Quest Journals
Journal of Research in Applied Mathematics
Volume 12 ~ Issue 8 (August 2026) pp: 24-39
ISSN (Online): 2394-0743 ISSN (Print): 2394-0735
www.questjournals.org

Research Paper

A Computational Approach to the Class Equation of the Symmetric Group S_n Using Python Programming

Naresh A. Wagh[1] and Kishor K. Gawade[2]
[1]*Department of Mathematics, KET'S V.G. Vaze college(Empowered Autonomous), Mulund(East), Mumbai-400081 Maharashtra, India*
[2]*Department of Mathematics, KET'S V.G. Vaze college(Empowered Autonomous), Mulund(East),Mumbai-400081 Maharashtra, India*

Abstract

The symmetric group S_n is one of the fundamental structures in finite group theory, and its conjugacy classes are characterized by the cycle types of permutations. The class equation of S_n can therefore be obtained from the integer partitions of n, where each partition corresponds to a unique conjugacy class. In this work, a computational approach is developed using Python to generate the conjugacy classes of the symmetric group S_n for an arbitrary positive integer n. The proposed algorithm generates all integer partitions of n, determines the corresponding cycle types, calculates the centralizer order, and computes the size of each conjugacy class using the formula

$$|C_\lambda| = \frac{n!}{\prod_{i=1}^{n} i^{m_i} m_i!},$$

where m_i denotes the number of cycles of length i. The individual class sizes are then summed to obtain the complete class equation. The computational results are automatically verified by confirming that the sum of the sizes of all conjugacy classes is equal to the order of the symmetric group, $n!$. The number of conjugacy classes is also verified to be equal to the partition number $p(n)$. The proposed Python-based framework provides a systematic computational method for exploring conjugacy classes and class equations of symmetric groups and can be extended to investigate their combinatorial and computational properties for increasing values of n.

Keywords: Symmetric group, S_n, conjugacy classes, class equation, cycle type, integer partitions, centralizer, Python, computational group theory.

*Received 05 Aug., 2026; **R**evised 14 Aug., 2026; **A**ccepted 16 Aug., 2026 © The author(s) 2026.*
P**ublished with open access at **www.questjournals.org

I. INTRODUCTION

Group theory is an important branch of abstract mathematics that studies algebraic structures known as groups. Among finite groups, the symmetric group S_n plays a fundamental role because it consists of all permutations of n objects and has order [1, 2, 3]

$$|S_n| = n!.$$

The study of conjugacy classes is important in understanding the internal structure of a finite group. Two elements a, b of a group G are said to be conjugate if there exists an element $g \in G$ such that [1, 2, 3]

$$b = gag^{-1}.$$

Conjugacy is an equivalence relation on a group and therefore partitions the group into disjoint conjugacy classes.

For the symmetric group S_n, conjugacy classes are completely determined by the cycle structure of permutations.[9, 4, 6] Consequently, the conjugacy classes of S_n are in one-to-one correspondence with the integer partitions of n.

If

$$n = m_1 + 2m_2 + \cdots + nm_n,$$

then the corresponding cycle type can be written as

$$1^{m_1} 2^{m_2} \cdots n^{m_n}.$$

The size of the conjugacy class corresponding to this cycle type is

$$|C_\lambda| = \frac{n!}{\prod_{i=1}^{n} i^{m_i} m_i!}.$$

Therefore, the class equation of S_n can be obtained by adding the sizes of all conjugacy classes.

The purpose of this work is to develop a Python-based computational framework for generating these conjugacy classes and verifying the resulting class equation for arbitrary positive integers n.

2 Objectives

The main objectives of this work are:

1. To study the conjugacy classes of the symmetric group S_n.
2. To generate all integer partitions of n computationally.
3. To associate each partition of n with a cycle type of S_n.
4. To calculate the centralizer order corresponding to each cycle type.
5. To calculate the size of every conjugacy class.
6. To construct the complete class equation of S_n.
7. To verify computationally that the sum of all class sizes is equal to $n!$.
8. To verify that the number of conjugacy classes is equal to the partition number $p(n)$.
9. To provide a reusable Python algorithm for arbitrary values of n.

3 Mathematical Preliminaries

3.1 Symmetric Group

The symmetric group on n symbols, denoted by S_n, is the group of all permutations of the set [1, 2, 3]

$$\{1, 2, \ldots, n\}.$$

The order of S_n is

$$\boxed{|S_n| = n!}.$$

For example,

$$|S_3| = 3! = 6,$$

and

$$|S_4| = 4! = 24.$$

3.2 Conjugacy

Let G be a group and let $a, b \in G$. The elements a and b are conjugate if there exists $g \in G$ such that[1, 2, 3]

$$b = gag^{-1}.$$

The conjugacy class of a is

$$Cl_G(a) = \{gag^{-1} : g \in G\}.$$

The conjugacy classes form a partition of the group G.[1, 3]

3.3 Cycle Type

Every permutation can be expressed as a product of disjoint cycles.[1, 2, 4] The lengths of these cycles determine the cycle type of the permutation.[4, 9, 6]

For example,

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

has cycle type

$$3\,2\,1.$$

In general, a cycle type can be represented as

$$1^{m_1}2^{m_2}\cdots n^{m_n},$$

where

$$\sum_{i=1}^{n} im_i = n.$$

3.3 Cycle Type

Every permutation can be expressed as a product of disjoint cycles.[1, 2, 4] The lengths of these cycles determine the cycle type of the permutation.[4, 9, 6]

For example,

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

has cycle type

$$3\,2\,1.$$

In general, a cycle type can be represented as

$$1^{m_1}2^{m_2}\cdots n^{m_n},$$

where

$$\sum_{i=1}^{n} im_i = n.$$

4 Integer Partitions and Conjugacy Classes

A partition of a positive integer n is a representation of n as a sum of positive integers, where the order of the summands is irrelevant.[19, 18]

For example, the partitions of 4 are

$$4, \quad 3+1, \quad 2+2, \quad 2+1+1, \quad 1+1+1+1.$$

Thus,

$$p(4) = 5.$$

The number of conjugacy classes of S_n is exactly $p(n)$.[4, 9, 6]
Therefore,

$$\boxed{k(S_n) = p(n)},$$

where $k(S_n)$ denotes the number of conjugacy classes of S_n.

5 Conjugacy Class Size

Let a permutation in S_n have cycle type[4, 9]

$$\lambda = 1^{m_1} 2^{m_2} \cdots n^{m_n}.$$

The order of the centralizer of the permutation is [4, 9]

$$\boxed{z_\lambda = \prod_{i=1}^{n} i^{m_i} m_i!}.$$

The order of the centralizer of the permutation is [4, 9]

$$\boxed{z_\lambda = \prod_{i=1}^{n} i^{m_i} m_i!}.$$

Hence the size of the corresponding conjugacy class is

$$|C_\lambda| = \frac{|S_n|}{|C_{S_n}(\lambda)|}.$$

Therefore, [4, 9, 6]

$$\boxed{|C_\lambda| = \frac{n!}{\prod_{i=1}^{n} i^{m_i} m_i!}}.$$

This formula is used in the Python implementation to calculate the size of each conjugacy class.

6 Class Equation of S_n

If the conjugacy classes of S_n are [1, 3]

$$C_1, C_2, \ldots, C_{p(n)},$$

then

$$S_n = C_1 \cup C_2 \cup \cdots \cup C_{p(n)}$$

and the classes are pairwise disjoint.
Consequently, [1, 3]

$$|S_n| = |C_1| + |C_2| + \cdots + |C_{p(n)}|.$$

Since

$$|S_n| = n!,$$

the class equation is [4, 9]

$$\boxed{n! = \sum_{\lambda \vdash n} \frac{n!}{\prod_{i=1}^{n} i^{m_i} m_i!}}.$$

Thus,

$$\boxed{\sum_{\lambda \vdash n} |C_\lambda| = n!}.$$

This identity is used as the main computational verification in the proposed algorithm.

7 Proposed Computational Method

The proposed method consists of the following steps.

1. Input a positive integer n.
2. Generate all integer partitions of n.
3. Interpret each partition as a cycle type of S_n.
4. Count the multiplicity m_i of each cycle length i.

$$S_n = C_1 \cup C_2 \cup \cdots \cup C_{p(n)}$$

and the classes are pairwise disjoint.
Consequently, [1, 3]

$$|S_n| = |C_1| + |C_2| + \cdots + |C_{p(n)}|.$$

Since

$$|S_n| = n!,$$

the class equation is [4, 9]

$$\boxed{n! = \sum_{\lambda \vdash n} \frac{n!}{\prod_{i=1}^{n} i^{m_i} m_i!}}.$$

Thus,

$$\boxed{\sum_{\lambda \vdash n} |C_\lambda| = n!}.$$

This identity is used as the main computational verification in the proposed algorithm.

7 Proposed Computational Method

The proposed method consists of the following steps.

1. Input a positive integer n.
2. Generate all integer partitions of n.
3. Interpret each partition as a cycle type of S_n.
4. Count the multiplicity m_i of each cycle length i.
5. Calculate the centralizer order

$$z_\lambda = \prod_i i^{m_i} m_i!.$$

6. Calculate the conjugacy class size

$$|C_\lambda| = \frac{n!}{z_\lambda}.$$

7. Store all class sizes.
8. Calculate

$$\sum_\lambda |C_\lambda|.$$

9. Compare this sum with $n!$.
10. Verify that

$$\sum_{\lambda} |C_\lambda| = n!.$$

8 Algorithm

Algorithm for Class Equation of S_n

Step 1: Read n.

Step 2: Generate all partitions of n.

Step 3: For every partition λ:

(a) Determine the multiplicities m_i.

(b) Calculate

$$z_\lambda = \prod_i i^{m_i} m_i!.$$

(c) Calculate

$$|C_\lambda| = \frac{n!}{z_\lambda}.$$

Step 4: Add all conjugacy class sizes.

Step 5: Display the class equation.

Step 6: Verify whether the sum is equal to $n!$.

9 Python Implementation

The following Python program implements the proposed computational method.

```
1
2  from math import factorial
3  from collections import Counter
4
5
6  # Generate all integer partitions of n
7  def partitions(n, max_part=None):
8
9      if n == 0:
10         yield []
11         return
12
13     if max_part is None:
14         max_part = n
15
16     for first in range(min(max_part, n), 0, -1):
17
```

```
        for rest in partitions(n - first, first):

            yield [first] + rest


# Calculate centralizer order
def centralizer_order(partition):

    counts = Counter(partition)

    z = 1

    for k, m in counts.items():

        z *= (k ** m) * factorial(m)

    return z


# Calculate conjugacy class size
def class_size(n, partition):

    z = centralizer_order(partition)

    return factorial(n) // z


# Convert partition into cycle type notation
def cycle_type(partition):

    counts = Counter(partition)

    result = []

    for k in sorted(counts.keys(), reverse=True):

        m = counts[k]

        if m == 1:

            result.append(str(k))

        else:

            result.append(f"{k}^{m}")

    return " ".join(result)


# Generate class equation of S_n
def class_equation_Sn(n):

    if n < 1:

        print("Please enter a positive integer.")

        return

```

```
    group_order = factorial(n)

    partition_list = list(partitions(n))

    class_sizes = []

    print("=" * 80)

    print(f"CLASS EQUATION OF THE SYMMETRIC GROUP S_{n}")

    print("=" * 80)

    print(f"\nOrder of S_{n} = {n}! = {group_order}")

    print("\nConjugacy Classes")

    print("-" * 80)

    print(
        f"{'No.':<6}"
        f"{'Partition':<20}"
        f"{'Cycle Type':<20}"
        f"{'Centralizer':<15}"
        f"{'Class Size':<15}"
    )

    print("-" * 80)

    for i, partition in enumerate(partition_list, 1):

        z = centralizer_order(partition)

        size = class_size(n, partition)

        class_sizes.append(size)

        print(
            f"{i:<6}"
            f"{str(partition):<20}"
            f"{cycle_type(partition):<20}"
            f"{z:<15}"
            f"{size:<15}"
        )

    print("-" * 80)

    number_of_classes = len(class_sizes)

    total = sum(class_sizes)

    print(
        f"\nNumber of conjugacy classes = "
        f"{number_of_classes}"
    )

    print(
        f"Partition number p({n}) = "
        f"{number_of_classes}"
```

```
    )

    print(
        f"\nSum of class sizes = {total}"
    )

    print(
        f"Order of S_{n} = {group_order}"
    )

    # Construct class equation

    equation = " + ".join(
        map(str, class_sizes)
    )

    print("\nCLASS EQUATION")

    print("-" * 80)

    print(
        f"{group_order} = {equation}"
    )

    print("\nVerification")

    print("-" * 80)

    print(
        f"{equation} = {total} = {group_order}"
    )

    if total == group_order:

        print(
            "\nClass equation verified successfully."
        )

    else:

        print(
            "\nVerification failed."
        )


# Main program

n = int(
    input("Enter n for S_n: ")
)

class_equation_Sn(n)
```

10 Computational Results

The proposed algorithm can be used for arbitrary positive integers *n*.

10.1 Example: S_3

The partitions of 3 are

$$3, \quad 2+1, \quad 1+1+1.$$

The corresponding conjugacy class sizes are

$$2, \quad 3, \quad 1.$$

Therefore,

$$\boxed{6 = 1+3+2}$$

and

$$1+3+2 = 6 = 3!.$$

The number of conjugacy classes is

$$p(3) = 3.$$

10.2 Example: S_4

The partitions of 4 are

$$4, \quad 3+1, \quad 2+2, \quad 2+1+1, \quad 1+1+1+1.$$

The corresponding conjugacy class sizes are

$$6, \quad 8, \quad 3, \quad 6, \quad 1.$$

Hence the class equation is

$$\boxed{24 = 1+6+3+8+6}.$$

Therefore,

$$1+6+3+8+6 = 24 = 4!.$$

The number of conjugacy classes is

$$p(4) = 5.$$

10.3 Example: S_5

The partitions of 5 are

$$5, \quad 4+1, \quad 3+2, \quad 3+1+1, \quad 2+2+1, \quad 2+1+1+1, \quad 1+1+1+1+1.$$

The corresponding class sizes are

$$24, \quad 30, \quad 20, \quad 20, \quad 15, \quad 10, \quad 1.$$

Thus,

$$\boxed{120 = 1 + 10 + 15 + 20 + 20 + 30 + 24}.$$

Hence,

$$1 + 10 + 15 + 20 + 20 + 30 + 24 = 120 = 5!.$$

The number of conjugacy classes is

$$p(5) = 7.$$

11 Summary of Computational Results

The results for the first few symmetric groups are shown below.

Table 1: Number of conjugacy classes and class equations

n	$\lvert S_n\rvert$	$p(n)$	Class Equation
1	1	1	$1 = 1$
2	2	2	$2 = 1 + 1$
3	6	3	$6 = 1 + 3 + 2$
4	24	5	$24 = 1 + 6 + 3 + 8 + 6$
5	120	7	$120 = 1 + 10 + 15 + 20 + 20 + 30 + 24$
6	720	11	$720 =$ sum of 11 class sizes

The computational results demonstrate that

$$\boxed{\sum_{\lambda \vdash n} |C_\lambda| = n!}$$

for the tested values of n.

12 Verification of the Class Equation

The main computational verification is based on the identity [1, 3]

$$S_n = C_1 \cup C_2 \cup \cdots \cup C_{p(n)},$$

where the conjugacy classes are pairwise disjoint.
Taking cardinalities gives [1, 3]

$$|S_n| = \sum_{i=1}^{p(n)} |C_j|.$$

Since

$$|S_n| = n!,$$

we obtain

$$\boxed{n! = \sum_{j=1}^{p(n)} |C_j|}.$$

The Python program independently calculates every class size and then computes their sum. Equality between the computed sum and $n!$ provides a direct computational verification of the class equation.

13 Discussion

The computational approach provides a systematic way of constructing the class equation of S_n. The use of integer partitions avoids the direct enumeration of all $n!$ permutations. This is particularly useful because the number of permutations grows factorially with n, whereas the number of conjugacy classes is equal to the partition number $p(n)$.

For example,

$$|S_{10}| = 10! = 3,628,800,$$

whereas

$$p(10) = 42.$$

Thus, instead of explicitly generating millions of permutations, the proposed method works with only 42 cycle types.

This demonstrates the usefulness of the partition-based approach for computational investigations of symmetric groups.

14 Computational Considerations

A direct computational approach that generates all permutations of S_n requires consideration of

$$n!$$

objects.

In contrast, the proposed method generates integer partitions of n, and the number of these partitions is

$$p(n).$$

The asymptotic behavior of the partition function is approximately [20]

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right).$$

Thus, the partition-based approach provides a substantial reduction compared with direct permutation enumeration for many values of n.

15 Advantages of the Proposed Method

The main advantages are:

1. It works for arbitrary positive integers n.
2. It does not require direct enumeration of all $n!$ permutations.
3. It uses the well-established relationship between partitions and conjugacy classes.
4. It automatically calculates centralizer orders.
5. It automatically calculates conjugacy class sizes.
6. It generates the complete class equation.
7. It verifies the class equation computationally.
8. It can be used as an educational tool for computational group theory.

16 Limitations

Although the proposed approach is computationally more efficient than direct enumeration of all permutations, the partition function $p(n)$ also grows rapidly as n increases. Consequently, very large values of n may require additional optimization.

The present implementation focuses on the class equation and does not explicitly construct every permutation or every conjugacy class as a list of permutations.

15 Advantages of the Proposed Method

The main advantages are:

1. It works for arbitrary positive integers n.
2. It does not require direct enumeration of all $n!$ permutations.
3. It uses the well-established relationship between partitions and conjugacy classes.
4. It automatically calculates centralizer orders.
5. It automatically calculates conjugacy class sizes.
6. It generates the complete class equation.
7. It verifies the class equation computationally.
8. It can be used as an educational tool for computational group theory.

16 Limitations

Although the proposed approach is computationally more efficient than direct enumeration of all permutations, the partition function $p(n)$ also grows rapidly as n increases. Consequently, very large values of n may require additional optimization.

The present implementation focuses on the class equation and does not explicitly construct every permutation or every conjugacy class as a list of permutations.

17 Future Work

The proposed computational framework can be extended in several directions.

1. Development of optimized algorithms for very large n.
2. Graphical visualization of conjugacy class sizes.
3. Comparison between direct permutation enumeration and partition-based computation.
4. Computational study of conjugacy classes of alternating groups A_n.
5. Extension to other finite groups.
6. Investigation of computational complexity as a function of n.
7. Development of an interactive Python-based educational tool.
8. Investigation of relationships between partition functions, conjugacy classes, and representation theory.

18 Conclusion

In this work, a Python-based computational approach for generating and verifying the class equation of the symmetric group S_n has been presented. The method uses the fundamental correspondence between integer partitions of n and conjugacy classes of S_n. For each partition, the corresponding centralizer order and conjugacy class size are calculated using

$$z_\lambda = \prod_{i=1}^{n} i^{m_i} m_i!$$

and

$$|C_\lambda| = \frac{n!}{z_\lambda}.$$

The class equation is then obtained by summing all conjugacy class sizes:

$$\boxed{\sum_{\lambda \vdash n} |C_\lambda| = n!}.$$

The computational verification confirms that the sum of all conjugacy class sizes equals the order of the symmetric group. Furthermore, the number of conjugacy classes is verified to be equal to the partition number $p(n)$.

The proposed implementation provides a simple and systematic framework for computational exploration of conjugacy classes and class equations and offers a useful connection between abstract group theory, integer partitions, and computer-based mathematical computation.

References

[1] D. S. Dummit and R. M. Foote, *Abstract Algebra*, 3rd Edition, John Wiley & Sons, 2004.

[2] J. J. Rotman, *An Introduction to the Theory of Groups*, 4th Edition, Springer, 1995.

[3] I. M. Isaacs, *Finite Group Theory*, American Mathematical Society, 2008.

[4] B. E. Sagan, *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*, 2nd Edition, Springer, 2001.

[5] R. P. Stanley, *Enumerative Combinatorics, Volume 1*, 2nd Edition, Cambridge University Press, 2012.

[6] E. Adan-Bante and H. Verrill, "Symmetric groups and conjugacy classes," *Journal of Group Theory*, Vol. 11, No. 3, pp. 371–379, 2008.

[7] F. Bédard and A. Goupil, "The Poset of Conjugacy Classes and Decomposition of Products in the Symmetric Group," *Canadian Mathematical Bulletin*, 2018.

[8] Y. A. Neretin, "Algebras of conjugacy classes in symmetric groups," arXiv:1604.05755, 2016.

[9] G. James and A. Kerber, *The Representation Theory of the Symmetric Group*, Addison-Wesley, Reading, Massachusetts, 1981.

[10] W. Fulton, *Young Tableaux: With Applications to Representation Theory and Geometry*, Cambridge University Press, 1997.

[11] I. G. Macdonald, *Symmetric Functions and Hall Polynomials*, 2nd Edition, Oxford University Press, 1995.

[12] A. Kerber, *Representations of Permutation Groups I*, Springer-Verlag, 1971.

[13] P. J. Cameron, *Permutation Groups*, Cambridge University Press, 1999.

[14] J. D. Dixon and B. Mortimer, *Permutation Groups*, Springer-Verlag, 1996.

[15] V. Ivanov and S. Kerov, "The algebra of conjugacy classes in symmetric groups, and partial permutations," *Journal of Mathematical Sciences*, Vol. 96, pp. 3572–3599, 1999.

[16] A. Goupil and G. Schaeffer, "Factoring N-cycles and counting maps of genus zero," *European Journal of Combinatorics*, Vol. 24, No. 8, pp. 1135–1160, 2003.

[17] I. P. Goulden and D. M. Jackson, *Combinatorial Enumeration*, John Wiley & Sons, 1987.

[18] R. P. Stanley, *Enumerative Combinatorics, Volume 1*, 2nd Edition, Cambridge University Press, 2012.

[19] G. E. Andrews, *The Theory of Partitions*, Cambridge University Press, 1998.

[20] G. H. Hardy and S. Ramanujan, "Asymptotic formulae in combinatory analysis," *Proceedings of the London Mathematical Society*, Vol. 17, No. 1, pp. 75–115, 1918.

[21] P. Erdős and J. Lehner, "The distribution of the number of summands in the partitions of a positive integer," *Duke Mathematical Journal*, Vol. 8, No. 2, pp. 335–345, 1941.

[22] J. B. Olsson, *Combinatorics and Representations of Finite Groups*, Vorlesungen aus dem Mathematischen Institut der Universität Bonn, 1993.

[23] F. Bédard and A. Goupil, "The poset of conjugacy classes and decomposition of products in the symmetric group," *Canadian Mathematical Bulletin*, 2018.

[24] J. B. Olsson, "Sign conjugacy classes in symmetric groups," *Journal of Algebra*, Vol. 322, No. 8, pp. 2793–2800, 2009.

[25] Python Software Foundation, *Python Documentation*, Available at: https://docs.python.org/3/

[26] The Sage Developers, *SageMath Documentation*, Available at: https://doc.sagemath.org/