# CS 577: Midterm

Daniel Ko

Summer 2020

## §1 Describe an efficient divide and conquer strategy that identifies all members of this majority party.

### §1.1 Present your algorithm and explain the notations as necessary.

We assume that checking if two members are part of the same party takes constant time. Let $n$ be the total number of members. The main idea is we divide up the members into $n/2$ groups, each group containing one or two members. In each group make both members shake hands with everyone else in the city counsel. We check if either member is in majority political party, i.e. which has more than $n/2$ people, and return the member who is in the majority. One we find a majority members, we can traverse through all the members and generate a list

```
Data: members, a collection of n members; originalmembers, the original/unmodified
      collection of n members
Result: maj, a member in the majority party
def majority(members):
    if len(members) $\leq$ 1:
        return members[0]
    left = majority(members[0:floor(len(members)/2)])
    right = majority(members[floor(len(members)/2):len(members)])
    # Where count(hamilton) is defined as testing "hamilton", a member, with all other
      members and counting the number of people in hamilton's party
    numOfLeft = originalmembers.count(left)
    numOfRight = originalmembers.count(right)
    if numOfLeft > len(originalmembers)/2:
        return left
    elif numOfRight > len(originalmembers)/2:
        return right
    else:
        return null
```

```
Data: members, a collection of n members; maj, a member in a party
Result: partylist, all members in the political party
def partylist(members):
    partylist $\leftarrow$ a list to represent all the members in the party
    for person in members:
        if person and maj is in the same party:
            add person to partylist
    return partylist
```

### §1.2 Prove the correctness of your algorithm.

*Proof.* We show by strong induction that our algorithm is correct. Let $P(n)$ be the predicate, "majority halts and returns a members in a collection of $n$ members such that more than $n/2$ are equivalent if it exists". We define $n \in \mathbb{N}$.

**Base Case:** When $n = 0$, majority halts and returns null which is the correct output because the collection is empty. Hence, $P(0)$ holds.

**Inductive step:** Suppose $P(n)$ holds for all $n \leq k \in \mathbb{N}$. We show that $P(k+1)$ holds.

We observe that majority recursively calls collections of sizes $\lfloor (k+1)/2 \rfloor$ and $(k+1) - \lfloor (k+1)/2 \rfloor$. Since $\lfloor (k+1)/2 \rfloor \leq k$ and $(k+1) - \lfloor (k+1)/2 \rfloor \leq k$, we know by our strong inductive hypothesis that these recursive calls must halt. Hence, majority halts with an input of $k+1$ members.

majority finds the majority members of its left and right side, of size $\lfloor (k+1)/2 \rfloor$ and $(k+1) - \lfloor (k+1)/2 \rfloor$ respectively. Since both of these collections are less than $k$, by our strong inductive hypothesis, these recursive calls must return the correct output, the majority member of that partition if it exists. Then our algorithm calculates the total number of the majority member on the right and left on our $k+1$ collection and returns the majority. If no majority exists, it returns null. Thus, majority returns the correct output for a collection of $k+1$ members.

By strong induction, we have proven that $P(n)$ is true for all $n \in \mathbb{N}$. Therefore, majority is correct. □

### §1.3 Show the recurrence of the computing complexity and explain how did you obtain it. Note that you do not have to solve the recurrence since we can derive the solution using the Master's theorem.

We claim that the time complexity for majority is $T(n) = 2T(n/2) + 2n$ which is $\Theta(n \log n)$.

*Proof.* Each call to our majority will result in two recursive calls with its input sized being halfed, which accounts for $2T(n/2)$. Additionally, calculating the number of majority will require $n$ time and we do this computation for both the left and right card, hence requiring an overall $2n$ time. The rest of the computations take constant time. Using the master theorem, it is clear that we get values of $a = 2, b = 2, d = 1$ for this particular recurrence which results in a complexity of $\Theta(n \log n)$. □

It's trivial to see that the time complexity for partylist is $\Theta(n)$, as there is one for loop which runs $n$ times and checking if two people are part of the same party takes constant time. Thus the total run time complexity of majority and partylist is $\Theta(n \log n + n) = \Theta(n \log n)$

## §2 Minimize the total shipping cost.

### §2.1 Define your objective function clearly.

Denote $OPT(i, d)$ to be the optimal, minimal, cost of transporting products for weeks $i - d$ to $i$ where $3 \leq d \leq 5$.

### §2.2 Set up the correct recurrence relationship among the sub-problems. Provide a short reasoning about the correctness of this recurrence relationship

Let $S$ represent the set consisting of the weight of the products from $i - d$ to $i$. For example let $S = \{1, 2, 3, 4, 5\}$. $OPT(i, d)$ will equal the min cost between having all of $S$ use service A or service b. Notice that there are only two possibilities, either all products use service $A$ or $B$ because a combination of $A$ and $B$ will always be greater than just using only $B$.

$$OPT(i, d) = min(\sum_{a=i-d}^{i} w_a r, c)$$

This recurrence relationship is correct because we are checking all possible combinations.

### §2.3 Write pseudocode for the iterative implementation of optimal cost finding. Note that you need to write each necessary loop explicitly. Also give a brief analysis of the computing complexity

```
Data: weights, a size $n$ list containing weights of products.
Result: Min cost of sending all $n$ products.
def cost(weights):
    dp ← a $nx5$ matrix used for memoization, initialized with 0's
    for $i$ increasing from 1 to $n$:
        for $d$ increasing from 3 to 5:
            $dp[i][d] = min(\sum_{a=i-d}^{i}$ weights$[a]r, c)$
    #find the cheapest combination
    return go through dp, and find cheapest weights
```

Computing dp takes $O(n)$ time as there is only one for loop with n iterations. The part where "go through dp, and find cheapest weights" can be implemented in many ways. The brute force method takes $n!$ time. There is probably a more efficient way. I would look at the the cheapest cost from 1 to 3, 1 to 4, 1 to 5 and pick the cheapest cost. You would need to use some sort of sliding window to get all values. I suppose this takes $O(n^3)$. So total time complexity will be $O(n^3)$.

## §3 Design an algorithm using dynamic programming to obtain the maximal prize that one could get

### §3.1 Define your objective function clearly.

Denote $OPT(t, k)$ to be the optimal, maximal, prize you can get given you have $t$ minutes to play the game and must reach stop $k$ at the end of $t$.

### §3.2 Set up the correct recurrence relationship among the sub-problems. Provide a short reasoning about the correctness of this recurrence relationship

$OPT(t, k) = \{$ OPT(t-n, k)(the min cost of the combination of the prizes you can win from $n+1$ to $k$ with $t$

Call $OPT(m, 2n+1)$

### §3.3 Write pseudocode for the iterative implementation of optimal cost finding. Note that you need to write each necessary loop explicitly. Also give a brief analysis of the computing complexity

This would take $(2k+1)^2$ time.

## §4 Employees on a training course.

### §4.1 Present your randomized algorithm idea.

Assign every person independently with either course A or B, each with probability of 1/2.

### §4.2 Prove that your algorithm satisfies the probability requirement on limiting the failure of the project groups.

Let $X_p$ be a random variable such that

$$X_p = \begin{cases} 1 & \textbf{if } \text{project p has employees in course A and B} \\ 0 & \textbf{otherwise} \end{cases}$$

There are $r^2$ ways to divide a group of $r$ into courses A and B. The probability for each course to have people is $1-(1/2)^r$. Let $Y$ be the random variable denoting the number of projects that has employees in course A and B,

$$E[Y] = E[\sum_{p \in P} X_p] = \sum_{p \in P} E[X_p] = r^2(1-(1/2)^r)k \leq k2^{1-r}$$