Technical Report: Political Representation Analysis of Nepal's House of Representatives.

Authors: Samikshya Dhamala

Email Sunway: samikshya.dhamala@mail.bcu.ac.uk

Email BCU: samikshya_25123833@sunway.edu.np

Date: 2025-06-06

Wordcount: 2841

Page count: 32

Confidential: Yes – Department only

Self-Assessment: Please highlight where you think your report grade should be. Example below.

Mark Range	Write simple algorithms using appropriate discrete data structures to solve computational problems (LO3)	Use appropriate methods to analyse the efficiency and correctness of algorithms (LO4)
Weight	25%	25%
0 – 29%	The algorithm does not solve an appropriate problem or has serious errors. There is little or no discussion of how the algorithm works. No discrete data structure has been used, or the choice of data structure is inappropriate.	The analysis is limited and seriously flawed.
30 – 39%	The algorithm solves part of an appropriate problem. There may be substantial aspects of the problem which are not attempted or explained, or errors in the solution. The explanation is unclear or missing important details about how the algorithm works.	An attempt has been made to analyse an algorithm, but appropriate methods of analysis were not used, and the results of the analysis may be incorrect or meaningless.
40 – 49%	A rudimentary algorithm solving a basic problem. There may be some errors which could be corrected with further work. There is a limited discussion of how the algorithm works. The choice of data structure is inappropriate, or unjustified.	An attempt has been made to measure the running time of the algorithm for some inputs, but the methodology is unclear or the measurement may be inaccurate. There is a limited discussion of some other issues relating to efficiency. Analysis of the algorithm's correctness is vague, or not attempted.
50 – 59%	The algorithm solves an appropriate problem, though it may have minor errors or fail to account for special cases. There is an explanation of how the algorithm works. The choice of data structure may be inappropriate or poorly justified.	The running time of the algorithm has been measured accurately for an appropriate range of inputs, and the methodology has been explained. There is some discussion of other issues relating to efficiency. There is a basic or informal analysis of the algorithm's correctness.
60 – 69%	The algorithm correctly solves an appropriate problem. There is a clear explanation of how the algorithm works. At least one appropriate data structure has been used, and the choice has been adequately justified.	The efficiency of the algorithm has been accurately measured using an appropriate methodology, which has been explained. The measurements may include more than one metric. There is an analysis of the algorithm's correctness, which may specify pre- and post-conditions for part of the algorithm.
70 – 79%	The algorithm correctly solves a challenging problem. There is a clear explanation of how the algorithm works, and the explanation makes clear references to the relevant parts of the source code. Appropriate data structures have been used, and justification is given for each with reference to the specific problem.	The efficiency of the algorithm has been accurately measured using an appropriate methodology, with multiple metrics and a clear explanation. The asymptotic complexity of the algorithm is given. The efficiency may be compared with appropriate alternative algorithm(s). There is a formal analysis of the correctness of at least part of the algorithm.
80 – 89%	A well-designed algorithm which correctly solves a challenging problem. There is a clear, detailed explanation of how the algorithm works, with clear references to the relevant parts of the source code. Appropriate data structures have been used, and justification is given for each with reference to the specific problem.	The efficiency of the algorithm has been accurately measured using an appropriate methodology, with multiple metrics and a clear, detailed explanation. The asymptotic complexity of the algorithm is given. The efficiency has been compared with appropriate alternative algorithm(s). There is a detailed formal analysis of the correctness of the algorithm.
90 – 100%	An excellent algorithm written, explained and evaluated to the highest standards.	An excellent analysis of the efficiency, complexity and correctness of an algorithm, conducted and explained to the highest standards.

Acknowledgement

I want to express my gratitude to Sunway College and BCU for providing me with this wonderful chance to deliver my technical report for the "Data Structure and Algorithms" module on the subject of "Political Representation Analysis of Nepal's House of Representatives." I ran into numerous problems while finishing the module, but my tutors were able to help me get past them with ease. I would like to thank them from the bottom of my heart for their help. Last but not least, the online resources I used to conduct my research and finish this assignment were quite beneficial.

Executive Summary

The technical report presents the Parliamentary Diversity Analysis System, through which one can determine a way to measure the representational diversity of electoral districts in Nepal by implementing basic data structures and algorithms.

To do that, it needs a custom Nepal class for data modeling, hash maps for grouping, merge sort for a diversity ranking of districts, and max heap to identify the most diverse regions. It calculates diversity scores by using set operations to find any diversity in gender, election type, or political party for the candidates in each district.

The performance evaluation indicates that the solution is very efficient and scalable. The total time complexity is O(C + D log D), where C is the total number of candidates and D is the number of districts; space complexity is O(C + D). Such metrics make the program efficient for both small and large datasets.

Table of Contents

1. Introduction

1.1 Political Diversity in Parliamentary Systems

In Nepal, the parliamentary system is practiced with representatives elected through different district and some from direct and proportional electoral systems. Every representative has a political party of their choice. The dominance of political parties within the House of Representatives might reflect general political trends, voting patterns, and representational fairness. Each district is visited and reported on to measure against inclusivity and diversity in the following respects:

• Gender diversity

• Election method diversity (Direct and Proportional)

• Political party diversity

1.2. Aims and Objectives

This report aims to:

- Parse and clean parliamentary data using object-oriented Python
- Implement algorithmic approaches to group and score districts
- Find the district with higher inclusivity
- Compare performance across algorithms

2. Section 1: Theory

2.1. Representation Metrics

Each district is rated in the range 0 to 3 based on the following factors:

- +1 if both Male and Female MPs are present
- +1 if both Direct and Proportional election types are present
- +1 in case MPs are present from more than one political party

2.2. Data Structures and Algorithms Used

2.2.1. Data Structure

1. Class

I use the Class data structure in Python to manage project data, keeping related information together. Represent each Member of Parliament (MP) with relevant attributes (name, district, gender, election, party). Class also Encapsulates data neatly. Promotes modularity and object-oriented design. Makes it easier to manipulate and access MP details consistently across functions.

2. Dictionary

A dictionary is a data structure that maps unique keys to corresponding values in order to organize or make data accessible. I used dictionaries to organize the data related to MPs from various districts of Nepal. Natural key-value structure suits grouping by district name. List-based grouping would require repeated scans (O(n)) to find district groups so I prefer to use dictionary.

3. Heap

A Heap is a complete binary tree data structure that satisfies the heap property: for every node, the value of its children is greater than or equal to its own value (Heap Data Structure, n.d.). Heaps are usually used to implement priority queues, where the smallest (or largest) element is always at the root of the tree. By inserting each district and its score as a tuple into the heap, the algorithm allows fast extraction of the top score and any other districts that share the same value. Max Heap is a balanced and scalable solution as compared with full sorting (O(n log n)) or linear max search (O(n)).

3. Priority Queue (Max Heap)

Priority queue is like a smart waiting line where instead of first-come-first-serve, people are served based on their importance or priority level. The thing about priority queue is that it automatically maintains order based on priority, so I don't need to sort the entire dataset every time I want to find top districts. Retrieve the top-k most inclusive districts. Since insertion and removal are easy in this I use this data structure.

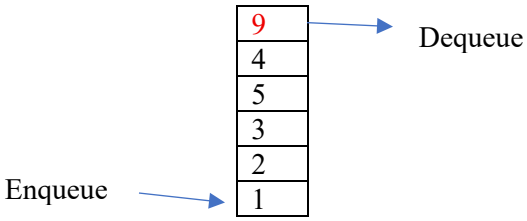

Fig1: Priority Queue

In addition to these, standard Python data structures like lists, sets, and array are used for various purposes.

2.2.2. Algorithms Use

1. Merge Sort

Among various Divide and Conquer sorting algorithms, Merge Sort has owned a wide range of applications, such sorting algorithm has a time complexity of O(n log n) and can work better than the Insertion sort in case of large arrays and it is even faster than the Quick Sort if the list is in reverse order (Paira et al., 2016).

Before choosing merge sort, I also looked at other sorting methods like bubble sort and quick sort. Bubble sort was too slow for my dataset size. Quick sort was fast but not stable, meaning it could change order of districts with same scores randomly. Some people might ask why not just use Python's built-in sort function. Actually, Python's built-in sort also uses merge sort principles, but implementing my own version helped me understand exactly how sorting works and gave me more control over the process.

2. Max Heap (Priority Queue)

Implementing max heap using Python's min heap with negative score conversion proved to be an elegant and efficient solution for finding top-performing districts in inclusivity analysis (Atkinson et al., 1986).

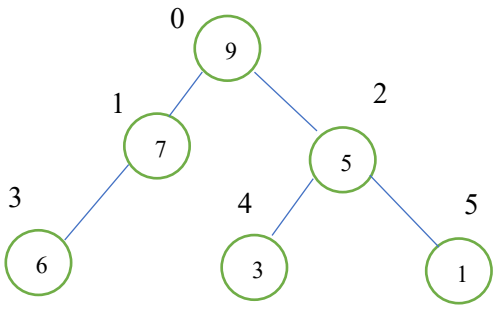

Fig3: Max Heap

Heap-based approach provides several performance advantages over alternative methods. Building the heap takes O(n) time where n is number of districts, which is same as linear search but provides better foundation for multiple operations. Finding top district takes O(log n) time per extraction, which becomes advantageous when finding multiple top districts. Max Heap provides a good balance between speed and flexibility. It also outperforms tree-based structures like Binary Search Trees (BSTs), which have more complex insertion/search logic and lack the same efficiency for repeated top-k access.

3. Hash Map grouping

What this hash map does is pretty straightforward: it takes each district and its score, and then groups all districts that share the same score together. So, if three districts have a score of 2, they all get put into one group under the label "2." This makes it super easy to see which districts have similar inclusivity levels. So overall, even if you have 10 or 100 or 1000 districts, the time it takes grows linearly. Compared to sorting or nested loops, hash maps offer better performance, cleaner logic, and easier data manipulation.

3. Section 2: Implementation

3.1. Problem Definition

The objective of this study is to find the inclusivity in districts in election based on gender diversity, election type variety and party representation. To find the inclusivity, a score is assigned to each district based on those factors. A higher the score the greater the inclusivity. The steps include grouping candidate by district, and then calculating the score, and structuring these districts using data structures algorithms like hash map, merge sort and max heap to find which districts have the more inclusivity.

Table 1. Problem notations

Symbol	Definition
d_i	The *i-th* district
c_j	The *j-th* candidate
C	Total number of candidates
D	Total number of districts
$g(c_j)$	Gender of candidate c_j
$e(c_j)$	Election type of candidate c_j
$p(c_j)$	Political party of candidate c_j
$G(d_i)$	Count of unique gender count in district d_i
$E(d_i)$	Count of unique election type count in district d_i
$P(d_i)$	Count of unique party count in district d_i
$S(d_i)$	Inclusivity score: $S(d_i) = G(d_i) + E(d_i) + P(d_i)$ (Max score = 3)

The algorithms' goal is to score the district with higher inclusivity and sort them, group them to find the district with high inclusivity score.

Table 2. General Steps of Algorithm

Input	**A list of candidates with name, gender, election type, party, and district attributes**
Output	A ranked list of districts based on the inclusivity score
Steps	1. Group candidates using a hash map by district 2. For each district, compute diversity scores $G(d_i)$,$E(d_i)$),$P(d_i)$ 3. Normalize or weight these scores to compute final $S(d_i)$ 4. Group districts by exact score using hash map 5. Apply Merge Sort to rank districts in descending order by score 6. Insert into a Max Heap and extract the top-k inclusive districts

3.2 Proposed Algorithms

This section introduced an advanced analysis and visualization tool for Nepal parliament house data (Nepal_house.csv).

nepal_house.csv (27.41 kB)

Detail | Compact | Column — 6 of 6 columns

nglishName	NepaliName	District	Election	Gender	party
277 unique values	277 unique values	Kathmandu 6%, Jhapa 4%, Other (247) 89%	Direct 60%, Proportion 40%	Male 66%, Female 34%	nepalicongress, NCP(UML), Other (110)
yut Prasad nali	अच्युत प्रसाद मैनाली	Bara	Direct	Male	NCP(UML)
y Kumar urasiya	अजय कुमार चौरसिया	Parsa	Direct	Male	nepalicongres
n Lal Modi	अमन लाल मोदी	Morang	Direct	Male	MoistCenter
ika Basnet	अम्बिका बस्नेत	Kathmandu	Proportion	Female	nepalicongres
ar Bahadur amajhi	अम्बर बहादुर रायमाझी	Udayapur	Direct	Male	NCP(UML)
ar Bahadur Thapa	अम्मर बहादुर थापा	Dailekh	Direct	Male	NCP(Unified Socialist)

Fig4: Snippet of a small portion of dataset and format

3.2.1. Proposed class to keep related information together

The code utilizes object-oriented programming (OOP) with "Nepal" class to encapsulate data attributes such as name, district, election type, gender and party. These attributes are vital for conducting detailed analyses and visualizations.

```
import csv
```

```
class Nepal:
  def __init__(self,name,district,election,gender,party):
    self.name=name
    self.district=district
    self.election=election
    self.gender=gender
    self.party=party

  def __str__(self):
        return f"Name: {self.name}, District: {self.district}, Election type: {self.election}, Gender:{self.gender}, and party name:{self.party}"

  def __repr__(self):
        return self.__str__()
```

Fig5: Code snippet of Nepal Class

3.2.2. Reading Data from CSV file

Here for this portion, read_mps_from_csv(filename) is designed to read data from a CSV file and convert it into a mp's list. It iterates through each row to create Nepal objects based on the data. Then initializes an empty list 'mps' where objects will be stored after being read.

```python
def read_mps_from_csv(filename):
    mps = []
    with open(filename, 'r') as file:
        next(file)  # Skip header
        for line in file:
            parts = line.strip().split(',')
            if len(parts) == 6:
                english_name = parts[0]
                # NepaliName = parts[1]  ← we skip this!
                district = parts[2]
                election = parts[4]
                gender = parts[3]
                party = parts[5] if parts[5] else "Independent"  # in case some are blank

                mps.append(Nepal(english_name, district, gender, election, party))
    return mps
```

```python
[8] filename = 'nepal_house.csv'
    mp_list = read_mps_from_csv(filename)

    for item in mp_list:
        print(item)
```

Fig6: Code Snippet of Reading CSV file code

3.2.3. Grouping and giving score to each district

For this first I created an empty dictionary “district_map” and loop through MP list to append on the dictionary where district name is a key and value is the list of MP’s from that district. And created another empty dictionary ‘disctrict_scores’ to store district with their score.

```python
def group_by_district(mp_list):
  district_map = {}
  for mp in mp_list:
        if mp.district not in district_map:
            district_map[mp.district] = []
        district_map[mp.district].append(mp)
  return district_map

district_map = group_by_district(mp_list)

# Now print the grouped result
print("\nDistrict-wise MP list:")
for district, mps in district_map.items():
    print(f"\nDistrict: {district}")
    for mp in mps:
        print(f"  {mp}")
```

Fig7: Code snippet of grouping MP’s

```
district_scores = {}
for district, mps in district_map.items():
    genders = set()
    elections = set()
    parties = set()
    for mp in mps:
        genders.add(mp.gender)
        elections.add(mp.election)
        if mp.party:  # avoid None
            parties.add(mp.party)

    score = 0
    if len(genders) > 1: score += 1
    if len(elections) > 1: score += 1
    if len(parties) > 1: score += 1
    district_scores[district] = score
    print(f"{district}: {score}")
```

Fig8: Code snippet of giving score to each district

3.2.4. Applying userinput

Once the program has determined each district's inclusion score based on diversity parameters (gender, election method, and political parties), the user can interactively query the score of any district that draws their interest.

```
user_input = input("Enter a district name to check its inclusion score: ").strip()

# Case-insensitive match
matched = None
for district in district_scores:
    if district.lower() == user_input.lower():
        matched = district
        break

if matched:
    print(f"Inclusion score for '{matched}': {district_scores[matched]}")
else:
    print(f" District '{user_input}' not found in the data.")
```

```
Enter a district name to check its inclusion score:
```

3.2.5. Applying merge sort

In my code, I create two functions - merge_sort and merge. The merge_sort function does the dividing part and merge function does the combining part. I also use something called lambda function which helps me specify what to sort by. In my case, I want to sort districts by their inclusivity scores.

So when I write key=lambda x: x[1], I am telling merge sort to look at the score part of each district-score pair and use that for sorting. This flexibility is really helpful because sometimes I might want to sort by district names instead of scores .

The merge function is where the real magic happens. It takes two sorted lists and combines them into one sorted list by comparing elements one by one.

```
def merge_sort(arr, key=lambda x: x):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid], key)
    right = merge_sort(arr[mid:], key)
    return merge(left, right, key)

def merge(left, right, key):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(right[j]):
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged
```

```
[14] # Print results
sorted_merge = merge_sort(list(district_scores.items()), key=lambda x: x[1])
print("Sorted with Merge Sort (Stable):")
for district, score in sorted_merge:
    print(f"{district}: {score}")
```

Fig9: Code snippet of Merge Sort

3.2.6. Applying Hash map grouping

The hash_map_grouping function works like organizing students into groups based on their test scores. Just like how you might put all students who scored 90 together and all students who scored 85 together, this code puts districts with the same scores into the same group.

When I first looked at this code, I was curious about how it handles the grouping process. The function takes a collection of districts and their scores, then creates separate groups for each unique score. This is similar to how we might organize data in real life - like grouping products by price range or students by grade levels.

```
[15] def hash_map_grouping(scores):
         """Group districts by score, return highest scoring group"""
         score_groups = {}
         # max_score = 0

         for district, score in district_scores.items():
             if score not in score_groups:
                 score_groups[score] = []
             score_groups[score].append(district)
             # max_score = max(max_score, score)

         return score_groups#[max_score], max_score

     all_groups = hash_map_grouping(district_scores)
     for score, districts in all_groups.items():
         print(f"Score {score}: {districts}")
```

Fig10: Code snippet of Hash map grouping

3.2.7. Applying Max heap (priority queue)

The next step was to design a Max Heap data structure for the candidate districts from the ground up, thus avoiding the use of Python's built-in libraries. The implemented MaxHeap class will have insert, pop_max, and internal sift-up/sift-down operations that are battle-tested. This will allow fast retrieval for the highest-scoring district in O(log n) time.

But my implementation goes beyond just finding single top district. Since multiple districts might have same highest score, I continue popping elements from heap as long as they have the same score as the first element. This ensures I capture all districts that share the maximum inclusivity score, which is crucial for fair analysis.

```python
class MaxHeap:
    def __init__(self):
        self.data = []

    def insert(self, item):
        self.data.append(item)
        self._sift_up(len(self.data) - 1)

    def pop_max(self):
        if len(self.data) == 0:
            return None
        self._swap(0, len(self.data) - 1)
        max_item = self.data.pop()
        self._sift_down(0)
        return max_item

    def _sift_up(self, idx):
        parent = (idx - 1) // 2
        while idx > 0 and self.data[idx][0] > self.data[parent][0]:
            self._swap(idx, parent)
            idx = parent
            parent = (idx - 1) // 2

    def _sift_down(self, idx):
        child = 2 * idx + 1
        while child < len(self.data):
            right = child + 1
            if right < len(self.data) and self.data[right][0] > self.data[child][0]:
                child = right
            if self.data[idx][0] >= self.data[child][0]:
                break
            self._swap(idx, child)
            idx = child
            child = 2 * idx + 1

    def _swap(self, i, j):
        self.data[i], self.data[j] = self.data[j], self.data[i]

    def is_empty(self):
        return len(self.data) == 0
```

```
[21] heap = MaxHeap()
     for district, score in district_scores.items():
         heap.insert((score, district))

     top_score, top_district = heap.pop_max()
     most_inclusive = [top_district]

     while not heap.is_empty():
         score, district = heap.pop_max()
         if score == top_score:
             most_inclusive.append(district)
         else:
             break

     print(f"Most inclusive districts (Score {top_score}):")
     for district in most_inclusive:
         print(f"- {district}")
```

Fig11: Code snippet of Max heap

4. Section 3: Performance Evaluation

4.1. Time Complexity

Table 3. Time complexity

Function / Block	Description	Time Complexity
Nepal.init()	MP object creation	O(1)
read_mps_from_csv()	Read and parse C lines from CSV	O(C)
group_by_district()	Group MPs by district	O(C)
Diversity Score Calculation	Per district, calculate diversity score	O(C)
merge_sort()	Sort D districts by score	O(D log D)
hash_map_grouping()	Group districts by score	O(D)
MaxHeap	Insert all districts and find the top k	O(D log D)

Where C is total numbers of MP and D is the total numbers of districts.

Overall time complexity: *O(C+DlogD)*

Since usually C>D, the total time is linear in number of candidates and log-linear in number of districts.

4.2. Space Complexity

Table 4. Space complexity

Structure	Purpose	Space Complexity
mp_list	All C candidates	O(C)
district_map	Grouping C candidates into D districts	O(C + D)
district_scores	Score per district	O(D)
sorted_merge	Merge sort storage	O(D)
score_groups	Group districts by score	O(D)
MaxHeap	Heap for districts	O(D)

Where C is total numbers of MP and D is the total numbers of districts.

Total Space Complexity

O(C+D), Space grows linearly with the number of candidates and districts.

4.3. Comparing the best algorithms

Since your goal is only to find the most inclusive district(s) based on score, the best algorithm is your custom-built MaxHeap,

- Avoids full sorting (unnecessary overhead)
- Handles ties naturally
- Is efficient (O(D log D))
- Fastest, cleanest method for categorization
- Max Heap directly solves this problem with the right balance of performance and capability
- Need to extract all districts with the highest score → Max Heap handles this naturally

Table 5.Algorithum comparison

Algorithm	Time Complexity	Why Max Heap Is Better
Max Heap	O(DlogD)	Directly retrieves the top-scoring district(s) efficiently; handles ties automatically.
Merge Sort	O(DlogD)	Requires full sorting even if only the top score is needed; less efficient for this task.
Hash Map Grouping	O(D)	Excellent for organizing data, but cannot determine top score or rank districts alone.

4.4. Data Integrity and Null Handling

Data integrity is done to remove Nepali name from the dataset as it is not required and in empty value in party column is replaced with independent party.

4.5. Assertion Table:

Table 6. Assertion table of project

S. N	Description	Output(Working or not)

1	Verifies the output of read_mps_from_csv is a list
	`def test_read_mps_from_csv(filename):`
	`  print("Running assertion tests for: read_mps_from_csv")`
	`  mps = read_mps_from_csv(filename)`
	`  expected_type = list`
	`  actual_type = type(mps)`
	`  print(f"Expected type: {expected_type}, Actual type: {actual_type}")`
	`  assert actual_type == expected_type, "Failed: Output is not a list."`
		Working
		`Running assertion tests for: read_mps_from_csv`
		`Expected type: <class 'list'>, Actual type: <class 'list'>`

2	Ensures the MP list is not empty
	`expected_non_empty = True`
	`actual_non_empty = len(mps) > 0`
	`print(f"Expected non-empty list: {expected_non_empty}, Actual: {actual_non_empty}")`
	`assert actual_non_empty, "Failed: MP list is empty."`
		Working
		`Expected non-empty list: True, Actual: True`

3	Runs 5 times for each attribute and check if MP has all different attributes or not
	`required_attrs = ['name', 'district', 'election', 'gender', 'party']`
	`for attr in required_attrs:`
	`  print(f"Checking if '{attr}' exists in first MP...")`
	`  assert hasattr(mps[0], attr), f"Failed: Missing attribute '{attr}' in Nepal object."`
	`print("All assertions passed.")`
		Working
		`Checking if 'name' exists in first MP...`
		`Checking if 'district' exists in first MP...`
		`Checking if 'election' exists in first MP...`
		`Checking if 'gender' exists in first MP...`
		`Checking if 'party' exists in first MP...`
		`All assertions passed.`

4	Checks that for each score, the grouped districts are stored in a list data structure.
	`def assert_hash_map_grouping(district_scores, grouped_scores):`
	`  try:`
	`  # Check each group is a list and non-empty`
	`  for score, districts in grouped_scores.items():`
	`  assert isinstance(districts, list), f"Districts for score {score} should be a list"`
	`  assert len(districts) > 0, f"Districts list for score {score} should not be empty"`
	`  print("All groups are lists and non-empty.")`
	`  except AssertionError as e:`
	`  print(f"Assertion failed: {e}")`
		Working
		`All groups are lists and non-empty.`

5	Ensures that no group of districts is empty for any score key.
	`def assert_hash_map_grouping(district_scores, grouped_scores):`
	`  try:`
	`  # Check each group is a list and non-empty`
	`  for score, districts in grouped_scores.items():`
	`  assert isinstance(districts, list), f"Districts for score {score} should be a list"`
	`  assert len(districts) > 0, f"Districts list for score {score} should not be empty"`
	`  print("All groups are lists and non-empty.")`
	`  except AssertionError as e:`
	`  print(f"Assertion failed: {e}")`
		Working
		`All groups are lists and non-empty.`

6	Ensures the heap is not empty before popping the maximum element.
	`def assert_maxheap_top_districts(district_scores, heap, top_score, top_district, most_inclusive):`
	`  try:`
	`  assert not heap.is_empty(), "Heap should not be empty before popping max"`
	`  print("Heap is not empty before popping max.")`
	`  except AssertionError as e:`
	`  print(f"Assertion failed: {e}")`
		Working
		`Heap is not empty before popping max.`

7	Checks that the top score popped from the heap is numeric.
	`try:`
	`  assert isinstance(top_score, (int, float)), "Top score should be a number"`
	`  print("Top score is a number.")`
	`except AssertionError as e:`
	`  print(f"Assertion failed: {e}")`
		Working
		`Top score is a number.`

8	Verifies all districts in the most inclusive list have the exact top score value.
	`try:`
	`  for district in most_inclusive:`
	`  assert district_scores[district] == top_score, f"District {district} does not have the top score {top_score}"`
	`  print("All districts in most inclusive list have the top score.")`
	`except AssertionError as e:`
	`  print(f"Assertion failed: {e}")`
		Working
		`All districts in most inclusive list have the top score.`

5.Conclusion

This report presents three algorithmic strategies to measure political diversity in Nepal's parliament. Each method achieves the same goal with distinct trade-offs in complexity, performance, and extensibility. From which Max heap turns out to be very effective to solve this particular problem, Future enhancements may include caste, age, or ethnic diversity, or visualization with libraries like matplotlib. These insights offer valuable knowledge for policymakers and stakeholders aiming to improve well-being globally.

6.References

Alake, R. (2022) *Merge Sort Explained: A Data Scientist's Algorithm Guide*. Available at: https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/.

Amos, D. (2023) *Object-Oriented programming (OOP) in python 3 – real python*. Available at: https://realpython.com/python3-object-oriented-programming/.

An Introduction to Tree in Data Structure (n.d.). Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/trees-in-data-structure.

Applications, Advantages and Disadvantages of Binary Search Tree (2022). Available at: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/.

Atkinson, M.D., Sack, J.-R. ., Santoro, N., et al. (1986) Min-max heaps and generalized priority queues. *Communications of the ACM*, 29 (10): 996–1000. doi:https://doi.org/10.1145/6617.6621.

Paira, S., Chandra, S. and Alam, S.K.S. (2016) Enhanced Merge Sort- A New Approach to the Merging Process. *Procedia Computer Science*, 93: 982–987. doi:https://doi.org/10.1016/j.procs.2016.07.292.

W3schools (2018) *Python Dictionaries*. Available at: https://www.w3schools.com/python/python_dictionaries.asp.

Heap Data Structure (n.d.). Available at: https://www.geeksforgeeks.org/heap-data-structure/.

Git hub link for the code : https://github.com/samikshyadhamala/DSA

7. Appendix: Code and Output

```
[ ] import csv
```

```
[ ] class Nepal:
      def __init__(self,name,district,election,gender,party):
        self.name=name
        self.district=district
        self.election=election
        self.gender=gender
        self.party=party

      def __str__(self):
            return f"Name: {self.name}, District: {self.district}, Election type: {self.election}, Gender:{self.gender}, and party name:{self.party}"

      def __repr__(self):
            return self.__str__()
```

Read Data from the dataset

```
def read_mps_from_csv(filename):
    mps = []
    with open(filename, 'r') as file:
        next(file)  # Skip header
        for line in file:
            parts = line.strip().split(',')
            if len(parts) == 6:
                english_name = parts[0]
                # NepaliName = parts[1]  ← we skip this!
                district = parts[2]
                election = parts[4]
                gender = parts[3]
                party = parts[5] if parts[5] else "Independent"  # in case some are blank

                mps.append(Nepal(english_name, district, gender, election, party))
    return mps
```

```
[ ] filename = 'nepal_house.csv'
    mp_list = read_mps_from_csv(filename)

    for item in mp_list:
        print(item)
```

```
Name: Abdul Khan, District: Bardiya, Election type: Proportion, Gender:Male, and party name:janmatparty
Name: Achyut Prasad Mainali, District: Bara, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Ajay Kumar Chaurasiya, District: Parsa, Election type: Direct, Gender:Male, and party name:nepalicongress
Name: Aman Lal Modi, District: Morang, Election type: Direct, Gender:Male, and party name:MoistCenter
Name: Ambika Basnet, District: Kathmandu, Election type: Proportion, Gender:Female, and party name:nepalicongress
Name: Ammar Bahadur Rayamajhi, District: Udayapur, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Ammar Bahadur Thapa, District: Dailekh, Election type: Direct, Gender:Male, and party name:NCP(Unified Socialist)
Name: Amrit Lal Rajbanshi, District: Jhapa, Election type: Proportion, Gender:Male, and party name:NCP(UML)
Name: Amrita Devi Agrahari, District: Kapilbastu, Election type: Proportion, Gender:Female, and party name:NCP(UML)
Name: Anisha Nepali, District: Salyan, Election type: Proportion, Gender:Female, and party name:RastriyaPrajatantraParty
Name: Anita Devi, District: Dhanusa, Election type: Proportion, Gender:Female, and party name:janmatparty
Name: Anjani Shrestha, District: Chitwan, Election type: Proportion, Gender:Female, and party name:nepalicongress
Name: Arjun Narsingh K.C., District: Nuwakot, Election type: Direct, Gender:Male, and party name:nepalicongress
Name: Arun Kumar Chaudhary, District: Kailali, Election type: Direct, Gender:Male, and party name:nagrikUnmuktiparty
Name: Asha B.K., District: Banke, Election type: Proportion, Gender:Female, and party name:nepalicongress
Name: Ashim Shah, District: Kathmandu, Election type: Proportion, Gender:Male, and party name:NationalindependentParty
Name: Ashma Kumari Chaudhary, District: Dang, Election type: Proportion, Gender:Female, and party name:NCP(UML)
Name: Ashok Kumar Chaudhary, District: Sunsari, Election type: Proportion, Gender:Male, and party name:NationalindependentParty
Name: Ashok Kumar Rai, District: Sunsari, Election type: Direct, Gender:Male, and party name:JanataSamajbadiparty
Name: Badri Prasad Pandey, District: Bajura, Election type: Direct, Gender:Male, and party name:nepalicongress
Name: Balaram Adhikari, District: Kapilbastu, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Barshaman Pun, District: Rolpa, Election type: Direct, Gender:Male, and party name:MoistCenter
Name: Basant Kumar Nemwang, District: Panchthar, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Basudev Ghimire, District: Rupandehi, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Bhagbati Chaudhary, District: Sunsari, Election type: Direct, Gender:Female, and party name:NCP(UML)
Name: Bhanubhakta Joshi, District: Bajhang, Election type: Direct, Gender:Male, and party name:NCP(Unified Socialist)
Name: Bhim Prasad Acharya, District: Sunsari, Election type: Direct, Gender:Male, and party name:NCP(UML)
Name: Bidhya Bhattarai, District: Kaski, Election type: Direct, Gender:Female, and party name:NCP(UML)
Name: Bijula Rayamajhi, District: Arghakhanchi, Election type: Proportion, Gender:Female, and party name:NCP(UML)
Name: Bikram Pandey, District: Chitwan, Election type: Direct, Gender:Male, and party name:RastriyaPrajatantraParty
Name: Bimala Subedi, District: Nuwakot, Election type: Proportion, Gender:Female, and party name:MoistCenter
Name: Bimalendra Nidhi, District: Dhanusa, Election type: Proportion, Gender:Male, and party name:nepalicongress
Name: Bina Devi, District: Rautahat, Election type: Proportion, Gender:Female, and party name:NCP(UML)
Name: Bina Jaiswal, District: Parsa, Election type: Proportion, Gender:Female, and party name:RastriyaPrajatantraParty
```

```python
def group_by_district(mp_list):
  district_map = {}
  for mp in mp_list:
        if mp.district not in district_map:
            district_map[mp.district] = []
        district_map[mp.district].append(mp)
  return district_map

district_map = group_by_district(mp_list)

# Now print the grouped result
print("\nDistrict-wise MP list:")
for district, mps in district_map.items():
    print(f"\nDistrict: {district}")
    for mp in mps:
        print(f"  {mp}")
```

```
District-wise MP list:

District: Bardiya
  Name: Abdul Khan, District: Bardiya, Election type: Proportion, Gender:Male, and party name:janmatparty
  Name: Gita Basnet, District: Bardiya, Election type: Proportion, Gender:Female, and party name:RastriyaPrajatantraParty
  Name: Jabeda Khatun Jaga, District: Bardiya, Election type: Proportion, Gender:Female, and party name:nepalicongress
  Name: Lalbir Chaudhary, District: Bardiya, Election type: Direct, Gender:Male, and party name:nagrikUnmuktiparty
  Name: Sanjaya Kumar Gautam, District: Bardiya, Election type: Direct, Gender:Male, and party name:nepalicongress
  Name: Shanti Chaudhary, District: Bardiya, Election type: Proportion, Gender:Female, and party name:NCP(UML)
  Name: Shova Gyawali, District: Bardiya, Election type: Proportion, Gender:Female, and party name:NCP(UML)

District: Bara
  Name: Achyut Prasad Mainali, District: Bara, Election type: Direct, Gender:Male, and party name:NCP(UML)
  Name: Jwala Kumari Shah, District: Bara, Election type: Direct, Gender:Female, and party name:NCP(UML)
  Name: Krishna Kumar Shrestha, District: Bara, Election type: Direct, Gender:Male, and party name:NCP(Unified Socialist)
  Name: Umaravati Devi , District: Bara, Election type: Proportion, Gender:Female, and party name:MoistCenter
  Name: Upendra Yadav, District: Bara, Election type: Direct, Gender:Male, and party name:SamajbadiParty

District: Parsa
  Name: Ajay Kumar Chaurasiya, District: Parsa, Election type: Direct, Gender:Male, and party name:nepalicongress
  Name: Bina Jaiswal, District: Parsa, Election type: Proportion, Gender:Female, and party name:RastriyaPrajatantraParty
  Name: Nirmala Koirala, District: Parsa, Election type: Proportion, Gender:Female, and party name:NCP(UML)
  Name: Prabhu Hajara Dushadh, District: Parsa, Election type: Proportion, Gender:Male, and party name:NCP(UML)
  Name: Pradip Yadav, District: Parsa, Election type: Direct, Gender:Male, and party name:JanataSamajbadiparty
  Name: Raj Kumar Gupta, District: Parsa, Election type: Direct, Gender:Male, and party name:NCP(UML)
  Name: Ramesh Rijal, District: Parsa, Election type: Direct, Gender:Male, and party name:nepalicongress

District: Morang
  Name: Aman Lal Modi, District: Morang, Election type: Direct, Gender:Male, and party name:MoistCenter
  Name: Dig Bahadur Limbu, District: Morang, Election type: Direct, Gender:Male, and party name:nepalicongress
```

```
[10] district_scores = {}
     for district, mps in district_map.items():
         genders = set()
         elections = set()
         parties = set()
         for mp in mps:
             genders.add(mp.gender)
             elections.add(mp.election)
             if mp.party:  # avoid None
                 parties.add(mp.party)

         score = 0
         if len(genders) > 1: score += 1
         if len(elections) > 1: score += 1
         if len(parties) > 1: score += 1
         district_scores[district] = score
         print(f"{district}: {score}")
```

```
Panchthar: 0
Rupandehi: 3
Bajhang: 0
Kaski: 3
Arghakhanchi: 3
Rautahat: 3
Nawalparasi (East of Bardaghat Susta): 2
Dhading: 3
Makawanpur: 3
Jumla: 3
Siraha: 3
Nawalparasi: 3
Sarlahi: 3
Lalitpur: 3
Rukum West: 3
Humla: 0
Baglung: 1
Baitadi: 0
Dolpa: 0
Syangja: 0
```

```
user_input = input("Enter a district name to check its inclusion score: ").strip()

# Case-insensitive match
matched = None
for district in district_scores:
    if district.lower() == user_input.lower():
        matched = district
        break

if matched:
    print(f"Inclusion score for '{matched}': {district_scores[matched]}")
else:
    print(f" District '{user_input}' not found in the data.")
```

```
Enter a district name to check its inclusion score:
```

```
[11] def merge_sort(arr, key=lambda x: x):
         if len(arr) <= 1:
             return arr
         mid = len(arr) // 2
         left = merge_sort(arr[:mid], key)
         right = merge_sort(arr[mid:], key)
         return merge(left, right, key)

     def merge(left, right, key):
         merged = []
         i = j = 0
         while i < len(left) and j < len(right):
             if key(left[i]) <= key(right[j]):
                 merged.append(left[i])
                 i += 1
             else:
                 merged.append(right[j])
                 j += 1
         merged.extend(left[i:])
         merged.extend(right[j:])
         return merged
```

```
# Print results
sorted_merge = merge_sort(list(district_scores.items()), key=lambda x: x[1])
print("Sorted with Merge Sort (Stable):")
for district, score in sorted_merge:
    print(f"{district}: {score}")
```

```
Okhaldhunga: 0
Terhathum: 0
Pyuthan: 0
Manang: 0
Mugu: 0
Mustang: 0
Baglung: 1
Gulmi: 1
Tanahun: 1
Kanchanpur: 1
Achham: 1
Bajura: 2
Nawalparasi (East of Bardaghat Susta): 2
Dadeldhura: 2
Gorkha: 2
Dolakha: 2
Sindhupalchowk: 2
Bardiya: 3
Bara: 3
Parsa: 3
Morang: 3
Kathmandu: 3
Udayapur: 3
Dailekh: 3
Jhapa: 3
Kapilbastu: 3
```

```python
def hash_map_grouping(scores):
    """Group districts by score, return highest scoring group"""
    score_groups = {}
    # max_score = 0

    for district, score in district_scores.items():
        if score not in score_groups:
            score_groups[score] = []
        score_groups[score].append(district)
        # max_score = max(max_score, score)

    return score_groups#[max_score], max_score

all_groups = hash_map_grouping(district_scores)
for score, districts in all_groups.items():
    print(f"Score {score}: {districts}")
```

```
Score 3: ['Bardiya', 'Bara', 'Parsa', 'Morang', 'Kathmandu', 'Udayapur', 'Dailekh', 'Jhapa', 'Kapilbastu', 'Salyan', 'Dhanusa', 'Chitwan
Score 2: ['Bajura', 'Nawalparasi (East of Bardaghat Susta)', 'Dadeldhura', 'Gorkha', 'Dolakha', 'Sindhupalchowk']
Score 0: ['Panchthar', 'Bajhang', 'Humla', 'Baitadi', 'Dolpa', 'Syangja', 'Darchula', 'Sankhubasabha', 'Surkhet', 'Myagdi', 'Kalikot', '
Score 1: ['Baglung', 'Gulmi', 'Tanahun', 'Kanchanpur', 'Achham']
```

```python
class MaxHeap:
    def __init__(self):
        self.data = []

    def insert(self, item):
        self.data.append(item)
        self._sift_up(len(self.data) - 1)

    def pop_max(self):
        if len(self.data) == 0:
            return None
        self._swap(0, len(self.data) - 1)
        max_item = self.data.pop()
        self._sift_down(0)
        return max_item

    def _sift_up(self, idx):
        parent = (idx - 1) // 2
        while idx > 0 and self.data[idx][0] > self.data[parent][0]:
            self._swap(idx, parent)
            idx = parent
            parent = (idx - 1) // 2

    def _sift_down(self, idx):
        child = 2 * idx + 1
        while child < len(self.data):
            right = child + 1
            if right < len(self.data) and self.data[right][0] > self.data[child][0]:
                child = right
            if self.data[idx][0] >= self.data[child][0]:
                break
            self._swap(idx, child)
            idx = child
            child = 2 * idx + 1

    def _swap(self, i, j):
        self.data[i], self.data[j] = self.data[j], self.data[i]

    def is_empty(self):
        return len(self.data) == 0
```

```
[15] heap = MaxHeap()
     for district, score in district_scores.items():
         heap.insert((score, district))

     top_score, top_district = heap.pop_max()
     most_inclusive = [top_district]

     while not heap.is_empty():
         score, district = heap.pop_max()
         if score == top_score:
             most_inclusive.append(district)
         else:
             break

     print(f"Most inclusive districts (Score {top_score}):")
     for district in most_inclusive:
         print(f"- {district}")
```

```
Most inclusive districts (Score 3):
- Bardiya
- Bara
- Morang
- Jhapa
- Dang
- Sarlahi
- Lalitpur
- Lamjung
- Sunsari
- Rukum West
- Khotang
- Kapilbastu
- Rolpa
- Kathmandu
- Salyan
- Saptari
- Rupandehi
- Dhanusa
- Jajarkot
- Kaski
- Bhojpur
- Bhaktapur
- Sindhuli
- Rukum East
- Parsa
- Udayapur
- Chitwan
- Arghakhanchi
- Rautahat
- Kavrepalanchowk
- Mahottari
```

8. Appendix: Math for complexity calculation

1. Group by district

Logic:

- Insert each MP into a dictionary: O(1) per insertion (average case)
- Done for C(total no) MPs

Time Complexity:

- O(C)(one pass over all MPs)

2. Diversity Score Calculation

Logic:

- For each district, process its list of MPs
- All MPs are covered exactly once

Time Complexity:

- O(C)(total MPs across all districts)

3. Merge Sort (Custom Implementation)

Logic:

Let D = len(district).

- Each call splits the array in half → two subproblems of size D/2.
- Merging takes O(D) time (comparing and copying elements).

This gives us the recurrence:

- $T(D)=2T(D/2)+O(D)$

Solving Using Master Theorem:

For:

- $T(n)=aT(n/b)+O(n^d)$

We have:

a=2, b=2, d=1

Compare:

$n^{\log_b a}=n^{\log_2 2}=n^1 \Rightarrow d=\log_b a \Rightarrow$

Case 2 of Master Theorem$\Rightarrow T(n)=O(n\log n)$

Final:

So, O(DlogD) time complexity for Merge Sort

4. Max Heap (Custom Implementation)

- Inserting all D districts into a Max Heap as (score, district) tuples.
- Using pop_max() repeatedly to extract the highest-scoring districts.

Let:

- D = total number of districts (i.e., len(district_scores))

Each insert() operation calls _sift_up().

- In the worst case, _sift_up() travels from a leaf to the root → height of heap = $\log_2 D$
- So one insert:

$$T_{insert}(1)=O(\log D)$$

Total inserts for D districts:

$$T_{insert}(D)=O(\log 1+\log 2+\cdots+\log D)=O(D\log D)$$

Each pop_max() operation calls:

- _swap() → O(1)
- _sift_down() → worst case O(log D)

Suppose you extract K districts with the same top score (worst case: K = D

Then:

$T_{pop}(K)=O(K\log D)$

In the worst case, all districts have the same score:

$$T_{pop}(D)=O(D\log D)$$

Final Answer:

$$T(D)=O(D\log D)$$

Where:

- D is the number of districts
- All operations scale logarithmically due to the heap property

5. Hash map grouping

Each operation:

- if score not in score_groups: → average-case O(1)

- score_groups[score] = [] → O(1)
- score_groups[score].append(district) → amortized O(1)

These operations happen once per district.

Time (in total for all districts):

O(D)

Total Time Complexity: O(D)