Problem Set 5

Anthony Tam, CSC236 10/28/2016

Question 1: Write a divide-and-conquer algorithm for the given problem.

ANSWER: Part a)

```
def findWinner(votes, vote_list = {}):
    if len(votes) == 1:
        return votes[0]
    elif len(votes) == 0:
        return None
    else:
        if len(votes) % 2 == 0:
            required_votes = len(votes) // 2
        else:
            required_votes = (len(votes) // 2) + 1
        left_winner = findWinner(votes[:len(votes) // 2], vote_list)
        right_winner = findWinner(votes[len(votes) // 2:], vote_list)
        if left_winner != None:
            if left_winner in vote_list.keys():
                vote_list[left_winner] += 1
            else:
                vote_list[left_winner] = 1
        if right_winner != None:
            if right_winner in vote_list.keys():
                vote_list[right_winner] += 1
            else:
                vote_list[right_winner] = 1
        index = 0
        winner = ["", -1, False]
        for candidate in vote_list:
            if vote_list[candidate] > winner[1]:
                winner = [candidate, vote_list[candidate], False]
            elif vote_list[candidate] == winner[1]:
                winner[2] = True
        if winner[2] == False and winner[1] >= required_votes:
            return winner[0]
        else:
            return None
```

Part b)

1. Firstly, the program checks 2 base cases. If there are no votes, there is no winner: return None. If there is only 1 vote, that candidate is the winner: return it.
2. If we are in a case with multiple votes, first we calculate how many votes are required in order for a candidate to win.
3. We will then divide the list in half and calculate which candidate won in each half of the list. A dictionary keeps track of their votes and is passed by reference. We can remember the number of votes as we travel recursively.
4. Once the left and right winners are found, we increment their vote counts by 1. If they have not received any votes already, they are added to the dictionary and set to 1
5. We then use a for though all the voted candidates and check which has the highest number of votes. We also keep track if the current highest candidate has tied with someone else.
6. After finishing the loop, we can check if the highest candidate has not resulted in a tie and if they had the minimum number of votes. If both of these criteria is satisfied, the candidate is returned, otherwise None.

Part c)

$$T(n) = \begin{cases} a & n = 0 \\ a & n = 1 \\ c + 2T(\frac{n}{2}) + d & \textit{Otherwise} \end{cases}$$

1. In our base cases, we only have a return statement, which takes a constant amount of time.
2. Otherwise, we first must calculate the minimum votes required to win, which takes constant time.
3. We then make 2 recursive calls, each of which are half of the list creating $2T(\frac{n}{2})$ time
4. Afterwards we must increment the vote counts based on the left and right winners, find which candidate won the election and return this value. This all takes constant time d.

Part d)

$k = 1$

$c + 2T(\frac{n}{2}) + d$

$k = 2$

$c + 2(c + 2(c + 2T(\frac{n}{4}) + d) + d$

$3c + 4T(\frac{n}{4}) + 3d$

$k = 3$

$3c + 4(c + 2T(\frac{n}{8}) + d) + 3d$

$7c + 8T(\frac{n}{8}) + 7d$

$k = 4$

$7c + 8(c + 2T(\frac{n}{16}) + d) + 7d$

$15c + 16T(\frac{n}{16}) + 15d$

Potential Closed Form: $(2^k - 1)c + 2^k T(\frac{n}{2^k}) + (2^k - 1)d$

Solve for k using the base case:

$\frac{n}{2^k} = 1$

$n = 2^k$

$log_2 n = k$

Sub Back:

$(2^{log_2 n} - 1)c + 2^{log_2 n} T(1) + (2^{log_2 n} - 1)d$

$= (n - 1)c + n + (n - 1)d$

$= cn - c + n + dn - d$

$\therefore$ The closed form is: $cn - c + n + dn - d$

Part e)

Proving the Big O of the algorithm:

$\mathcal{O}(n) = cn - c + n + dn - d$

$\mathcal{O}(n) = cn - cn + n + dn - dn$

$\mathcal{O}(n) = n$

$\therefore$ the algorithm is in $\mathcal{O}(n)$.