
105 % delete(E,T,Tn): delete the element E from SBT T
106 %   to obtain SBT Tn.
107
108 % when E is leaf
109 deleteT(tree(X,nil,nil),T,Tn) :-
110   substituteT(tree(X,nil,nil),nil,T,Tn).
111
112 % when E has a single child
113 deleteT(tree(X,nil,R),T,Tn) :-
114   substituteT(tree(X,nil,R),R,T,Tn1),
115   deleteT(R,Tn1,Tn).
116
117 deleteT(tree(X,L,nil),T,Tn) :-
118   substituteT(tree(X,L,nil),L,T,Tn1),
119   deleteT(L,Tn1,Tn).
120
121 % when E has two children
122 deleteT(tree(X,L,R),T,Tn) :-
123   inOrderSuccessor(Xios,X,T),
124   substitute(tree(X,L,R),tree(Xios,L,R),T,Tn1),
125   deleteT(tree(Xios,L1,R1),Tn1,Tn).
126
127 % substitute(X,Y,TreeX,TreeY): TreeY is result of
128 % replacing all occurrences of X in TreeX with Y.
129 substituteT(X,Y,nil,nil).
130
```

4. Exercises 8.3.1 (i), (iii), (vi), (vii)

i. Write an iterative version for triangle(N,T), posed as Exercise 8.2(i).

Exercise 8.2

(i) The Nth triangular number is the sum of the numbers up to and including N. Write a program for the relation triangle (N ,T), where '1' is the Nth triangular number. (Hint: Adapt Program 8.2.)

%factorial(N,F): F is the integer N factorial.

factorial(N,F) :-

N > 0, N1 is N-1, factorial(N1,F1), F is N*F1.

factorial(0,1).

Program 8.2 Computing the factorial of a number

Ans.

%i. Nth_triangular(N,T): T = N*(N-1)/2

% triangle(N,T): T is sum of first N natural numbers.

triangle(N,T) :- triangle(0,N,0,T).

triangle(I,N,A,T) :-

I < N, I1 is I+1,

A1 is A+I1, triangle(I1,N,A1,T).

triangle(N,N,T,T).

iii. Rewrite Program 8.5 so that the successive integers are generated in descending order.

% between(I,J,K): K is an integer between the integers I and J inclusive.

between(I,J,I) :- I <= J.

between(I,J,K) :- I < J, I1 is I+1, between(I1,J,K).

Program 8.5 Generating a range of integers

Ans.

% between(I,J,K): K is an integer between the

% integers I and J inclusive.

between(I,J,J) :- I <= J.

between(I,J,K) :- I < J, J1 is J-1, between(I,J1,K).

vi. Write a program to find the minimum of a list of integers.

Ans.

% min(List,least): least integer from List

%min([M|_],M) :- min([M|_],M).

min([M],M).

min([H|T],H) :-

min(T,M),

H <= M.

min([H|T],M) :-

min(T,M),

M < H.

```
?- ['a02q04.pl'].
true.
```

```
?- triangle(3,T).
T = 6 ;
false.
```

```
?- triangle(5,T).
T = 15 ;
false.
```

```
?- 
```

```
?- between(1,7,X).
X = 7 ;
X = 6 ;
X = 5 ;
X = 4 ;
X = 3 ;
X = 2 ;
X = 1 ;
false.
```

```
?- 
```

```
?- ['a02q04.pl'].
true.
```

```
?- min([1,2,3,0,12],M).
M = 0 ;
false.
```

```
?- min([21,42,-56,23],M).
M = -56 ;
false.
```

```
?- 
```

vii. Rewrite Program 8.11 for finding the length of a list so that it is iterative.
(Hint: Use a counter, as in Program 8.3.)

```
% length(Xs,N): N is the length of the list Xs.
```

```
length([X|Xe],N) :- length(Xs,N1), N is N1+1.  
length([],0).
```

Program 8.11 Finding the length of a list

```
% factorial(N,F): F is the integer N factorial.
```

```
factorial(N,F) :- factorial(0,N,1,F).
```

```
factorial(I,N,T,F) :-  
    I < N, I1 is I+1, T1 is T*I1, factorial(I1,N,T1,F).  
factorial(N,N,F,F).
```

Program 8.3 An iterative factorial

Ans.

```
% length_itr(List,A,L): L is length of List,  
%     A being accumulator
```

```
lengthL([],0).  
lengthL([H|T],L) :- length_itr([H|T],0,L).  
length_itr([H|T],A,L) :-  
    A1 is A+1,  
    length_itr(T,A1,L).  
length_itr([],A,A).
```

NOTE:

1. All the programs are in files **a02q0X.pl**, where X is Question No s.t.
X={1,2,3,4}.

2. Images are also attached as **asn2XY.png**, where X is Question No and Y is Sub-
Question No. s.t. X={1,2,3,4} and Y={0,1,2,3,4,...}.

```
?- ['a02q04.pl'].  
true.  
  
?- lengthL([1,2,3,0,12],L).  
L = 5.  
  
?- lengthL([a,b,c,d,e,f],L).  
L = 6.  
  
?- 
```