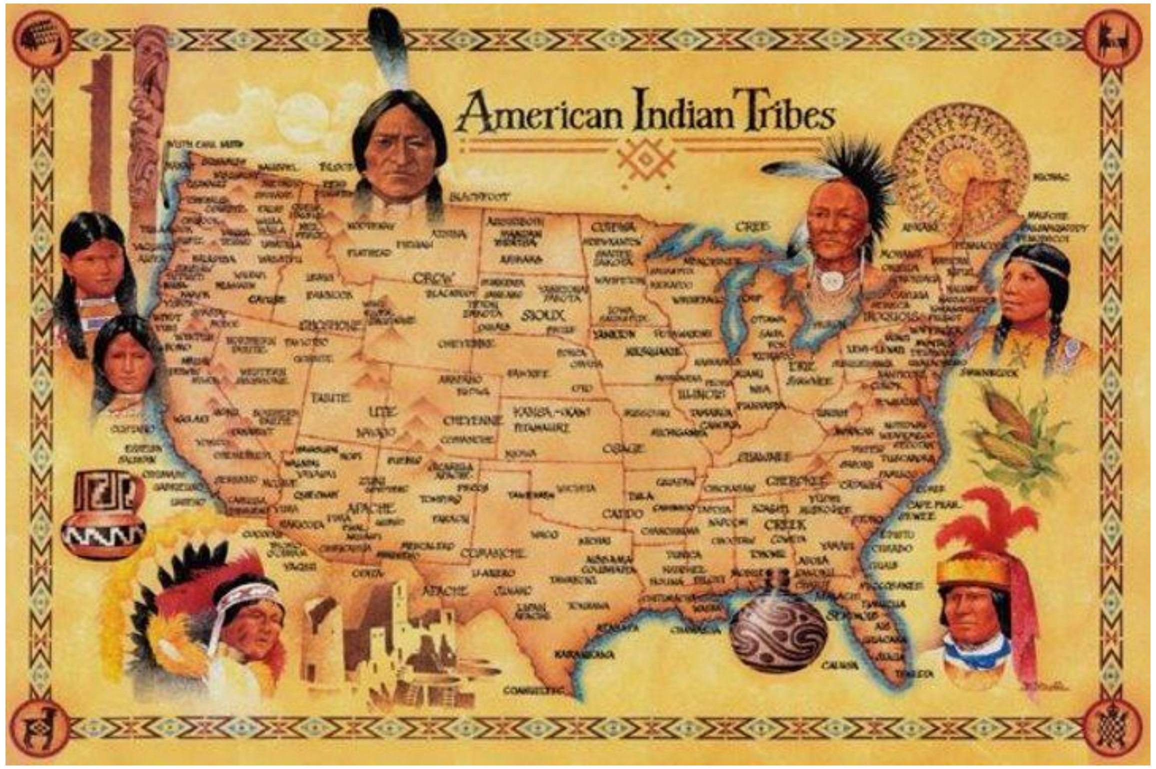
# COMP-305 ALGORITHMS&COMPLEXITY ROOM-22 PROJECT PRESENTATION

BERKE CAN RIZAI(69282)
EMRE DÜZAKIN(69663)
EREN BARIŞ BOSTANCI(68770)
BARIŞ KAPLAN(69054)

# TABLE OF CONTENTS

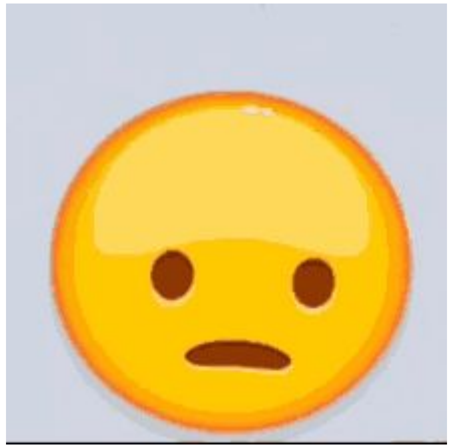


## Our Problem

There is an election for a shaman king, each tribe will vote and select a number of electors to vote on the main election. Since each tribe has different amount of tribal votes and populations vary, the minimum population to win the election is not 50% + 1 of whole population, we need to find the minimum number.

## Our Algorithm

We approached this problem in a greedy but complete way. First, we listed the tribes given in the file in accordance with their population (from lowest population to the highest) and delegate value (computed by dividing the delegate count with the total population). We created two priority queues and used these values as keys.



## Execution

We started popping from the delegate value queue one by one. This
ensures that we always count the most valuable population votes first.After popping, we add the
delegate count of the poped tribe to the variable current_delegates, and the population needed t
o win the tribes delegate to another variable current_voters (%50+1 of the total population,
since we are aiming for the minimum). We also remove this popped tribe from the second list
where we hold the tribes by their popluation count. This approach is mostly true by it self, but
there is an edge case that need handling.

## Edge Case

When we pick the last tribe from the delegate value queue, there is a chance of us exceeding the required elector votes. For example, if the elector votes required to win the election is 10, we can get 11 electors if the number of electors in the most valuable tribes is 4, 4, and 3 respectively. We are still picking the smallest number of population votes to reach 11 electors, but since the last elector is not required to win the election we might be getting a non-optimal solution. If there is another tribe with 2 elector votes and a population less then the one that has 3, picking it would be more sufficient.

## Handling of The Edge Case

If the total number of electors we pick surpasses the amount that is required to win the election, we start checking the population queue. As a remark, the population queue lists the tribes from least populous to most populous. We pop an element from the population queue and check if its delegate count added to the current delegate count is sufficient to win the election. If that's the case, we pick this tribe instead of the most valuable option. If not we pop another tribe from the population queue and add it with our first entry. We keep doing this until the total delegate count for the tribes we pop from the population queue equals the one we popped from the delegate value queue, or the total population of these tribes passes the one we popped from the delegate value queue.

## Our Choices of Data Structures

2 Priority Queues to list the tribes in accordance with two values, the delegate value (delegate_count/population) and tribes population

Array lists to hold the raw data read from the input file.

## The Reasons of Our Data Structure Choices

Since we can remove the elements in the priority queue with poll() and the remove() methods in constant time for each call, it is efficient to use the priority queues.

# Time Complexity of Our Algorithm

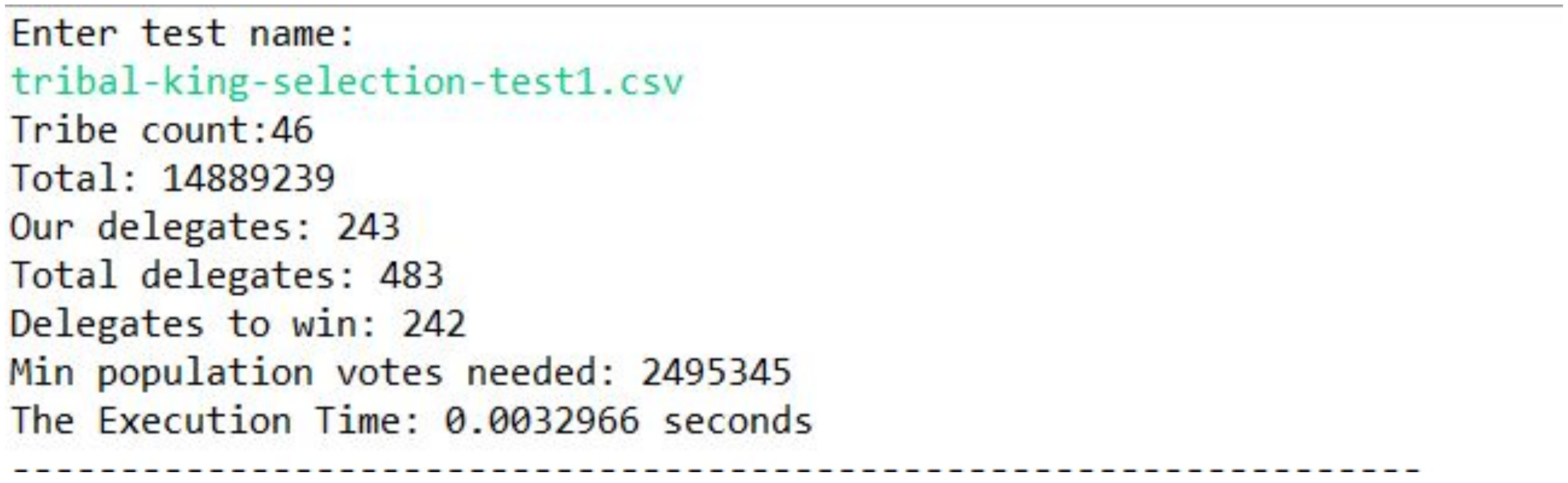
**Let the size of the array list called tribes be N. Then, since we add all of the elements of the array list called tribes to the priority queues called prioTribes and populoTribes, the sizes of the priority queues will also be N. There are two separate for loops in our algorithm. Moreover, there are two nested while loops. The first while loop creates a time complexity depending on the size of the prioTribes queue, which is O(N). The second while loop creates a complexity depending on the size of the populoTribes queue, which is also O(N). This leads a time complexity of O(N) since we enter the second while loop only in the last iteration which makes it 2*N.**

## Space Complexity of Our Algorithm

We used ArrayList to store the tribes while parsing the test data. Then, we created 2 Priority Queues to list the tribes in accordance with the delegate value and tribes' population. Each of the lists' capacity are equal to the number of the tribes. Thus, if we let the size of the queues and the size of the array list equal to n, the space complexity of our algorithm is O(3*n), which is O(n).

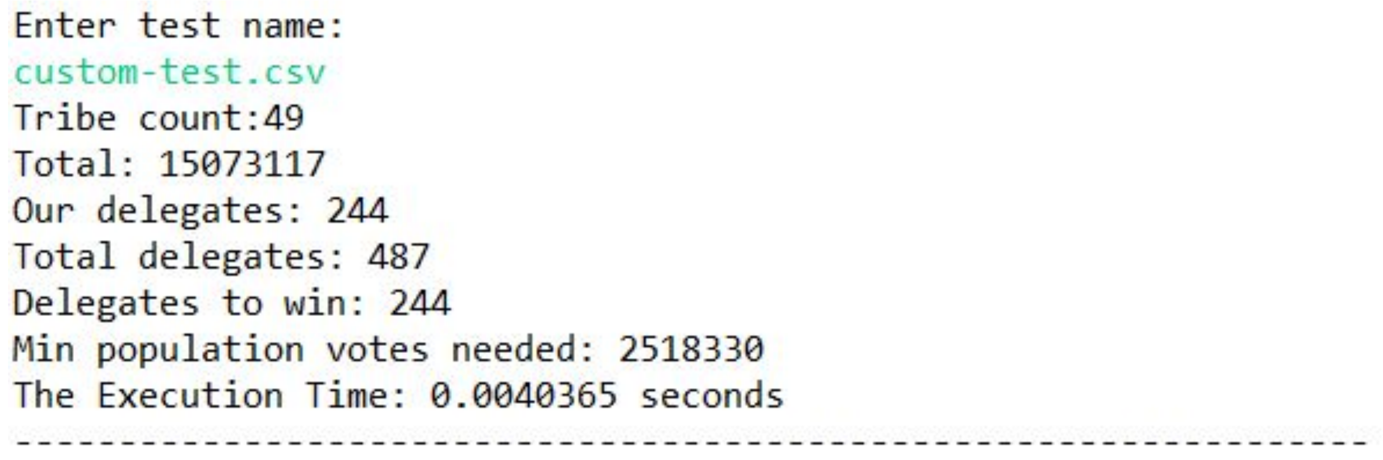
# Run Times for Each of the Test Cases

The screenshot for the run time of the test case 1 in eclipse



```
Enter test name:
tribal-king-selection-test1.csv
Tribe count:46
Total: 14889239
Our delegates: 243
Total delegates: 483
Delegates to win: 242
Min population votes needed: 2495345
The Execution Time: 0.0032966 seconds
-----------------------------------------------------------------
```

## The screenshot for the runtime of the test case 2 in eclipse



```
Enter test name:
custom-test.csv
Tribe count:49
Total: 15073117
Our delegates: 244
Total delegates: 487
Delegates to win: 244
Min population votes needed: 2518330
The Execution Time: 0.0040365 seconds
-----------------------------------------------------------------
```

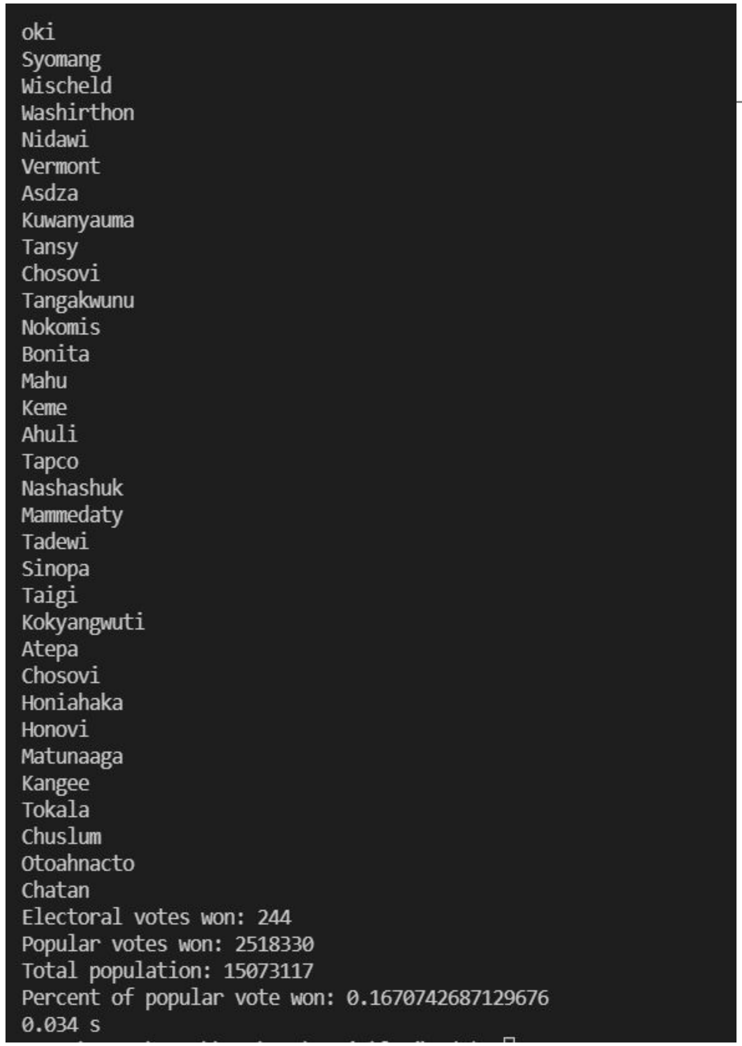
## Different Approaches

If population of each district where Pi and number of delegates were Di, **linear programming** could be used where goal is to minimize ΣPi subject to; Σ(x*Di)>=(ΣDi)/2 where x is the binary decision variable determining if we would use that district in election. With this approach we have n many decision variables which would be complex however, if we take the duality of problem we have n many constraints and just single decision variable.

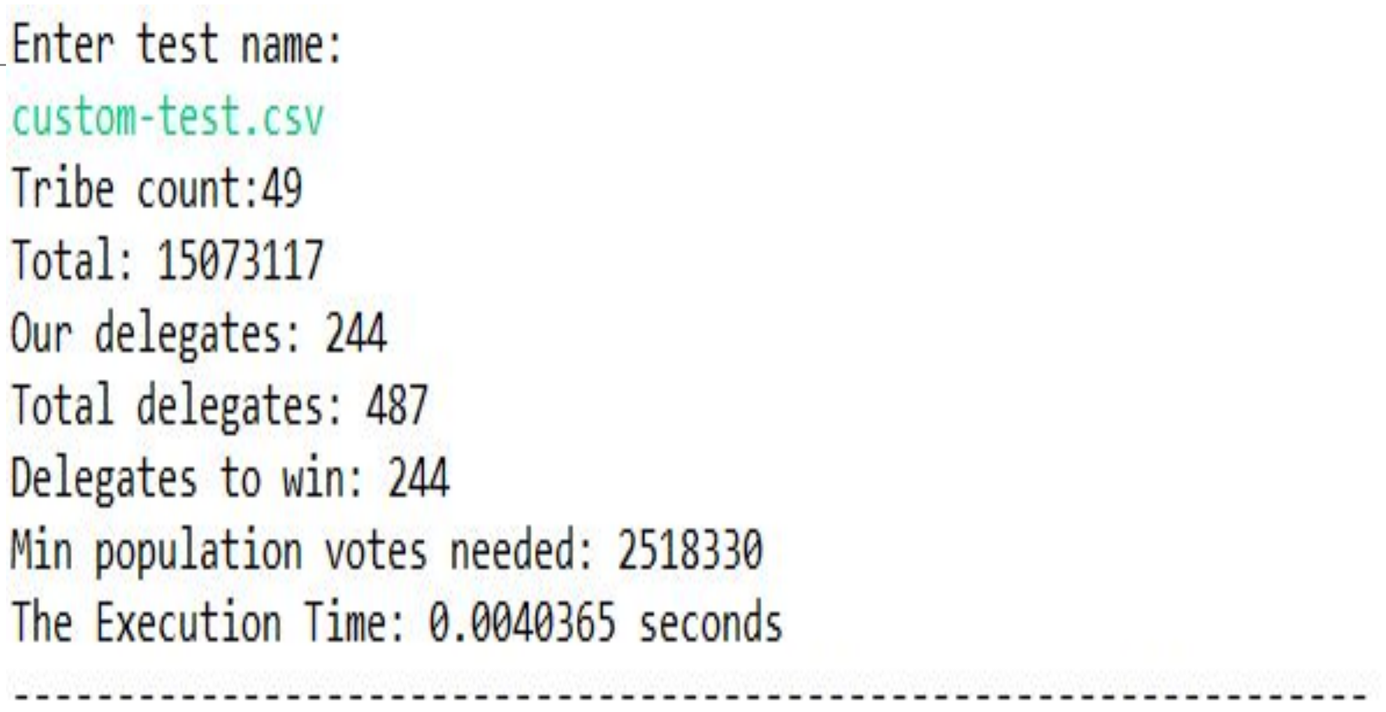
## Different Approaches

- Recursive method for solving knapsack problem.
- This is also good approach as this electoral problem is quite similar to knapsack problem.
- We can convert the inputs of the known NP-complete subset sum problem to the inputs of this problem, where the first set includes the delegate numbers of the tribes and where the second set includes the population votes of the tribes. Here, our target sums will be the minDelegatesToWin and minPopVotesNeeded for the sets respecitvely. Moreover, the subsets should include the delegate number of each tribe in tribes and the population votes of each tribe in tribes for which we satisfy the minDelegatesToWin and minPopVotesNeeded situations. We also need to run this for more delegates than minDelegatesToWin as there might be case that we could achieve more delegates with less population.

# SUBSET SUM VS OUR ALGORITHM



```
oki
Syomang
Wischeld
Washirthon
Nidawi
Vermont
Asdza
Kuwanyauma
Tansy
Chosovi
Tangakwunu
Nokomis
Bonita
Mahu
Keme
Ahuli
Tapco
Nashashuk
Mammedaty
Tadewi
Sinopa
Taigi
Kokyangwuti
Atepa
Chosovi
Honiahaka
Honovi
Matunaaga
Kangee
Tokala
Chuslum
Otoahnacto
Chatan
Electoral votes won: 244
Popular votes won: 2518330
Total population: 15073117
Percent of popular vote won: 0.1670742687129676
0.034 s
```



```
Enter test name:
custom-test.csv
Tribe count:49
Total: 15073117
Our delegates: 244
Total delegates: 487
Delegates to win: 244
Min population votes needed: 2518330
The Execution Time: 0.0040365 seconds
--------------------------------------------------------------------
```

**The result in the left is the subset sum problem's result and the result in the right is our algorithm's result. These results are both for the custom-test.csv file, which we made as an additional test case . Even the Subset algorithm run on javascript (which is faster than java approximately 6 times) Our algorithm (worked on java) is more faster. We used cutom-test.csv for comparsion.**

## Further improvements that can be done

More testing can be done with different samples to increase the accuracy of correctness in our algorithm.

***Further Improvements**

**It is still possible that there are cases where our algorithm does not yield the correct result. Even though it is very efficient compared to the alternatives such as subset-sum, we haven't formally proven its correctness. Further analysis on its correctness and handling of the edge cases can be done to further improve the algortihm.**

THANK YOU FOR LISTENING 😊