Homework 1
22C:196 Computational Combinatorics
Due on September 18, 2001

Part I

For the following 3 problems, you do *not* have to write code.

1. Describe an algorithm to compute the rank of an n-permutation in Johnson-Trotter order (assuming that the Johnson-Trotter algorithm starts with I_n).
(**Hint:** For an n-permutation p express $Rank(p)$ in terms of $Rank(q)$, where q is an $(n-1)$-permutation obtained from p. Use this to devise a recursive algorithm.)

2. An *order-k inversion vector* is an inversion vector whose entries sum up to k. Describe an algorithm that takes positive integers n and k and generates all order-k inversion vectors of length $n-1$.
(**Hint:** The first element of an $n-1$-inversion vector can have any integer value between 1 and $\min\{n-1, k\}$. For each value that the first element takes, generate all possible inversion vectors of length $n-2$ and of the appropriate order.)

3. Let L_n be denote the number of length n involutions, that is, n-permutations that have cycles of length at most two. Prove the recurrence

$$L_n = L_{n-1} + (n-1)L_{n-2}$$

for any integer $n > 1$, letting $L_1 = L_0 = 1$. Use a technique similar to the one used in the proof of the recurrence for the Stirling numbers of the first kind.

Part II

For the following 3 problems you have to write *Mathematica* code or perform experiments with *Combinatorica* functions. Submit a *Mathematica* notebook containing solutions to these three problems.

1. Using the solution to Problem 1 in Part I, implement a function called `RankJTPermutation` in *Mathematica* to compute the rank in Johnson-Trotter order of a given n-permutation.

2. Using the solution in Problem 2 in Part I, implement a function called `KInversionVectors` in *Mathematica* that takes as inputs n and k and generates all order-k inversion vectors of length $n-1$.

   It is fairly easy to show that no two permutations have the same inversion vector. This is usually done by constructing an algorithm that takes as input an inversion vector v of length $(n-1)$ and returns an n-permutation p such that v is an inversion vector of p. A *Combinatorica* function called `FromInversionVector` provides an implementation of this algorithm.

   Implement a function called `KInversionPermutations` in *Mathematica* that takes as input positive integers n and k and generates all n-permutations that have k inversions.

3. The *Combinatorica* function `MakeGraph[v, f]` constructs the graph whose vertices correspond to v and whose edges are between pairs of vertices for which the binary relation defined by the boolean function f is true. This function is useful in defining graphs for combinatorial objects. For example, consider the graph $P_n = (V_n, E_n)$ defined in class as having vertex set V_n equal to the the set of all n-permutations and edge set E_n containing edges between pairs of permutations that can be obtained from each other by a swap.

   The code below defines a *Mathematica* function called `SwapGraph` that takes a positive integer n and returns the graph P_n.

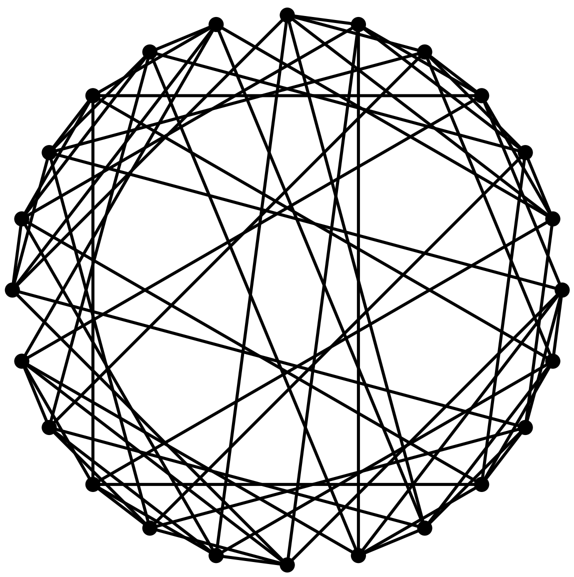
```
SwapGraph[n_Integer?Positive] :=
       MakeGraph[Permutations[n],
                 Length[
                    Select[
                       ToCycles[
                          Permute[InversePermutation[#1], #2]
                       ],
                       Length[#] == 1&
                    ]
                 ] == n - 2 &,
                 Type -> Undirected
       ]
```

The function simply calls, `MakeGraph` with the appropriate arguments. The first argument is `Permutations[n]` because this is the set of vertices of the graph. The second argument is a boolean function that takes two permutations as arguments referred to as `#1` and `#2` above. For any n-permutations p, q, and r, if $p \times q = r$, then $q = p^{-1} \times r$. Specifically, if p and r can be obtained from each other via a single swap, then q is a swap, that is, an n-permutation with $n-2$ 1-cycles and one 2-cycle. Thus the boolean function computes $p^{-1} \times r$, converts the result into its cycle structure, selects cycles of length 1, and then checks to see if these are $n-2$ in number.

For example, typing

`ShowGraph[SwapGraph[4]]`

produces P_4. As expected this graph has 24-vertex 6-regular graph.



In this problem, I want you to write a function `TwoSwapGraph` to generate a graph $P_{n,k} = (V_{n,k}, E_{n,k})$ whose vertex set $V_{n,k}$ equals the set of all n-permutations with exactly k cycles and whose edge set contains edges connecting permutations that can be obtained from each other by exactly two distinct swaps. `TwoSwapGraph` depends on being able to generate the set of all n-permutations with k cycles. Write a function called `KCyclePermutations` to do this.

Then use the *Combinatorica* function `HamiltonianCycle` to determine if $P_{6,3}$ has a Hamiltonian cycle. Try to answer this question for $P_{n,3}$ for n larger than 6? How much higher can you go before the graph becomes too large for Hamiltonian cycle.