PART 1

PREDEFINED ALGOS/DATA STRUC

Dictionary for each letter's corresponding button
Dictionary for each buttons press time
Array to store total time for each word
Bucket Sort to store common times with their corresponding words

1. Take in the words

The program begins by looping through each word within the txt file

2. Analyze letters

For each word, each letter is indexed and computed within calculateWordTime()

A If its the first letter

Match the letter to a predefined dictionary of values that correspond to the time it takes to press each letter

B Any other letter

Pass the current letter and the previous letter to calculateTimeBetween()

B2 Compare Buttons

If the two letters are on the same button, increment the wait time by 0.5, otheriwse increment the wait time by 0.25

3 Check Uppercase

If the letter is an uppercase, increment the total time by 2

B3 Analyze Letter Position

Depending on the buttons position, add the time it takes to press, either 0.25, 0.5, or 0.75 (a time of 0 for the first letter is case A)

4 Store Values

The total time calculated from the previous steps 1-3 is added to a bucket sort with its associated word. Due to the nature of the bucket sort, words with the same time will be grouped together

5 Store Values

Use a built in python method to sort and print the bucket for the smallest time, along with its corresponding word

FINISHED

PART 2

1. Take in data

The program begins by looping through the text file, grabbing the button number that is broken along with the words

2. Analyze letters

Check to see if the letter is on a broken button

If it is not a broken letter, repeat step 3-5

If it is a broken letter, call calculateBrokenButton() which applies the new rules

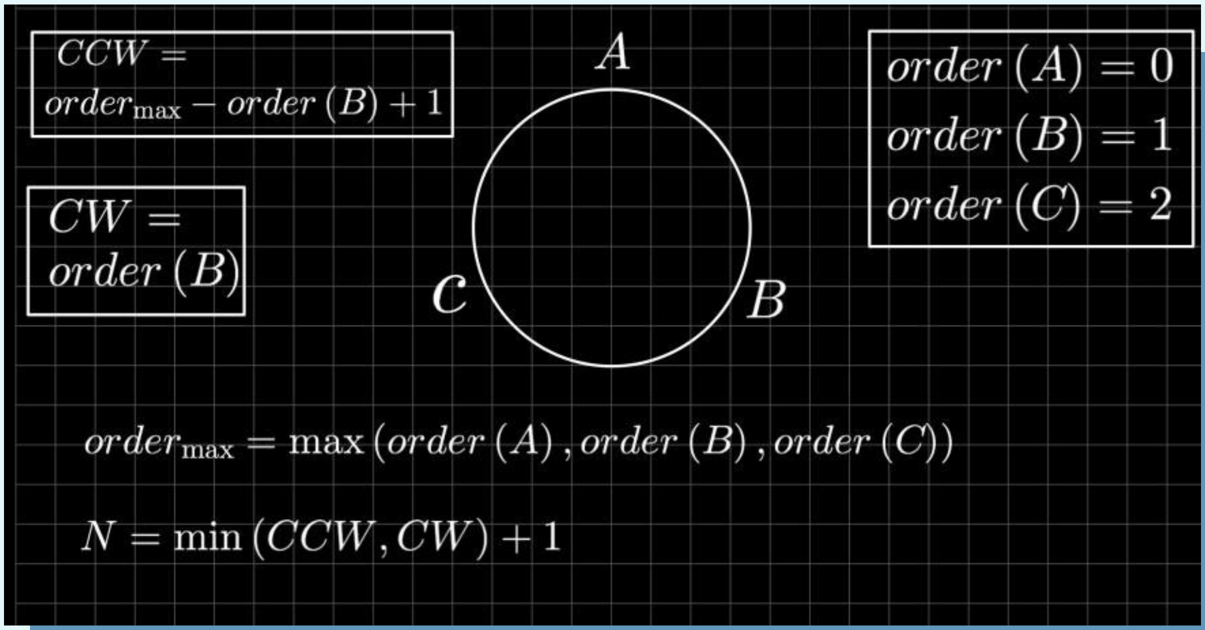
HOW IT WORKS

A TO ACCESS A LETTER ON A BROKEN BUTTON, BEGIN BY CLICKING THE POUND KEY THEN THE UP ARROW KEY(#) : TOTAL TIME 0.25 SECONDS

B A MENU OPTION WILL APPEAR ON THE DISPLAY SHOWCASING THE LETTERS OF THE BROKEN BUTTONS

C THE USER UTILIZES THE LEFT AND RIGHT ARROW KEYS TO LAND ON THE LETTER THEY DESIRE
EACH BUTTON CLICK: 0.25 SECONCS
FEATURE: FRONT AND BACK SCROLLING

D ONCE THE USER LANDS ON THE DESIRED LETTER, THEY LET GO OF THE # BUTTON TO LOCK IN THEIR CHOICE : 0 SECONDS



$CCW = order_{\max} - order(B) + 1$

$CW = order(B)$

$order(A) = 0$

$order(B) = 1$

$order(C) = 2$

$$order_{\max} = \max(order(A), order(B), order(C))$$

$$N = \min(CCW, CW) + 1$$

ADDED RULES

Rule 8:

If the letter is one from the broken button, and it is the first letter of the current tested word, it takes 0s to press the first button # (pound key) and takes 0.25s to press the up arrow key. Therefore it takes a total of 0.25s.

Rule 9:

When attempting to type a letter from the broken button, the wait time from any button to the # (pound key) is 0.25s. Even if the previous button was the # (pound key), it does not take any added time to press the button again since it must only be released and re-pressed. This is a clone of Rule 2, however for broken letters, Rule 4 does not apply (the 0.5s wait time).

PART 3

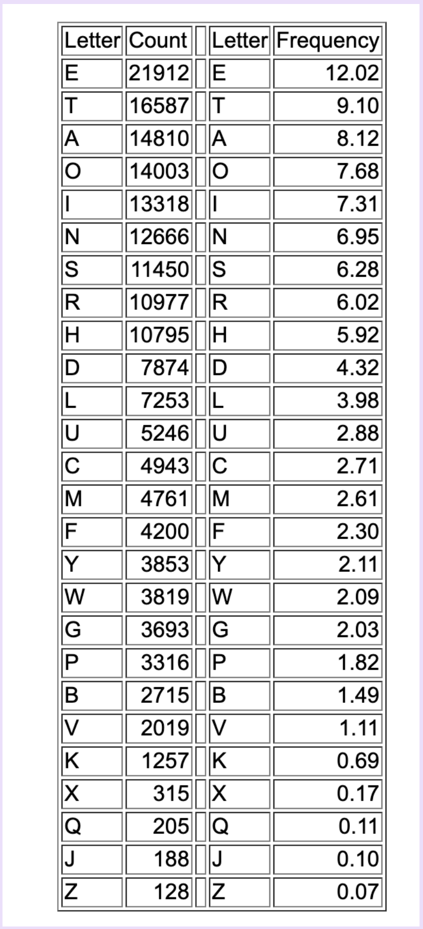
OUR IDEA: Restructure the button values

1. Reduced the number of letters on each button

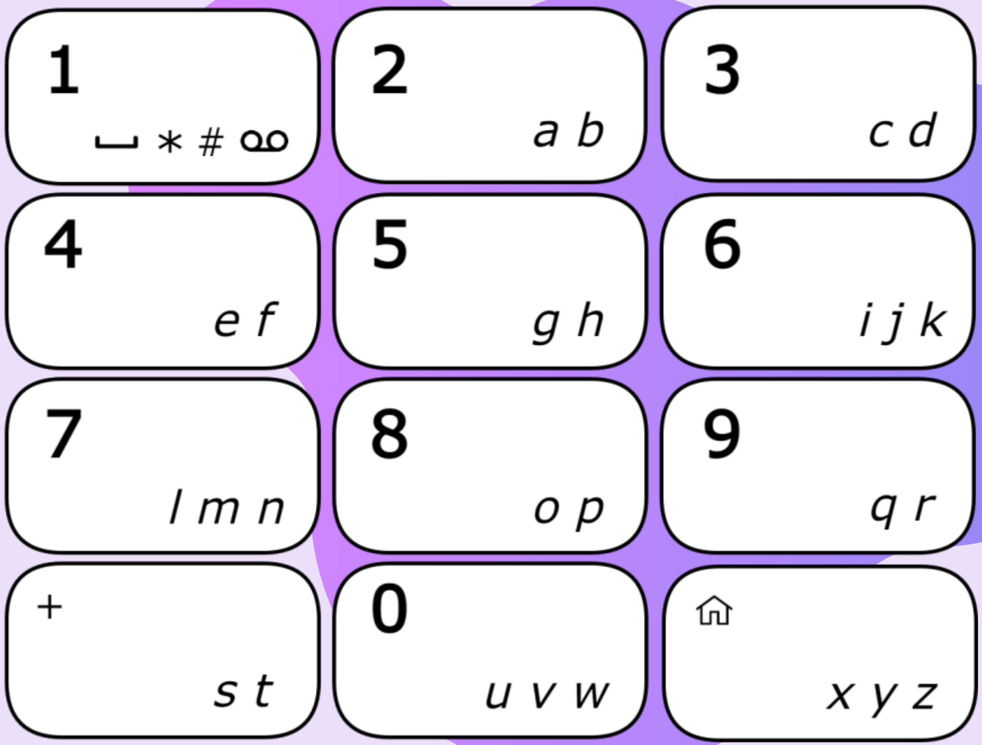
- Problem with previous design: Typing the third letter on the buttons takes 0.5 seconds and the fourth letter takes 0.75 seconds
- Our solution: Move the * and # from the bottom left and right buttons and move it to button 1, allowing for letters to be distributed

2. Prioritized letters with the highest frequency

- We researched commonly used letters, and found that they were primarily vowels. Our group decided to organize vowels to be the first letters on their respective keys.



Letter	Count		Letter	Frequency
E	21912		E	12.02
T	16587		T	9.10
A	14810		A	8.12
O	14003		O	7.68
I	13318		I	7.31
N	12666		N	6.95
S	11450		S	6.28
R	10977		R	6.02
H	10795		H	5.92
D	7874		D	4.32
L	7253		L	3.98
U	5246		U	2.88
C	4943		C	2.71
M	4761		M	2.61
F	4200		F	2.30
Y	3853		Y	2.11
W	3819		W	2.09
G	3693		G	2.03
P	3316		P	1.82
B	2715		B	1.49
V	2019		V	1.11
K	1257		K	0.69
X	315		X	0.17
Q	205		Q	0.11
J	188		J	0.10
Z	128		Z	0.07



References:
http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html