16.5

UNIVERSITY OF CAMBRIDGE

CATAM

PART II

Permutation Groups

Question 1

Methods for computing products and inverses of 2-permutations have been provided at the end of this document.

Our inverse algorithm retrieves n entries of a permutation π and assigns these n values to different indices of π^{-1}, so it performs $2n$ steps and thus is $\mathcal{O}(n)$.

Example Code:

```
>>  invert([1,3,5,2,4])
    ans =

      1
      4
      2
      5
      3

>> comp([1;4;2;5;3],[1;3;5;2;4])

    ans =

      1
      2
      3
      4
      5
```

Question 2

Suppose $A = \{\pi_1, \pi_2, ..., \pi_k\}$ is our generating set. Note that replacing π_i by $\pi_j^{-1}\pi_i$ $(i \neq j)$ yields the same generating set since $\pi_i = \pi_j(\pi_j^{-1}\pi_i)$.

Now, if $B = \{\beta_1, ..., \beta_l\}$ $(l \leq k)$ is the generating set generated by the stripping algorithm, then all β_i will be of the form $\beta_i = g^{-1}\pi_j$ for some $g \in \langle B \rangle$. By relabelling we may take $j = i$. Now for $i \in \{l+1, ..., k\}$ we have $g^{-1}\pi_i = id$ for some $g \in \langle B \rangle$. Therefore,

$$\begin{aligned}
\langle B \rangle &= \langle \beta_1, ..., \beta_l \rangle \\
&= \langle \pi_1, ..., \pi_l \rangle \\
&= \langle \pi_1, ..., \pi_l, ..., \pi_k \rangle \\
&= \langle A \rangle.
\end{aligned}$$

All elements reaching row n will be disregarded as they must fix $1, \ldots, n-1$ pointwise and hence must fix n too. Therefore, the new generating set B satisfies $|B| \leq n(n-1)$, for $(n > 1)$.

In terms of complexity, each of the k original generators will undergo $\leq n$ checks, with each check witnessing inversion and composition ($\leq 2n + 2n$ operations). Therefore, the number of operations has a bound

$$|\text{ops}| \leq 4n^2k.$$

Question 3

My Sims Array code is included at the end of this document. An example output is included below.

Example Code:

```
% This script firs builds all elements of D5
% Then we feed it to Sims Algorithm to strip this set down

r = [5;1;2;3;4];     % r is a rotation of 2pi/5
q = [1;2;3;4;5];     % q = id
B = zeros(5,10);     % initialising all elems of D5

for i = 1:5
    q = comp(q,r);  % generating successive rotations
    B(:,i) = q;     % storing successive rotations
end

q = [1;5;4;3;2];     % now an arb reflection takes role of id above as we
                     % generate all reflections

for i = 1:5
   q = comp(q,r);   % generating successive reflections
    B(:,i+5) = q;   % storing successive reflections
end

[A,C] = Sims_Array(B);  % calculating sims array and stripped generators

>> C    % Our set of stripped generators

C =

     2     3     4     5     1
     3     4     5     1     5
     4     5     1     2     4
     5     1     2     3     3
     1     2     3     4     2

>> A    % Our sims array, the permutations are stored by looking deeper
% into this 3D array

A(:,:,1) =

   NaN     2     3     4     5
   NaN   NaN   NaN   NaN     1
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN


A(:,:,2) =
```

```
   NaN     3     4     5     1
   NaN   NaN   NaN   NaN     5
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN


A(:,:,3) =

   NaN     4     5     1     2
   NaN   NaN   NaN   NaN     4
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN


A(:,:,4) =

   NaN     5     1     2     3
   NaN   NaN   NaN   NaN     3
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN


A(:,:,5) =

   NaN     1     2     3     4
   NaN   NaN   NaN   NaN     2
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
```

Question 4

Define the function ϕ by

$$\begin{aligned} \phi : G/G_\alpha &\to Orb(\alpha), \\ gG_\alpha &\to g * \alpha. \end{aligned} \tag{1}$$

$\boxed{\phi \text{ well defined}}$

$$\begin{aligned} gG_\alpha = hG_\alpha &\iff gh^{-1} \in G_\alpha \\ &\iff gh^{-1} * \alpha = \alpha \\ &\iff g * \alpha = h * \alpha \end{aligned}$$

$\boxed{\phi \text{ injective}}$

$$\begin{aligned}\phi(gG_\alpha) = \phi(hG_\alpha) &\iff g * \alpha = h * \alpha \\ &\iff gh^{-1} * \alpha = \alpha \\ &\iff gh^{-1} \in G_\alpha \\ &\iff gG_\alpha = hG_\alpha\end{aligned}$$

$\boxed{\phi \text{ surjective}}$

Let $\beta = g * \alpha$, then $\phi(gG_\alpha) = \beta$. So we have

$$|G \cdot \alpha| = |G/G_\alpha| = |G|/|G_\alpha|,$$

which is the orbit-stabiliser theorem!

Question 5

The code for this procedure is included at the end of this document.

The data is stored in an $(m \times 2)$ array D where $m = |G \cdot \alpha|$ and

$$D(i,j) = \begin{cases} i^{th} \text{ element of orbit} & j = 1 \\ i^{th} \text{ witness} & j = 2 \end{cases}$$

We start with $D = (\alpha \quad id)$ and a generating set B. We then carry out the following procedure:

1. Apply each element g of B to α. If this generates a new element of the orbit, say β, append the row $(\beta \quad g)$ to D.
2. Now apply the same procedure recursively to all newly generated rows. So if h applied to β generates a new element γ, then append the row $(\gamma \quad h \cdot g)$ to D.
3. This will generate all elements of the orbit since if $g \in G$, then $g = g_1, ..., g_l$, $g_i \in B$ and at stage l of our algorithm we compute $(g_1 g_2 \cdots g_l) * \alpha$.
4. Terminate when we have checked that all generators g applied to all orbit elements α do not give anything new.

Example Code:

```
% We will calculate the orbit of 1 in the group D5
% This generating set C is defined by a script of ours called D5_test

C =

     2     3     4     5     1
     3     4     5     1     5
     4     5     1     2     4
     5     1     2     3     3
     1     2     3     4     2

>> D = orb(1,C)      % This calculates orbit of 1 and stores each element alongside its witness

D =
```

```
  5×2 cell array

    {[1]}    {5×1 double}       % col 1 = orb elem, col 2 = witness
    {[2]}    {5×1 double}
    {[3]}    {5×1 double}
    {[4]}    {5×1 double}
    {[5]}    {5×1 double}

>> D{1,:}   %If we want to look at row 1 of D then

ans =

     1      % orbit element


ans =
    % Witness
     1
     2
     3
     4
     5

>> D{3,:}

ans =

     3


ans =

     3
     4
     5
     1
     2
```

Question 6

First, define $\alpha_i = t_i * \alpha$, then

$$\begin{aligned}\alpha_{r+1} &= t_{r+1} * \alpha \\ &= \phi(y_r t_r) * \alpha \\ &= (y_r t_r) * \alpha \\ &= y_r * \alpha_r.\end{aligned}$$

Therefore,

$$\begin{aligned}\alpha_{r+1} &= (y_r y_{r-1}\cdots y_1) * \alpha_1 \\ &= x * \alpha \\ &= \alpha.\end{aligned}$$

So $t_{r+1} * \alpha = t_1 * \alpha$, and since T is a compete set of representatives, $t_{r+1} = t$. Next, let $x \in G_\alpha$, and then

$$\begin{aligned}x &= y_r y_{r-1}\cdots y_1 \\ &= t_{r+1}t_{r+1}^{-1}y_r t_r t_r^{-1} y_{r-1} t_{r-1} t_{r-1}^{-1} y_{r-2} \cdots t_2 t_2^{-1} y_1 t_1 t_1^{-1} \\ &= t_{r+1}\phi(y_r t_r)^{-1} y_r t_r \phi(y_r t_r)^{-1} y_r t_r \cdots \phi(y_1 t_1)^{-1} y_1 t_1 t_1^{-1} \\ &= t_1 r t_1^{-1},\end{aligned}$$

so $t_1^{-1} x t_1 = r \in \langle \phi(yt)^{-1} yt | y \in Y, t \in T\rangle$. Since conjugation is bijective from G_α to G_α, this shows that

$$G_\alpha \leq \langle \phi(yt)^{-1} yt | y \in Y, t \in T\rangle.$$

Note also that

$$(\phi(yt)^{-1} yt) * \alpha = \phi(yt)^{-1} * (yt * \alpha) = \alpha$$

for all $y \in Y, t \in T$. Thus

$$G_\alpha = \langle \phi(yt)^{-1} yt | y \in Y, t \in T\rangle.$$

Question 7

The code for this section is included in the appendix.

This procedure takes as an input some $\alpha \in \{1, \ldots, n\}$ and some generating set Y. It then does the following:

1. Compute a complete set of repetitions for the set of cosets G/G_α by computing witnesses of orbit elements.
2. Compute the set
$$\{\phi(yt)^{-1} yt | y \in Y, t \in T\}. \tag{2}$$
3. Apply the stripping algorithm to (2). Let $k = |Y|$, then the complexity of each stage is
   - Let $m = |G \cdot \alpha|$, then we perform m compositions and m storages, and for each composition we perform $\leq k$ checks of the value of $y(\beta)$. So the complexity is $\mathcal{O}(nmk) = \mathcal{O}(n^2 k)$.
   - This stage performs km assignments, each involving compositiona nd inversion, which are $\mathcal{O}(n)$, so complexity is $\mathcal{O}(n^2 k)$.
   - We saw in question 1 that this was $\mathcal{O}(n^2 k)$.

   So our procedure has complexity $\mathcal{O}(n^2 k)$.

   Example Code:

```
%This script gives a set of generators for Q8 and allows us to use some of
%our functions on it. Note that a and b defined below satisfy a^4 = e,
%a^2 = b^2, aba = b
a = [2;4;6;7;3;8;1;5];
b = [3;5;4;8;7;2;6;1];
c = comp(a,b);  %This isn't required to generate the group, but let's see
% if it gets removed!

[A,C] = Sims_Array([a,b]);  % calculating sims array and stripped generators

>> C    % Our stripped set of generators

C =     % We see comp(a,b) has been stripped

     2     3
     4     5
     6     4
     7     8
     3     7
     8     2
     1     6
     5     1

>> stab(1,C)

ans =

     []     % Since stabiliser in Q8 is trivial, it gets returned as empty set
```

Question 8

The code from this section is included in the appendix.
Our procedure computes the following stabilisers:

$$
\begin{aligned}
H_0 &= G, \\
H_1 &= G_1 \\
H_2 &= G_1 \cap G_2 \\
&\vdots \\
H_k &= G_1 \cap \cdots \cap G_k = \{e\},
\end{aligned}
$$

where k exists as $n < \infty$. Then define:

$$
\begin{aligned}
\theta_1 &= |H_0 \cdot 1| = \frac{|H_0|}{|H_1|} \\
\theta_2 &= |H_1 \cdot 2| = \frac{|H_1|}{|H_2|} \\
&\vdots \\
\theta_k &= |H_{k-1} \cdot k| = \frac{|H_{k-1}|}{|H_k|}.
\end{aligned}
$$

So $\theta_1 \cdots \theta_k = \frac{|H_0|}{|H_k|} = \frac{|G|}{1} = |G|$.

If we didn't apply the stripping algorithm at each stage, the number of generators would be allowed to get very large. If $G = \langle g, h \rangle \leq S_{20}$, then $G_1 \leq S_{19}$, so $|G_1| \leq 19! \cong 1.2 \times 10^{17}$. Therefore, if G is a large group, we could have many generators at each stage, which would make our code very slow. Some examples are given below.

Example Code:

```
>> Q8_Test % This gives us a gen. set C of Q8
>> C

C =

     2     3
     4     5
     6     4
     7     8
     3     7
     8     2
     1     6
     5     1

>> order(C)

ans =

     8      %Great! Just as expected

>> D5_Test  % This gives us a gen. set C of D5
>> C

C =

     2     3     4     5     1
     3     4     5     1     5
     4     5     1     2     4
     5     1     2     3     3
     1     2     3     4     2

>> order(C)

ans =

    10     2    % Perfect! Not only do we store the order (leftmost entry)
                % but also the order of the subgroup calculated along
                % the way!

>>
```

Question 9

We know that $P_n > 0$ because

$$\langle (1 \quad 2), (1 \quad 2 \quad \cdots \quad n) \rangle = S_n.$$

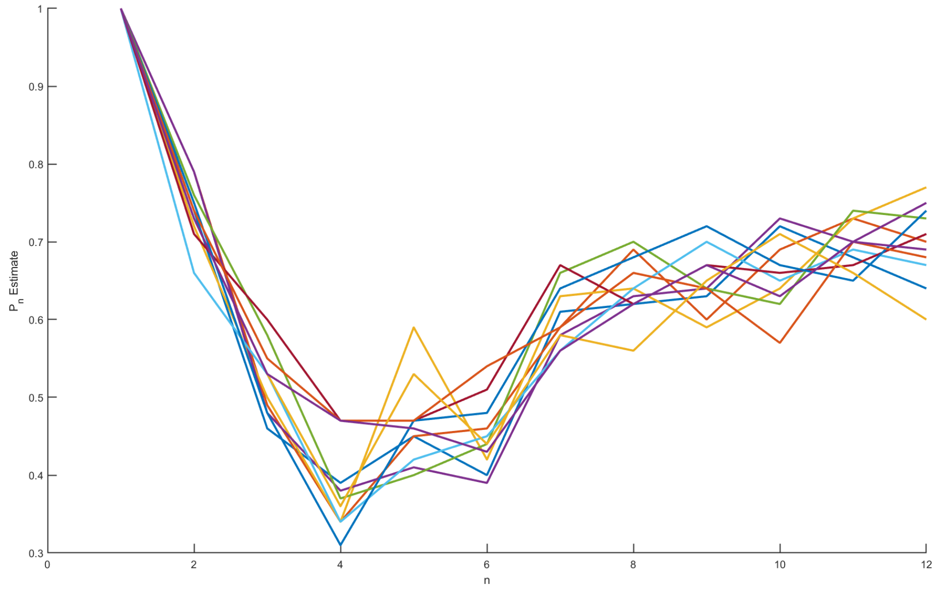

Figure 1: Estimates for the probability P_n

However, we may also bound P_n above by noting that if our random permutations g and h are both over, then $\langle g, h \rangle \leq A_n$, so

$$P_n \leq \mathbb{P}(g, h \text{ not both even}) = 3/4.$$

For small n :

n	P_n	Comments
1	1	Trivial
2	3/4	Fails iff $g = h = c$
3	1/2	Works iff g, h both 2-cycles (6 cases) or one 2, one 3 (12 cases)

To generate a random permutation, my code does the following:

- Start with $U = \{1, \ldots, n\}$
- Choose a random $i \in U$, then set $v(1) = i$, and $U = U \backslash \{i\}$
- Do this n times, each time incrementing the index of v up by one.

Trying 100 random pairs from S_n, for $n \in \{1, \ldots, 11\}$ 10 times yields the following plot:

We see a general dip towards $n = 4$, but for larger values of n, P_n appears to be tending to 0.75.

Appendix

```
function [s] = comp(p,q)
%This function composes the permutations p and q to return p*q
%The permutations p,q of {1,..,n} are represented by n*1 vectors, with p(i)
% = the value of i under the permutation p
```

```
n = length(p);
s = zeros(n,1);    %initialising the output permutation

for i = 1:n
    s(i) = p(q(i));
end
end



function [s] = invert(p)
% This function inverts the permutation p
% The permutation p of {1,..,n} is represented by an n*1 vector
% Works by setting the p(i)^th index of s as i and then returning s

n = length(p);
s = zeros(n,1);    %initialising the output permutation

for i = 1:n
    s(p(i)) = i;
end

end




function [A,C] = Sims_Array(B)
% B is an n*k matrix whose columns are perms generating some G < S_n
% C is an n*r matrix (r<=k) which reduces the generating set to a smaller
% generating set by removing redundancy

if isempty(B) == 0
    n = length(B(:,1)); %num of elems in column of B = n
    k = length(B(1,:)); %num of elements in original generating set = num rows
    A = NaN(n,n,n); %A is our Sims array

    for i = 1:k     %we now loop through col 1 of B and put them in the array A
        j = 1;  %j is just a counter
        while (j <= n)  %j counts which row we are in, only care about j <= n
            if(min(isnan(A(j,B(j,i),:))) == 1 && B(j,i) ~= j)
                %This is a slick way of checking if entry A(j,B(j,i),:) is
                %empty. isnan returns an array of 0s and 1s and if all 1s it is
                %empty. Also checking B(:,i) doesn't fix j
                A(j,B(j,i),:) = B(:,i); %Assignment
                break
                %Done job of assigning B(:,i) so break out of while loop and
                %move on to next element
            elseif (max(isnan(A(j,B(j,i),:))) == 0)
                %This condition checks if there is a vector in A(j,B(j,i),:)
                g = squeeze(A(j,B(j,i),:));
                %This just removes dimensions of length 1, sets this perm = g
                B(:,i) = comp(invert(g),squeeze(B(:,i)));
```

```
            end
            j = j+1;
        end
    end

    %Next we want to retrieve all nonempty vectors in A and place these in the
    %columns of C

    C=[];   %Initialising C, this ensures that if B = [id], then C = [empty]
    j = 1;  %Reinitialising counter

    %loop through each element in the array and see if it contains a perm
    for i = 1:n
        for h = 1:n
            if(max(isnan(A(i,h,:))) == 0)
                %we need all entries of isnan(A) == 0 so that we can say
                %there's a vector inside A(i,j,:)
                C(:,j) = squeeze(A(i,h,:));
                j = j+1;
            end
        end
    end
else
    A = [];
    C = [];
end
end
```

```
% Generates all of D5 then strips
r = [5;1;2;3;4];    % r is a rotation of 2pi/5
q = [1;2;3;4;5];    % q = id
B = zeros(5,10);    % initialising all elems of D5

for i = 1:5
    q = comp(q,r);  % generating successive rotations
    B(:,i) = q;     % storing successive rotations
end

q = [1;5;4;3;2];    % now an arb reflection takes role of id above as we
                    % generate all reflections

for i = 1:5
   q = comp(q,r);   % generating successive reflections
    B(:,i+5) = q;   % storing successive reflections
end

[A,C] = Sims_Array(B);  % calculating sims array and stripped generators
```

```
%This script gives a set of generators for Q8 and allows us to use some of
```

```
%our functions on it. Note that a and b defined below satisfy a^4 = e,
%a^2 = b^2, aba = b
a = [2;4;6;7;3;8;1;5];
b = [3;5;4;8;7;2;6;1];
c = comp(a,b);  %This isn't required to generate the group, but let's see
% if it gets removed!

[A,C] = Sims_Array([a,b]);  % calculating sims array and stripped generators
```

```
function [D] = orb(a,B)
% Q5 is a nice opportunity to work with cell arrays!
% Cell arrays are good when we want to store information of different data
% types in an array
% D is a cell array with col 1 giving elems of orbits and corresponding row
% in col 2 giving witnesses
% B = set of generators of G
% a = element of {1,2,...,n} whose orbit we wish to compute

    n = length(B(:,1));      % n is the size of the set we are permuting
    r = length(B(1,:));      % r is the number of generators
    D = {a,(1:n)'};          % The identity clearly sends a to a
    [k,~] = size(D);         % k = size of orbit computed thus far
    i = 1;                   % initialising counter

    while i <= k
        % This condition ensures that the while loop terminates when we
        % go through our currently computed orbit, apply our generating set
        % to it and this doesn't generate any new orbit elements
        for j = 1:r     % looping over all cols in B
            if ismember(B(D{i,1},j),[D{:,1}]) == 0
                %This if statement takes an element in the orbit, applies a
                %generator to it and sees if the resulting element is
                %already in the orbit. If not, we add it and its witness,
                %then we repeat the process on this element. We continue
                %until this method doesn't generate anything new
                D{k+1,1} = B(D{i,1},j);     %add new orb elem
                D{k+1,2} = comp(B(:,j),D{i,2});     %add new witness
                [k,~] = size(D);    %increment k
            end
        end
        i = i+1;
    end

end
```

```
function [C] = stab(a,B)
% a = elem of {1,2,..,n} whose stabiliser we will compute
% B = n*r matrix of generators
```

```
% C = n*s matrix of generators for stabilizer of a

n = length(B(:,1));      % n = sizze of set being permuted
r = length(B(1,:));      % r = num of generators for G
T = orb(a,B);            % T = orbit and witnesses
m = length([T{:,1}]);    % m = size of orbit
C = zeros(n,1);          % initialising C
k = 1;                   % k = counter for num of cols of C

% No complicated strategy here, we just store all possible elements of the
% set given in the question and then strip it using our Sims algorithm

for i = 1:r     % looping through generators of G
    for j = 1:m % looping through elems of T
        u = invert(phi(comp(B(:,i),T{j,2}),a,T));
        v = comp(B(:,i),T{j,2});
        C(:,k) = comp(u,v);
        k = k+1;
    end
end

[~,C] = Sims_Array(C);

end



function [h] = phi(g,a,D)
% g = elem. of G (n*1)
% a = elem. of {1,2,..,n} (1*1)
% D = orb. of a with witnesses (1*2 cell)
b = g(a);   % b is the value of g(a), b in {1,2,..,n}
k = find([D{:,1}]==b);  % k = row index of b in orbit
h = D{k,2};
end




function [m] = order(B)
% B is an n*r matrix whose cols are generators of some group G
% m is a vector whose entries are all orders of the subgroups we compute
% in descending order

    u = []; % u is a vector whose kth entry is the size of the orbit of k
    % in the subgroup G_1 intersect G_2 intersect ... intersect G_k-1
    v = []; % v is a matrix w/ 2 cols and as many rows as entries in u, it
    % stores in col 1 the size of gen set for subgrp before stripping and
    % in col 2 after stripping
    k = 1;  % counter

    % strategy: find stabiliser of 1 in G, stabiliser of 2 in G_1,
    % stabiliser of 3 in G_1 and G_2, and so on. Eventually, we get the
```

```
    % trivial group, by calculating the sizes of these stabilisers and
    % using the orbit stabiliser theorem, and then multiplying these
    % together, we get the otder of G and the order of all these subgroups
    % in the process


    while isempty(B) == 0
        [u(k), ~] = size(orb(k,B)); % size of orbit
        B = stab(k,B); % set B = new set of generators for next subgrp
        v(k,1) = size(B,2);     % retrieve no. cols of B (ie num generators)
        [~, B] = Sims_Array(B); % strip B
        v(k,2) = size(B,2);     % retrieve no. cols of B (ie num generators)
        k = k+1;
    end

    t = length(u);  % t is the num of steps we have gone through
    m = zeros(1,t); % initialise m, vector of all subgroup sizes

    for i = 1:t
        m(i) = prod(u(i:t));    %m(1) = u_1*..u_t, m(2) = u_2*.._u_t, ...
    end

end
```

```
%This script creates a vector v st v(i) is the approx probability of 2
%random perms generating S_i. We do this 10 times and then plot all vectors
%against n

hold on
for j = 1:11
    v = zeros(1,12);    %initialise v

    for i = 1:12
        v(i) = prob_estimate(i,100);
    end

    plot(1:12,v);
end
```

```
function [p] = prob_estimate(n,m)
% n = num of elements we are permuting
% m = num of random pairs we are creating

B = zeros(n,2*m);   % B stores random perms in cols, we have m pairs, with
% pair i in cols 2i-1 and 2i
s = 0;  % s = num of pairs which generate all of S_n
```

```
for i = 1:2*m
    B(:,i) = rp(n);
end

for i = 1:m
    if order(B(:,[2*i-1,2*i])) == factorial(n)
        % this condition checks if the ith random pair generates all of
        % S_n. If the answer is yes, then increment s.
        s = s+1;
    end
end

p = s/m;    % p is the empirical probability estimate

end
```


```
function [p] = prob_estimate(n,m)
% n = num of elements we are permuting
% m = num of random pairs we are creating

B = zeros(n,2*m);   % B stores random perms in cols, we have m pairs, with
% pair i in cols 2i-1 and 2i
s = 0;  % s = num of pairs which generate all of S_n

for i = 1:2*m
    B(:,i) = rp(n);
end

for i = 1:m
    if order(B(:,[2*i-1,2*i])) == factorial(n)
        % this condition checks if the ith random pair generates all of
        % S_n. If the answer is yes, then increment s.
        s = s+1;
    end
end

p = s/m;    % p is the empirical probability estimate

end
```