16.5

Permutation Groups

Question 1

We will identify permutations with arrays in the following way: the array $(a_1, a_2, \ldots, a_n)$ represents the permutation $1 \to a_1$, $2 \to a_2$ and so on. Here are some outputs from the inverse and product programs.

$$(2, 3, 6, 5, 1, 4)^{-1} = (5, 1, 2, 6, 4, 3)$$

$$(2, 3, 6, 5, 1, 4) \cdot (4, 3, 5, 1, 2, 6) = (5, 6, 1, 2, 3, 4)$$

So both programmes work as expected. The complexity of the inverse procedure is $O(n^2)$: for each $k \in [n]$ it iterates through, on average, half the positions of the array, of which there are $n/2$.

We will henceforth denote the set of permutations $\{\pi_1, \ldots, \pi_k\}$ by the $n \times k$ array

$$\begin{pmatrix} \uparrow & & \uparrow \\ \pi_1 & \ldots & \pi_k \\ \downarrow & & \downarrow \end{pmatrix}$$

We will also make use of the standard notation throughout.

Question 2

For each $1 \leq i \leq n$ let π_i and π_i' denote the original and modified permutations respectively.

Modified set generates the group. If π_i' is in the array then we know that $\pi_i' = g^{-1}\pi_i$ for some g expressible as a product of π_j' permutations for $j < i$ in the array. So $\pi_i = g\pi_i'$ and we can generate all the π_i from permutations only in the array.

If π_i' is not in the array (it is the identity) then we know that $g^{-1}\pi_i = e$ for some g expressible as a product of π_j' permutations for $j < i$ in the array. So $\pi_i = g$ and we can generate all the π_i again.

Hence the modified set of permutations generates the same group.

Size of modified set. If a permutation is put in position (i, j) in the array then it must fix all $a < i$ and send i to j. Hence $j > i$ since the permutation is bijective (and hence injective) and it does not fix i. So we can only place permutations in the top triangle, giving at most $n(n-1)/2$ generators. The size of the modified set of generators is also at most the size of the original set. So it is at most $\min(k, n(n-1)/2)$.

Complexity. The loop iterating through the elements of the generating set contributes a factor of $O(k)$. The loop iterating through $[n]$ contributes a factor of $O(n)$. Inside the loop the storing of permutations in the array A contributes another factor of $O(n)$. So the overall complexity is $O(kn^2)$.

Question 3

We will give a few examples to show the program is working correctly.

Example 1. The following set contains all elements of the group S_3.

$$\begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 1 & 2 \\ 3 & 2 & 3 & 1 & 2 & 1 \end{pmatrix}$$

From the Stripping Algorithm of Sims we obtain the array A and hence the reduced set of permutations. In the procedure the array A from the question has been modified to a $n^2 \times n$ array so that the permutations may be written out in full.

$$\begin{pmatrix} 0 & 2 & 3 \\ 0 & 1 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 1 \\ 2 & 3 & 2 \end{pmatrix}$$

Example 2. The following set contains all the elements of the group A_4.

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 \\ 2 & 3 & 4 & 1 & 3 & 4 & 1 & 2 & 4 & 3 & 1 & 2 \\ 3 & 4 & 2 & 4 & 1 & 3 & 2 & 4 & 1 & 2 & 3 & 1 \\ 4 & 2 & 3 & 3 & 4 & 1 & 4 & 1 & 2 & 1 & 2 & 3 \end{pmatrix}$$

Using the program we have

$$\begin{pmatrix} 0 & 2 & 3 & 4 \\ 0 & 1 & 1 & 3 \\ 0 & 4 & 2 & 2 \\ 0 & 3 & 4 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 1 & 3 \\ 4 & 2 & 4 & 2 & 2 \\ 2 & 3 & 3 & 4 & 1 \end{pmatrix}$$

Question 4

There is a bijection between the set of left cosets of G_α in G and the orbit of α, given by $\Psi : G/G_\alpha \to G \cdot \alpha$, $gG_\alpha \mapsto g\alpha$. Intuitively this is because the difference of two elements in the same coset is in the stabilizer and so they have the same effect on α.

The orbit-stabilizer theorem states that for a finite group G and an element $\alpha \in X$,

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

Question 5

Description. First apply every permutation in the generating set to α and add every new orbit element to the list. Continue by repeating this with every element in the list. Stop once no new elements are found. Keeping track of the permutations used allows each witness to be computed.

Explanation. For every element $\beta \in X$ in the orbit of α there is some $g \in G$ with $g\alpha = \beta$. If g is in the generating set then β will be added after the first iteration. Otherwise g can be written as a product of m elements from the generating set. By induction on m at some point β will be added to the list, since both the group G and the set X are finite.

We will give a few examples to show the program works as required.

Example 1. Take the subgroup of S_5 generated by the permutations (123) and (45).

$$\begin{pmatrix} 2 & 1 \\ 3 & 2 \\ 1 & 3 \\ 4 & 5 \\ 5 & 4 \end{pmatrix}$$

The elements of the orbits are displayed in the top row of the matrices in bold with a witness below each one. The results are shown for the elements 2 and 4.

$$\begin{pmatrix} \mathbf{2} & \mathbf{3} & \mathbf{1} \\ 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \end{pmatrix}, \begin{pmatrix} \mathbf{4} & \mathbf{5} \\ 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 5 \\ 5 & 4 \end{pmatrix}$$

Example 2. Take the subgroup of S_6 generated by the permutations (14), (36) and (24).

$$\begin{pmatrix} 4 & 1 & 1 \\ 2 & 2 & 4 \\ 3 & 6 & 3 \\ 1 & 4 & 2 \\ 5 & 5 & 5 \\ 6 & 3 & 6 \end{pmatrix}$$

The results are shown for the elements 2, 5 and 6.

$$\begin{pmatrix} \mathbf{2} & \mathbf{4} & \mathbf{1} \\ 1 & 1 & 4 \\ 2 & 4 & 1 \\ 3 & 3 & 3 \\ 4 & 2 & 2 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{pmatrix}, \begin{pmatrix} \mathbf{5} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}, \begin{pmatrix} \mathbf{6} & \mathbf{3} \\ 1 & 1 \\ 2 & 2 \\ 3 & 6 \\ 4 & 4 \\ 5 & 5 \\ 6 & 3 \end{pmatrix}$$

So indeed the program works as expected.

Question 6

We consider the action of t_{i+1} on α

$$t_{i+1}\alpha = \varphi(y_i t_i)\alpha = y_i t_i \alpha$$

Using the definition of φ.

$$t_{r+1}\alpha = y_r t_r \alpha = y_r \dots y_1 t_1 \alpha = x t_1 \alpha = x\alpha = \alpha$$

Since both x and t_1 are in G_α. So t_{r+1} is also in G_α. Since it is also in T so we must have $t_{r+1} = t_1$.

We now show that $G_\alpha = \langle \{\varphi(yt)^{-1} \cdot y \cdot t | y \in Y, t \in T\} \rangle$.

$$yt\alpha = \varphi(yt)\alpha \Rightarrow \varphi(yt)^{-1} yt\alpha = \alpha \Rightarrow \varphi(yt)^{-1}yt \in G_\alpha$$

$\forall y \in Y, t \in T$. So $\langle \{\varphi(yt)^{-1} \cdot y \cdot t | y \in Y, t \in T\} \rangle \subseteq G_\alpha$.

Now $\forall x \in G_\alpha$, write $x = y_r \dots y_1$ with each y_i an element of Y. Choose $t_1 = e$ and construct the other t_i recursively by $t_{i+1} = \varphi(y_i t_i)$.

$$\begin{aligned} x = y_r \dots y_1 &= (t_{r+1}^{-1} y_r t_r)(t_r^{-1} y_{r-1} t_{r-1}) \dots (t_2^{-1} y_1 t_1) \\ &= [\varphi(y_r t_r)^{-1} y_r t_r][\varphi(y_{r-1} t_{r-1})^{-1} y_{r-1} t_{r-1}] \dots [\varphi(y_1 t_1)^{-1} y_1 t_1] \end{aligned}$$

Using $t_{r+1} = t_1 = e$. This is a product of elements from the set. Hence $G_\alpha \subseteq \langle \{\varphi(yt)^{-1} \cdot y \cdot t | y \in Y, t \in T\} \rangle$ and so the two are in fact equal.

Question 7

This program iterates through all elements in both sets T and Y and uses the result from question 6 to find a generating set for the stabilizer. We will use the same examples used in Question 5 to show the program works as required.

Example 1. The procedure yields the following generating sets for the stabilizers of the elements 2 and 4.

$$\begin{pmatrix}1\\2\\3\\5\\4\end{pmatrix}, \begin{pmatrix}2\\3\\1\\4\\5\end{pmatrix}$$

Example 2. The procedure yields the following generating sets for the stabilizers of the elements 2, 5 and 6.

$$\begin{pmatrix}1&4\\2&2\\6&3\\4&1\\5&5\\3&6\end{pmatrix}, \begin{pmatrix}2&1&1\\1&2&4\\3&6&3\\4&4&2\\5&5&5\\6&3&6\end{pmatrix}, \begin{pmatrix}2&1\\1&4\\3&3\\4&2\\5&5\\6&6\end{pmatrix}$$

Complexity. In the worst case, the set of witnesses T contains n elements (when the action is transitive). There are at most k elements in the set of generators Y. Inside the loops, the inverse function dominates (since multiplication is linear in n) and therefore there is a contribution of $O(n^2)$. Hence the overall complexity of the algorithm is $O(kn^3)$.

To test this, the procedure was timed for a few different values of n (the value of k used was 5 and the permutations were randomised). The results are recorded below. The gradient of the plot is about 2.5 which suggests that the calculation above is indeed correct.

n	80	100	120	140	160	180	200	220	240	260
Time	4.581	7.489	10.394	14.483	20.245	25.948	35.4	42.793	52.017	71.029

Table 1: Running times of stabilizer procedure

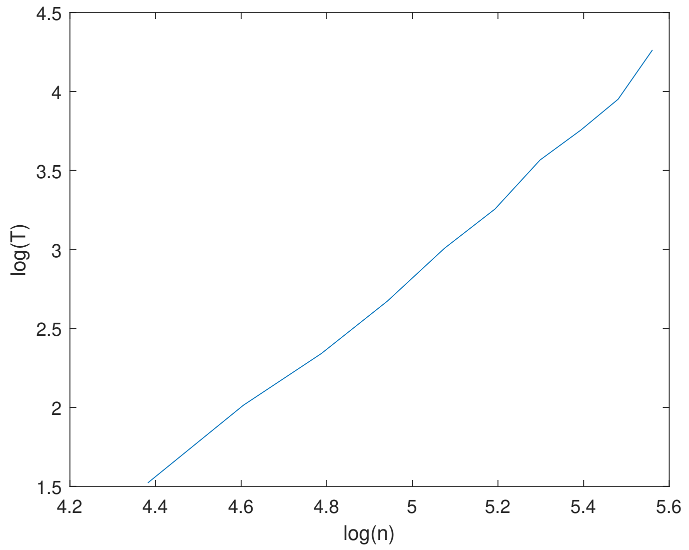

Figure 1: plot of log(T) against log(n)

Question 8

Procedure. We construct the following chain of subgroups:

$$G \geq G_1 \geq G_{(1,2)} \cdots \geq G_{(1,\ldots,n)} = e$$

From which we obtain

$$|G| = |\mathcal{O}_G(1)||G_1| = |\mathcal{O}_G(1)||\mathcal{O}_{G_1}(2)||G_{(1,2)}| = \cdots = |\mathcal{O}_G(1)| \ldots |\mathcal{O}_{G_{(1,\ldots,n-1)}}(n)|$$

Where $G_{(1,\ldots,k)}$ is the subgroup which fixes elements $1, \ldots, k$ and $\mathcal{O}$ is the Orbit. So by finding the subgroups iteratively and the sizes of the orbits, the order of the group can be determined.

Example. The following permutations generate S_5

$$\begin{pmatrix} 2 & 1 \\ 1 & 3 \\ 3 & 4 \\ 4 & 5 \\ 5 & 2 \end{pmatrix}$$

Subgroup G_1 of order 24, before and after stripping.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 3 & 5 & 4 & 2 \\ 4 & 3 & 3 & 5 \\ 5 & 4 & 2 & 3 \\ 2 & 2 & 5 & 4 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 3 & 5 & 4 & 2 \\ 4 & 3 & 3 & 5 \\ 5 & 4 & 2 & 3 \\ 2 & 2 & 5 & 4 \end{pmatrix}$$

Subgroup $G_{(1,2)}$ of order 6.

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 5 & 3 & 4 \\ 3 & 5 & 5 \\ 4 & 4 & 3 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 5 & 3 & 4 \\ 3 & 5 & 5 \\ 4 & 4 & 3 \end{pmatrix}$$

Subgroup $G_{(1,2,3)}$ of order 2.

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 4 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 4 \end{pmatrix}$$

The program outputs 120 for the order which is indeed correct.

Need for Stripping Algorithm. The procedure was executed with the permutations (12) and $(12 \cdots 20)$, which generate S_{20}. It was performed twice and timed, once using the Stripping Algorithm at every stage (this took 573s) and once without using it (this ran over 4000s and was halted before completion). So we see that not using the Stripping Algorithm significantly increases the time required.

Question 9

We will show that S_n is generated by the cycles (12) and $(12\cdots n)$. We know from 1A that for a permutation $\sigma \in S_n$ and a cycle $(a_1 \cdots a_m) \in S_n$

$$\sigma(a_1 \cdots a_m)\sigma^{-1} = (\sigma(a_1) \cdots \sigma(a_m))$$

So $(12\cdots n)(12)(12\cdots n)^{-1} = (23)$ and we can iterate to generate all the transpositions of the form $(j, j+1)$. We can then obtain all transpositions (i, j) (wlog $j > i$) by

$$(i,j) = (j, j-1)(j-1, j-2)\dots(i+2, i+1)(i, i+1)(i+1, i+2)\dots(j-1, j)$$

We can write every element in S_n as a product of transpositions and so we can generate every element. Therefore $P_n > 0$ since there is always at least one pair of permutations that generates S_n.

If g and h are both even permutations then all the permutations they generate are also even, and so they can not generate S_n. Half of all permutations are even ($|A_n| = |S_n|/2$) so there is a 1/4 probability that both g and h are even. So independently of n, $P_n \leq 3/4$.

Estimates for P_n have been calculated by generating 100 random pairs of permutations for each value of n. The values agree with the upper bound found above and are in fact quite close to it.

n	5	6	7	8	9	10	11	12	13	14
P_n	0.42	0.36	0.68	0.62	0.68	0.73	0.65	0.66	0.72	0.72

Table 2: Estimates of P_n

End of Project

Product Function

```
function M = Mul(X1,X2)
n = length(X1);
M = zeros(1,n);
for i = 1:n
    k = X2(i);
    M(i) = X1(k);
end
```

Inverse Function

```
function I = Inv(X)
n = length(X);
I = zeros(1,n);
for i = 1:n
    for j = 1:n
        if X(j) == i
            I(i) = j;
            break
        end
    end
end
```

Stripping Algorithm of Sims

```
function A = Sims(X)
n = size(X,1); %number of elements of X
k = size(X,2); %number of given generators
P = zeros(n^2,n); %Array of permutations
A = []; %Array of generators
for m = 1:k %go through the permutations
    p = X(:,m);
    for i = 1:n %go through each element
        if p(i) ~= i
            if P(n*(i-1)+1,p(i)) == 0 %if entry is empty
                P(n*(i-1)+1:i*n,p(i)) = p;
                A = [A,p]; %add to list of generators
                break
            else
                g = P(n*(i-1)+1:i*n,p(i));
                p = transpose(Mul(Inv(g),p)); %change value of p
            end
        end
    end
end
```

Orbit Algorithm

```
function  O  = Orb(x,S)
A = Sims(S);
n = size(A,1); %number of elements of X
O = [x;transpose(1:n)]; %top row orbit elements, under each a witness
m = size(O,2); %number of elements in orbit
k = size(A,2); %number of generators
i = 1;
while i <= m %to run through the elements in the orbit
    for j = 1:k %to run through the generators
        d = A(O(1,i),j); %element in orbit
        if ~ismember(d,O(1,:))
            p2 = A(:,j); %Perm to get from x to the intermediate
            p1 = O(2:end,i); %Perm to get from the intermediate to d
            p = transpose(Mul(p2,p1)); %Perm to get from x to d
            c = [d;p]; %create column of element and witness
            O = [O,c]; %concatenate to O
            m = m+1; %increment size of orbit
        end
    end
    i = i+1; %go onto the next element in orbit
end
```

Representative function φ

```
function F = Phi(a,p,A)
o = p(a); %image of a under the perm
B = Orb(a,A);
for j = 1:size(B,2)
    if B(1,j) == o
        F = B(2:end,j); %the witness of the orbit element
    end
end
```

Stabilizer Algorithm

```
function S = Stab(a,A)
O = Orb(a,A); %Orbit of a with the witnesses
Y = Sims(A); %Generating set for the group
T = O(2:end,:); %Set of witnesses/representatives
Sg = []; %Set of stabilisers
for i = 1:size(T,2) %iterating through witnesses
    for j = 1:size(Y,2) %iterating through generators
        t = T(:,i); %witness
        y = Y(:,j); %generator
```

```
            z = Inv(Phi(a,Mul(y,t),A));
            s = transpose(Mul(Mul(z,y),t)); %using schreier's theorem
            Sg = [Sg,s];%concatenate s to S
        end
end
S = Sims(Sg);
```

Order Algorithm

```
function O = Ord(A)
O = 1; %To start off the product
n = size(A,1);
G = A; %The initial group
for i = 1:n
    Gs = Sims(G);
%   DM(Gs)
    orb = Orb(i,Gs);
%   size(orb,2)
    O = O*size(orb,2);
    if isempty(Stab(i,Gs))
        break
    else
        G = Stab(i,Gs); %Look at orbit of element i+1 under group of stabiliser
%       DM(G)
    end
end
```

Questions 3,5,7,8

```
%Question 3
%The group S3
B1 = [1;2;3]; B2 = [1;3;2]; B3 = [2;1;3];
B4 = [2;3;1]; B5 = [3;1;2]; B6 = [3;2;1];
GB = [B1,B2,B3,B4,B5,B6];
%The group A4
A1 = [1;2;3;4]; A2 = [1;3;4;2]; A3 = [1;4;2;3]; A4 = [2;1;4;3];
A5 = [2;3;1;4]; A6 = [2;4;3;1]; A7 = [3;1;2;4]; A8 = [3;2;4;1];
A9 = [3;4;1;2]; A10 = [4;3;2;1]; A11 = [4;1;3;2]; A12 = [4;2;1;3];
GA = [A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12];

%Questions 5 and 7
%Example 1
C1 = [2;3;1;4;5]; C2 = [1;2;3;5;4];
GC = [C1,C2];
%Example 2
D1 = [4;2;3;1;5;6]; D2 = [1;2;6;4;5;3]; D3 = [1;4;3;2;5;6];
```

```
GD = [D1,D2,D3];

% Question 7 Complexity
n = 260;
E1 = randperm(n,n); E2 = randperm(n,n); E3 = randperm(n,n);
E4 = randperm(n,n); E5 = randperm(n,n);
GE = transpose([E1;E2;E3;E4;E5]);
% Time plot for Complexity
N = 80:20:260;
T = [4.581,7.489,10.394,14.483,20.245 ,25.948 ,35.400,42.793,52.017 ,71.029];
plot(log(N),log(T))
gradient(log(T),log(N))

%Question 8
%S5
F1 = [2;3;4;5;1]; F2 = [2;1;3;4;5];
GF = [F1,F2];
%S20
G1 = [2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;1];
G2 = [2;1;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20];
GG = [G1,G2];
```

Question 9

```
n = 14;
count = 0;%number of pairs that generate S_n
for i = 1:100
    A = randperm(n,n);
    B = randperm(n,n);
    G = transpose([A;B]);
    o = Ord(G);
    if o == factorial(n)
        count = count+1;
    end
end
count/100
```