Main objects:
Irreducible Coxeter groups (A,B,D,E,F,G,H) implemented as permutation group with minimal permutation degree.

Coxeter Group Type	Perm. Rep.	Perm. degree
A_n	Sym(n)	n
B_n	2^n : Sym(n)	2n
D_n	2^{n-1} : Sym(n)	2n
E_6, E_7, E_8		27, 30, 240
F_4		24
G_2	Dih(12)	6
H_3, H_4		7, 120

The permutation degrees above are proven to be minimal in (Table 1):
N. Saunders, Minimal faithful permutation degrees for irreducible Coxeter groups and binary polyhedral groups, *J. Group Theory* **17** (2014), 805 -- 832

There is also a function that constructs the Coxeter groups abstract. One can test that my permutation implementations are corrected by using the built-in IsomorphismGroups function.

Methods & Functions:
Let (W,S) be a Coxeter system and T be the set of reflections.

Reduced factorisations:

1. Compute the set Red_{T,n}(w) = { [t_1, t_2, … t_n] | w = t_1 t_2 … t_n } where T the set of reflections and w in W is a quasi-coxeter element (namely, W is generated by t_1, t_2, …, t_n.
2. My algorithm works for any set of involutions T of W and w can be an arbitrary element.
3. The code is implemented as a GAP kernel function and still needs some work.

Braid Action & Hurwitz Orbit :
1. The function that computes the braid action was adapted from code by Tobias Jakobi (I recall fixing a couple of errors)
2. The braid action itself acts on tuples of reflections from T (it works for tuples of involutions in general)
3. Hurwitz orbit is just another name for orbit of tuple under the appropriate braid action.

Both *reduced factorisations* and *Hurwitz orbit* were developed to test the conjecture that w is quasi-Coxeter if and only if the braid action acts transitively on Red_{T,n}(w) (where n = rank(W)). The *Hurwitz orbit* still has many uses.

Quasi-Coxeter classes:
This computes the conjugacy classes in W of quasi-Coxeter elements (including the Coxeter elements). It depends on the result that the Hurwitz action is transitive on Red_{T,n}(w) for a quasi-Coxeter element w of rank n.

1. Finds a random quasi-Coxeter element and then discards all of its W-conjugates.
2. Repeats this process until all classes are found (the number of classes for each type of Coxeter group is known)
3. The function returns pairs (w, wt) where w is a representative element of the class and wt is the tuples of reflections whose product is w.
4. The order of the pairs is random (This could possibly be fixed?)

Type	A_n	B_n	D_n	E_6	E_7	E_8	F_4	G_2
# Classes	1	1	floor(n/2)	3	5	9	2	1

This function is pretty much in final draft form although it requires a lot of parameters.

Non-crossing partitions:
For a quasi-Coxeter element w, the non-crossing partition NC(w) is the poset whose elements are the products of prefixes of all the elements in Red_{T,n}(w).

1. For each k in [1,n] and each x=[t_1, …, t_n] in Red_T(w) we compute both [t_1, .. t_k] and its product t_1 .. .t_k. The product is the actual element of NC(w) but it's useful to remember the tuple(s) that produced it.
2. Level k, L(k) consists of the pairs (u, Ru) where u is a (parabolic quasi-Coxeter) element of W and Ru is the set of k-prefixes of Red_T(w) whose product is w.
3. Then NC(w) is the union of L(k) for k=0,1, … , n. (Here L(0) = [[id, [emptyset]]).

Total ordering, O = [t_1, t_2, .., t_p], of the reflections T is said to be compatible with w if:
For each k in [2,n]:
    For each (u,Ru) in L(k) there exists a unique x in Ru such that
    x[1] < ... < x[k] respect to O.

There is a conjecture that says it's enough to show it holds for k=2. It has been verified for D_4. The code only realistically works for D_4 because we brute force through all possible total orderings for the reflections. In the D_4 case there are 12! cases, in D_5 there are 20! cases. The D_4 case took several hours and there is no hope for the D_5 case. A better approach is needed.

Calculating bowties has also been implemented but it's quite ad-hoc and needs to be corrected.