The ABBA Modification of the Josephus Problem

Sarah Burke[1] and Robert Davis[2]

The Josephus Problem is the basis for our research. Instead of killing all but one person, we kill exactly half of our population, which is distinct from the other half. For example, there are *n* bad guys and *n* good guys, and we kill all the bad guys. We have written a program that will find a skip factor that will achieve this goal for any arbitrary ordering of bad guys and good guys.

1. Introduction

Flavius Josephus, who lived from about 37 A.D. to 100 A.D., was an important historical figure and soldier. The Romans, who were led by Vespasian in the battle at Jotapata, captured him, along with forty of his men who jointly decided that a fate of dying would be more honorable than that of becoming a slave. Josephus, a man who loved himself more than anyone else, did not agree with this solution. He tried to sway his men into thinking suicide was not the answer, but was unable to do so. Therefore, he suggested that all the men stand in a circle and each man should kill the man in front of him until all the men, except one, is dead. This last man is the only one that is left to commit suicide. Josephus' soldiers loyally followed his plans, without knowing Josephus escaped to be that last man standing. Rather than kill himself, however, Josephus escaped to hide in a cave, where he was found by the Romans later and eventually was liberated by his captor, we learn this story in the test [3].

The way Josephus calculated which position to take in his circle of men is a very complicated, intriguing problem. He made all 41 men stand in a circle, where every man would kill the man in front of him one at a time. For example, if there were five men, the first man would kill the second man, the third would kill the fourth, the fifth would kill the first, and then the third would kill the fifth. This leaves the person in the third position as the survivor. If Josephus had been in this circle, he would have been in the third position in the circle. This sounds relatively simple, but as more people are added to the circle, it becomes more difficult to predict the survivor. Josephus and his men made a 41-man circle, and Josephus was able to calculate that he should be in the 19^{th} position in order to survive.

Our research problems are derived from the Josephus problem. However, we want to kill, or eliminate, only half of the men in the circle. For example, if we had a circle of 50 bad guys and 50 good guys, we would want to kill all of the bad guys without killing any of the good guys. We concentrate on the case when the number of bad and the number of good guys are equal. We represent the *n* bad guys with 'A' and *n* good guys with 'B'. In our previous example of bad guys and good guys, our *n* is 50, making our total population $2n = 100$.

[1] Department of Mathematics, Marshall University, Huntington, WV 25701; greenvalley81@hotmail.com
[2] Department of Mathematics, Delaware State University, Dover, DE 19901; eversblaze83@yahoo.com

To demonstrate, let n=2. We first find the permutations of A's and B's. Because there are n A's and B's instead of four different letters, the number of permutations is $\frac{4!}{(2)(2)}$=6. Here are the six permutations:

[AABB] [BBAA] [ABBA] [BAAB] **[ABAB]** [BABA]

As far as permutations are concerned, these six are all different. However, considered as cycles, there are only two distinct ones. This would be the first one, AABB, and the fifth one, ABAB. The second, third and fourth cycles are all rotations of the first one. You can see this if you mentally turn each circle clockwise. In each case, you will eventually locate the first cycle. You cannot do this with the fifth permutation. The sixth one can be found to be equivalent to the fifth using the same rotational method, which is why it is not a distinct cycle.

The skip factor, which is referred to as k, is the number of positions you skip in order to determine the next position that will be eliminated. In Josephus' case, the skip factor was 2 because every other person was killed. We have developed an algorithm that takes any arbitrary string S of n A's and n B's and finds a suitable skip factor k which eliminates all the A's without killing any B's. For example, if S=AABAABBB, then k=17 suffices.

2. ABBA Cycles

The method for n=2 seems relatively simple. We know all 6 permutations can be obtained by rotating the 2 distinct cycles AABB and ABAB. As n increases, there are more and more permutations that you must look through to find the distinct cycles. For example, when n=4, there are 70 permutations, and only 10 distinct cycles. Instead of discovering and sifting through all 70 permutations to find the distinct cycles, we wrote a computer program using MAPLE, which calculates permutations and distinct cycles. We can put in any value for n and the program displays the number of permutations, and prints out all of the distinct cycles. A copy of our program with explanations for each step, using n=3, can be seen in Appendix A.

3. The ABBA Elimination Problem

Our next goal is to show that if we have n A's and n B's sitting in a circle then we can discover k, a skip factor, whose use will kill the A's first when we start counting at seat number 1. This problem was first posed in *Mathematical Recreation and Essays* [1]. For example, if we have ABAB then k=2 will automatically kill the first B. Therefore, the skip factor of 2 does not work. However, if k=5, then this will kill the first A, and then the second, leaving all of the B's unharmed. The program we have written finds these appropriate k's, but for any arbitrary string of n A's and n B's. So that we may do the computations easier, we always write the string of A's

and B's so that there is an A in seat number 1. This is done without loss of generality since we can rotate the circle clockwise.

Definition 3.1: We define a metric on a string S of p A's and n B's, where an A is first and where $p \le n$. by: $d(S)=(m_1, m_2, m_3, \ldots, m_p)$ where m_j is the number of B's between successive A's.

Example 3.2: If S=ABAABBBABB, then $d(S)=(1,0,3,2)$.

Lemma 3.3: *If S is a string of p A's and n B's and if* $d(S)=(m_1, m_2, m_3, \ldots, m_p)$, *then we have that* $n= m_1 + m_2 + m_3 + \ldots + m_p$.

Proof: Since m_j is the number of B's between successive A's, the total number of B's, n, must equal the sum of $m_1, m_2, m_3, \ldots, m_p$. Therefore $n= m_1 + m_2 + m_3 + \ldots + m_p$.□

Lemma 3.4: *If* $n \ge 2$ *is an integer, then* $\gcd(n, n-1)=1$.

Proof: Assume, $d=\gcd(n,n-1)$ then, $d|n$ and $d|n-1$ thus $n=xd$ and $n-1=yd$ by substitution we find that $xd-1 = yd$. Therefore, $d|1$ and we conclude that d=1 because the only positive number that divides 1 is 1.□

Lemma 3.5: *Two consecutive odd integers are relatively prime to each other*.

Proof: Let $d=\gcd(n,n+2)$ where n is odd, then $d|n$ and $d|n+2$ thus $n=xd$ and $n+2=yd$. By substitution we find that $xd+2=yd$ and hence that $d(x-y)=2$. Therefore, d can only be 1 or 2. We know $d \neq 2$ since $d|n$ and n is odd. Thus d=1.□

We want to build up to our goal by making the number of A's small and having n B's. Let's start with 2 A's.

Proposition 3.6: *If S is a string of* 2 *A's and n B's, then there exists a* $k \in Z^+$ *whose use kills the A's first. In fact,* $k=m_1(n+2)+1$, *where* $d(S)=(m_1, m_2)$.

Proof: Assume $d(S)=(m_1, m_2)$. If we kill the A in seat 1 first, then the set of modular equations is

$$\begin{cases} k = 1 \bmod (n+2) \\ k = (m_1 + 1) \bmod (n+1) \end{cases}$$

By Lemma 3.4, $n+2$ is relatively prime to $n+1$, now we can use the Chinese Remainder Theorem.

$M = (n+1)(n+2)$
$M_1 = n+1$
$M_2 = n+2$

$(n+1)y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad -1y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad y_1 = -1$
$(n+2)y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad 1y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad y_2 = 1$

Therefore we conclude that
$k = (-1)(1)(n+1) + (1)(m_1+1)(n+2) = m_1(n+2)+1$. □

The situation gets slightly more complicated when we have 3 A's, but the same method applies.

Proposition 3.7*: If S is a string of 3 A's and n B's, then there exists a k∈ Z^+ that will kill the A's first. The value of k depends on the parity of n, m_1, m_2 and m_3 where d(S) = (m_1, m_2, m_3).*

Proof: Assume that d(*S*) = (m_1, m_2, m_3), and number the A's as A^1, A^2, A^3. There are two orders of elimination to consider.

a): If we kill them in the order A^1, A^2, A^3, then the system of equations is as follows:

$$\begin{cases} k = 1 \bmod (n+3) \\ k = (m_1+1) \bmod (n+2) \\ k = (m_2+1) \bmod (n+1) \end{cases}$$

b): If we kill them in the order A^1, A^3, A^2, then the system of equations is as follows:

$$\begin{cases} k = 1 \bmod (n+3) \\ k = (m_1+m_2+2) \bmod (n+2) \\ k = (m_3+m_1+1) \bmod (n+1) \end{cases}$$

Case 1. If *n* is even, then the Chinese Remainder Theorem will give us a solution to **a**. To verify we have to prove that $n+3$, $n+2$, and $n+1$ are pair-wise relatively prime in this case. By Lemma 2, we know that $n+3$ is relatively prime to $n+2$ and $n+2$ is relatively prime to $n+1$. Now we have to prove that $n+3$ is relatively prime to $n+1$. By Lemma 3.5, $n+3$ is relatively prime to $n+1$.

Proof: Assume *n* is even, then we can find a solution to **a.**

$$\begin{cases} k = 1 \bmod (n+3) \\ k = (m_1 + 1) \bmod (n+2) \\ k = (m_2 + 1) \bmod (n+1) \end{cases}$$

Using the Chinese Remainder Theorem,

$$M = (n+3)(n+2)(n+1)$$
$$M_1 = (n+2)(n+1)$$
$$M_2 = (n+3)(n+1)$$
$$M_3 = (n+3)(n+2)$$

$$(n+2)(n+1)y_1 = 1 \bmod (n+3) \quad \Rightarrow \quad 2y_1 = 1 \bmod (n+3) \quad \Rightarrow \quad y_1 = \frac{1}{2}n + 2$$
$$(n+3)(n+1)y_2 = 1 \bmod (n+2) \quad \Rightarrow \quad -1y_2 = 1 \bmod (n+2) \quad \Rightarrow \quad y_2 = -n - 3$$
$$(n+3)(n+2)y_3 = 1 \bmod (n+1) \quad \Rightarrow \quad 2y_3 = 1 \bmod (n+1) \quad \Rightarrow \quad y_3 = \frac{1}{2}n + 2$$

Therefore,

$$k = (n+2)(n+1)(\frac{1}{2}n+2) + (m_1+1)(n+3)(n+1)(-n-3) + (m_2+1)(n+3)(n+2)(\frac{1}{2}n+2).$$

Case 2a.

If n is odd and m_2 is even, then we can solve **a**.

$$\begin{cases} k = 1 \bmod (n+3) \\ k = (m_1 + 1) \bmod (n+2) \\ k = (m_2 + 1) \bmod (n+1) \end{cases}$$

We choose the sub-system $\begin{cases} k = (m_1 + 1) \bmod (n+2) \\ k = (m_2 + 1) \bmod (n+1) \end{cases}$ because if n is odd then $n+3$ and $n+1$ are not relatively prime so we can only choose two out of the three equations for the Chinese Remainder Theorem.

$$M = (n+1)(n+2)$$
$$M_1 = n+1$$
$$M_2 = n+2$$

$$(n+1)y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad -1y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad y_1 = -1$$
$$(n+2)y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad 1y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad y_2 = 1$$

We want to reconcile with the first equation so that we can find a k that works for all 3 equations in system 2.

$k = (-1)(m_1 + 1)(n+1) + (1)(m_2 + 1)(n+2) + (n+2)(n+1)y$ for some $y \in Z$. Hence,

$$k \bmod (n+3) = (-1)(m_1 + 1)(-2) + (1)(m_2 + 1)(-1) + (-1)(-2)y$$
$$= 2m_1 + 2 - m_2 - 1 + 2y.$$

Setting this equation equal to 1 we get $1 = 2m_1 + 2 - m_2 - 1 + 2y$ and $y = \dfrac{-2m_1 + m_2}{2}$.

Substituting y back into the equation for k we obtain

$$k = (-1)(m_1 + 1)(n+1) + (m_2 + 1)(n+2) + \tfrac{1}{2}(m_2 - 2m_1)(n+2)(n+1).$$

Case 2b: Assume n is odd, and m_2 odd, we can solve system **b**.

$$\begin{cases} k = 1 \bmod (n+3) \\ k = (m_1 + m_2 + 2) \bmod (n+2) \\ k = (m_3 + m_1 + 1) \bmod (n+1) \end{cases}$$

We choose the sub-system $\begin{cases} k = (m_1 + m_2 + 2) \bmod (n+2) \\ k = (m_3 + m_1 + 1) \bmod (n+1) \end{cases}$ because if n is odd then $n+3$ and $n+1$ are not relatively prime so we can only choose two out of the three equations when we use the Chinese Remainder Theorem.

$$M = (n+1)(n+2)$$
$$M_1 = n+1$$
$$M_2 = n+2$$

$$(n+1)y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad -1y_1 = 1 \bmod (n+2) \quad \Rightarrow \quad y_1 = -1$$
$$(n+2)y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad 1y_2 = 1 \bmod (n+1) \quad \Rightarrow \quad y_2 = 1$$

We want to reconcile with the first equation.

$k = (m_1 + m_2 + 2)(n+1)(-1) + (m_1 + m_3 + 1)(n+2)(1) + (n+1)(n+2)y$ for some $y \in Z$.

$$k \bmod (n+3) = (m_1 + m_2 + 2)(-2)(-1) + (m_1 + m_3 + 1)(-1)(1) + (-2)(-1)y$$
$$= 2m_1 + 2m_2 + 4 - m_1 - m_3 - 1 + 2y$$
$$= m_1 + 2m_2 - m_3 + 3 + 2y$$
$$= m_1 + m_2 + m_3 + m_2 - 2m_3 + 3 + 2y$$
$$= (n+3) + m_2 - 2m_3 + 2y$$
$$= m_2 - 2m_3 + 2y \bmod (n+3)$$

Setting this equation equal to 1 we obtain $1 = m_2 - 2m_3 + 2y$ and therefore $y = \dfrac{2m_3 - m_2 + 1}{2}$.

Therefore $k = (m_1 + m_2 + 1)(n+1)(-1) + (m_1 + m_3 + 1)(n+2) + \frac{1}{2}(2m_3 - m_2 + 1)(n+1)(n+2)$.□

Finally, we demonstrate that if we have 4 A's and 4 B's. Then we can find the *k* whose use will eliminate the A's without eliminating any of the B's.

Proposition 3.8: *If S is a string of* 4 *A's and* 4 *B's, then there exists a* $k \in Z^+$ *that will kill the A's first. The value of k depends on the parity of n,* m_2 *, and* m_3 *where d(S) =* (m_1, m_2, m_3, m_4)*.*

Proof: Assume that d(*S*) = (m_1, m_2, m_3, m_4), and number the A's as A^1, A^2, A^3,A^4. There are three orders of elimination to consider.

Case 1: If we kill in the order A^1, A^2, A^3, A^4
If m_2 is even we use this system of equations.

$$\begin{cases} k = 1 \bmod 8 \\ k = (m_1 + 1) \bmod 7 \\ k = (m_2 + 1) \bmod 6 \\ k = (m_3 + 1) \bmod 5 \end{cases}$$

We use the last 3 equations when applying the Chinese Remainder Theorem because 7, 6, and 5 are relatively prime.

$M = 7 \cdot 6 \cdot 5 = 210$
$M_1 = 30$
$M_2 = 35$
$M_3 = 42$
$30y_1 = 1 \bmod 7 \quad \Rightarrow \quad 2y_1 = 1 \bmod 7 \quad \Rightarrow \quad y_1 = 4$
$35y_2 = 1 \bmod 6 \quad \Rightarrow \quad 5y_2 = 1 \bmod 6 \quad \Rightarrow \quad y_2 = 5$
$42y_3 = 1 \bmod 5 \quad \Rightarrow \quad 2y_3 = 1 \bmod 5 \quad \Rightarrow \quad y_3 = 3$

$k = (m_1 + 1)(30)(4) + (m_2 + 1)(35)(5) + (m_3 + 1)(42)(3) + 210y$
$k = 120(m_1 + 1) + 175(m_2 + 1) + 126(m_3 + 1) + 210y$

We now want find a *k* that is sufficient for all 4 of the equations.

Since $k \bmod 8 = 0(m_1+1) + 7(m_2+1) + 6(m_3+1) - 6y$ we need $1 = 7(m_2+2) + 6(m_3+1) - 6y$ and thus $y = \dfrac{7(m_2+1) + 6(m_3+1) - 1}{6}$.

Substituting into our first equation,

$$k = 120(m_1+1) + 175(m_2+1) + 126(m_3+1) + 210\left(\frac{7(m_2+1) + 6(m_3+1) - 1}{6}\right)$$

Therefore, $k = 120m_1 + 420m_2 + 336m_3 + 841$. □

Case 2: If we kill in the order A^1, A^2, A^4, A^3
If m_2 is odd and m_3 is even, we use this system of equations.

$$\begin{cases} k = 1 \bmod 8 \\ k = (m_1+1) \bmod 7 \\ k = (m_2+m_3+2) \bmod 6 \\ k = (m_1+m_2+m_4+1) \bmod 5 \end{cases}$$

We use the last 3 equations when applying the Chinese Remainder Theorem because they are relatively prime by Lemma 3.4 and Lemma 3.5.

$M = 7 \cdot 6 \cdot 5 = 210$
$M_1 = 30$
$M_2 = 35$
$M_3 = 42$
$30y_1 = 1 \bmod 7 \quad \Rightarrow \quad 2y_1 = 1 \bmod 7 \quad \Rightarrow \quad y_1 = 4$
$35y_2 = 1 \bmod 6 \quad \Rightarrow \quad 5y_2 = 1 \bmod 6 \quad \Rightarrow \quad y_2 = 5$
$42y_3 = 1 \bmod 5 \quad \Rightarrow \quad 2y_3 = 1 \bmod 5 \quad \Rightarrow \quad y_3 = 3$

$k = (m_1+1)(30)(4) + (m_2+m_3+2)(35)(5) + (m_1+m_2+m_4+1)(42)(3) + 210y$
$k = 120(m_1+1) + 175(m_2+m_3+2) + 126(m_1+m_2+m_4+1) + 210y$

We now want to find a *k* that is sufficient for all 4 of the equations.

$$k \bmod 8 = 0(m_1+1) + 7(m_2+m_3+2) + 6(m_1+m_2+m_4+1) - 6y$$
$$1 = 7(m_2+m_3+2) + 6(m_1+m_2+m_4+1) - 6y$$
$$y = \frac{7(m_2+m_3+2) + 6(m_1+m_2+m_4+1) - 1}{6}$$

We have now found our *y*, so we need to substitute it into our first equation.

$$k = 120(m_1+1) + 175(m_2+m_3+2) + 126(m_1+m_2+m_4+1) + 210\left(\frac{7(m_2+m_3+2) + 6(m_1+m_2+m_4+1) - 1}{6}\right)$$

Therefore, $k = 456m_1 + 756m_2 + 420m_3 + 336m_4 + 1261$. □

Case 3: If we kill in the order A^1, A^3, A^2, A^4
If m_2 and m_3 are odd, we use this system of equations.

$$\begin{cases} k = 1 \bmod 8 \\ k = (m_1 + m_2 + 2) \bmod 7 \\ k = (m_1 + m_3 + m_4 + 2) \bmod 6 \\ k = (m_2 + m_3 + 1) \bmod 5 \end{cases}$$

We use the last 3 equations when applying the Chinese Remainder Theorem because they are relatively prime by Lemma 3.4 and Lemma 3.5.

$$M = 7 \cdot 6 \cdot 5 = 210$$
$$M_1 = 30$$
$$M_2 = 35$$
$$M_3 = 42$$
$$30y_1 = 1 \bmod 7 \quad \Rightarrow \quad 2y_1 = 1 \bmod 7 \quad \Rightarrow \quad y_1 = 4$$
$$35y_2 = 1 \bmod 6 \quad \Rightarrow \quad 5y_2 = 1 \bmod 6 \quad \Rightarrow \quad y_2 = 5$$
$$42y_3 = 1 \bmod 5 \quad \Rightarrow \quad 2y_3 = 1 \bmod 5 \quad \Rightarrow \quad y_3 = 3$$

$$k = (m_1 + m_2 + 2)(30)(4) + (m_1 + m_3 + m_4 + 2)(35)(5) + (m_2 + m_3 + 1)(42)(3) + 210y$$
$$k = 120(m_1 + m_2 + 2) + 175(m_1 + m_3 + m_4 + 2) + 126(m_2 + m_3 + 1) + 210y$$

We now want to find a *k* that is sufficient for all 4 of the equations.

$$k \bmod 8 = 0(m_1 + m_2 + 2) + 7(m_1 + m_3 + m_4 + 2) + 6(m_2 + m_3 + 1) - 6y$$
$$1 = 7(m_1 + m_3 + m_4 + 2) + 6(m_2 + m_3 + 1) - 6y$$
$$y = \frac{7(m_1 + m_3 + m_4 + 2) + 6(m_2 + m_3 + 1)) - 1}{6}$$

We have now found our *y*, so we need to substitute it into our first equation.

$$k = 120(m_1 + m_2 + 2) + 175(m_1 + m_3 + m_4 + 2) + 126(m_2 + m_3 + 1) + 210\frac{7(m_1 + m_3 + m_4 + 2) + 6(m_2 + m_3 + 1)) - 1}{6}$$

Therefore, $k = 540m_1 + 456m_2 + 756m_3 + 420m_4 + 1381$. □

The above proofs proved that there is a possible *k* for any arbitrary string of *n* A's and *n* B's. With this information, we are able to write an algorithm to find a *k* for any arbitrary string of *n* A's and *n* B's and know that it will work. We did this in the form of a program using MAPLE. We first entered our arbitrary string. The program then finds all the permutations, and all the distinct cycles. It shifts our arbitrary string of *n* A's and *n* B's until it matches with a distinct cycle. This step is in our program so that we will be able to start at the appropriate A so our *k* will work. Next, the program labels the spots that hold the A's. The next portion of the program, derived from [2], searches for a *k* that first kills all the A's. This *k* is then printed, along with the order of the deaths of the A's. (See Appendix B)

The reason for writing a program instead of computing everything by hand becomes evident as *n* increases. For *n*=2, we saw that there are only six permutations and two cycles. However, when *n*=7, there are 3432 permutations, and 246 cycles. Therefore, this program is not only helpful, but absolutely necessary when studying an *n* larger than 4. (See Appendix C)

4. Conclusion

Using the Josephus Problem as a basis, we were able to modify this problem and write two programs. One of these programs takes any arbitrary string of *n* A's and *n* B's, permutes it, and then finds the associated distinct cycles. The second program also does this, but then it finds a skip factor *k* which kills all the A's and no B's. From this, using the Chinese Remainder Theorem, we proved that for any arbitrary string of *n* A's and *n* B's, it is possible to find a *k* whose use will eliminate all of the A's without harming any B's.

With any research problem, there are always more questions that can be posed. In this case, there are 2 questions we would like to challenge you with. The first is this: Can this program be more efficient, or is it possible to make a more efficient program? The second question is: What if there are more A's than B's? Is it still possible to find a sufficient *k*? We hope you challenge yourself to answer these questions, and take advantage of the research and the programs we have created as a basis for further study.

Appendix A

A.1) Program to find Distinct Cycles.

```
1> restart;
2> with(combinat, permute, numbcomb):
3> n:=3:
4> TheEnd:=numbcomb(2*n,n);counter:=1:
```

TheEnd := 20

```
5> g:=permute([seq(a,i=1..n),seq(b,i=1..n)]):m:=1:
6> while counter <= TheEnd do
7> f:=g[1];
8> hold[m]:=f;
9> j:=1;k:=2;
10> while k > 1 and j <= 2*n do
11> temp:=f[1];
12> for i from 2 to 2*n do
13> f[i-1]:=f[i];od;
14> f[2*n]:=temp;
15> member(f,g,'k');
16> g:=subsop(k=NULL,g);counter:=counter+1;j:=j+1;od;m:=m+1;od:
17> for i from 1 to (m-1) do
18> print(i,hold[i]);od;
```

1, [a, a, a, b, b, b]
2, [a, a, b, a, b, b]
3, [a, a, b, b, a, b]
4, [a, b, a, b, a, b]

A.2) Comments

1 - Restarts program.
2 - Command to introduce permute and Numbcomb so that we can use them later.
3 – Initializes variable n for n number of A's and n number of B's.
4 - Numbcomb does 2*n* choose *n* to find the total number of permutations, *counter=1*.
5 - Permutes the A's and B's, initializes *m*=1.
6 – While the number of counted permutations is less than total number of permutations.
7 - Initialize *f* to the first permutation.
8 - Hold variable holds the distinct permutations.
9 - Initialize *j* and *k* for future use.
10 - As long as $k > 1$ and $j \leq 2n$, the program keeps running.
11 - Temporary variable used to hold *f*'s first letter.
12 - Loop to shift all a's and b's.
13 - Actually shifts a's and b's to left.

14 - Puts the first letter in the last position.
15 - Finds the position *k*, where *f* is located in *g*.
16 - Erases non-distinct perms, and increments all variables used.
17 - Loop to print all distinct perms in order.
18 - Prints the results of program.

Appendix B

B.1) Program find a Killer k.

```
1> restart;
2> with(combinat, permute, numbcomb):
3> n:=5:
4> TheEnd:=numbcomb(2*n,n);counter:=1:
```

$$TheEnd := 252$$

```
5> g:=permute([seq(a,i=1..n),seq(b,i=1..n)]):m:=1:
6> while counter <= TheEnd do
7> f:=g[1]:
8> hold[m]:=f;
9> j:=1:k:=2:
10> while k > 1 and j <= 2*n do
11> temp:=f[1];
12> for i from 2 to 2*n do
13> f[i-1]:=f[i];od:
14> f[2*n]:=temp;
15> member(f,g,'k'):
16> g:=subsop(k=NULL,g);counter:=counter+1;j:=j+1;od;m:=m+1:od:
17> k:=[a,a,a,b,b,a,b,b,a,b];
```

$$k := [a, a, a, b, b, a, b, b, a, b]$$

```
18> n:=5:i:=1:
19> while i <= (m-1) do
20> s:=hold[i];
21> for p from 1 to 2*n do
22> temp2:=s[1];
23> for t from 2 to 2*n do
24> s[t-1]:=s[t];od:
25> s[2*n]:=temp2;
26> if s=k then print(p mod (2*n));print(i,hold[i]);i:=m;fi;
27>  od:i:=i+1:  od:
```

$$0$$

$$12, [a, a, a, b, b, a, b, b, a, b]$$

```
28> P:={}:
29> for j from 1 to (2*n) do
30> if s[j]=a then
```

```
31> P:=P union {j} fi;od;
32> print(P);
```

{9, 6, 3, 2, 1}

```
> with(group):
33> n:=10:  maxk:=lcm($2..n):
34> J:=array(1..n):
35> store:=array(1..maxk):
36> for k from 1 to maxk do
37> f:=[$1..n];
38> m:=1:
39> for L from n by -1 to n/2 do
40> usek := k mod L;
41> twofs:=[seq(f[i],i=1..L),seq(f[i],i=1..L)];
42> usethisk :=usek + L;
43> J[m]:=twofs[usethisk];
44> f:=[seq(twofs[i],i=(usek+1)..(usethisk-1))];
45> m:=m+1:  od:
46> J[n]:=f[1]:
47> tempJ:=[seq(J[i],i=1..n/2)]:
48> store[k]:=tempJ: od:
49> x:=1:y:=1:
50> while x <= maxk and y<maxk+1 do
51> w:=store[x]:
52> if P={seq(w[a],a=1..n/2)} then
53> print(x);y:=maxk+1;fi;
54> x:=x+1:
55> od;print((x-1),store[x-1]);
```

483, [3, 9, 2, 1, 6]

B.2) Comments

1 - Restarts program.
2 - Command to introduce permute and Numbcomb so that we can use them later.
3 – Initializes variable n for n number of A's and n number of B's.
4 - Numbcomb does $2n$ choose n to find the total number of permutations, *counter=1*.
5 - Permutes the A's and B's, initializes m=1.
6 – While the number of counted permutations is less than total number of permutations.
7 - Initialize *f* to the first permutation.
8 - Hold variable holds the distinct permutations.
9 - Initialize j and k for future use.
10 - As long as $k > 1$ and $j \leq 2n$, the program keeps running.
11 - Temporary variable used to hold *f*'s first letter.
12 - Loop to shift all a's and b's.

13 - Actually shifts a's and b's to left.
14 - Puts the first letter in the last position.
15 - Finds the position *k*, where *f* is located in *g*.
16 - Erases non-distinct perms, and increments all variables used.
17 - Creates a arbitrary string of *n* a's and *n* b's.
18 - Initializes *n* to be equal to 5 and *i* to 1.
19 - While *i* is less than the number of distinct cycles minus 1.
20 - Initialize to the hold[*i*](first distinct cycle) and as *i* changes the distinct cycle changes.
21 - For loop to help shift distinct cycles around and compare.
22 - Temporary variable to check first element in *s*.
23 - Loop to shift the elements of *s* around.
24 - Actually shifts the elements over one space to the left.
25 - Sets the last element of *s* to the temporary variable (the first element).
26 - If permutation *s* equals permutation *k*, then print the number of shifts and end the loop.(This number of shifts tells us the position we want to start at in the arbitrary string of a's and b's to find an appropriate *k*.)
27 - If not then increment *i* by 1.
28 - Initialize *P* to be an empty set.
29 - Loop to go through the string to find positions of the a's.
30 - If *s*[*i*] is an a then…
31 - Put it into *P*'s set.
32 - Print *P*.
33 - Number of people, range of *k*.
34 - Initialize *J*, an array from 1 to *n*.
35 - Define array to store cycles.
36 - Loop to calculate death order and cycles for *k*.
37 - Makes *f* = [1,2,3,...,*m*].
38 - Each time we store the nth killing.
39 - Decrementing *L*.
40 - Number of seat to kill (0=last person).
41 - Vector with two copies of *f*.
42 - Number of seat to kill from two *f* vector.
43 - Record the seat of most recent killed.
44 - Change *f* so those livings remain and person after most recent killed in first position.
45 - Increment *n* and end loop for this *k*.
46 - Last person to die is the last one living.
47 - Turn *J* into a list.
48 - Store cycles in store array.
49 - Initialize *x* and *y* to 1.
50 - While $x \leq$ max value of *k* and *y*<max value of *k* + 1…
51 - Initialize *w* to store[*x*] an array.
52 - If *P* equal set of the elements looking through…
53 - Print *x*, stop the loop.
54 - If not, increment *x* by 1.
55 - Print $x - 1$ (the skip factor *k*), and the set that equals *P*.

Appendix C

C.1)

Distinct cycles for $n = 2$	Skip factor k
[a, a, b, b]	1
[a, b, a, b],	5

C.2)

Distinct cycles for $n = 3$	Skip factor k
[a, a, a, b, b, b]	1
[a, a, b, a, b, b]	4
[a, a, b, b, a, b]	17
[a, b, a, b, a, b]	19

C.3)

Distinct cycles for $n = 4$	Skip factor k
[a, a, a, a, b, b, b, b]	1
[a, a, a, b, a, b, b, b]	162
[a, a, a, b, b, a, b, b]	130
[a, a, a, b, b, b, a, b]	26
[a, a, b, a, a, b, b, b]	17
[a, a, b, a, b, a, b, b]	89
[a, a, b, a, b, b, a, b]	98
[a, a, b, b, a, a, b, b]	18
[a, a, b, b, a, b, a, b]	57
[a, b, a, b, a, b, a, b]	27

Acknowledgements

We would like to acknowledge Miami University for providing facilities for research, and for having us out here to do research. We also thank NSA and NSF for the funding of this program. We recognize Dr. Chawne Kimber for her moral and technical support throughout the process of our project. Without Dr. Kimber, we would not have gotten as far as we did. Also we would like to recognize Angela Grant for being a great Graduate assistant and helping us along with our project. She was a big part in us finishing our project and without her leadership we not have been able to accomplish what we did. Special thanks also goes to Dr. Waikar, Dr. Dowling, Dr. Davenport, Moira Miller, and Bonita Porter for organizing this program. To all of the above, we thank you for all of your time and effort in helping us complete our project and giving us this opportunity.

References

[1] W.W. Rouse Ball and H.S.M. Coxeter, *Mathematical Recreations and Essays*, Dover: Toronto, 1987.

[2] L Casburn, A. Grant, T Phan, The Orthogonal Josephus Problem, SUMSRI Journal, 2001.

[3] Josephus, *Josephus: The Complete Works,* (William Whiston, A.M., translator), Nelson: Nashville, 1998.