1. Given a list of numbers, create a function that removes 25% from every number in the list except the smallest number, and adds the total amount removed to the smallest number.

Examples:
show_the_love([4, 1, 4]) ➞ [3, 3, 3]
show_the_love([16, 10, 8]) ➞ [12, 7.5, 14.5]
show_the_love([2, 100]) ➞ [27, 75]

**Ans**:

In [21]:
```
def show_the_love(in_list):
    output = in_list.copy()
    sum_num = 0
    for i in range(len(output)):
        if output[i] is not min(output):
            sum_num += output[i]/4
            output[i] =  output[i]-(output[i]/4)
    output[output.index(min(output))] = sum_num +min(output)
    print(f'show_the_love({in_list}) ➞ {output}')
show_the_love([4, 1, 4])
show_the_love([16, 10, 8])
show_the_love([2, 100])
```

```
show_the_love([4, 1, 4]) ➞ [3.0, 3.0, 3.0]
show_the_love([16, 10, 8]) ➞ [12.0, 7.5, 14.5]
show_the_love([2, 100]) ➞ [27.0, 75.0]
```

2. Create a function that takes in two words as input and returns a list of three elements, in the following order:

1.Shared letters between two words. 2.Letters unique to word 1. 3.Letters unique to word 2.

Each element should have unique letters, and have each letter be alphabetically sorted.

Examples:
letters("sharp", "soap") ➞ ["aps", "hr", "o"]
letters("board", "bored") ➞ ["bdor", "a", "e"]
letters("happiness", "envelope") ➞ ["enp", "ahis", "lov"]
letters("kerfuffle", "fluffy") ➞ ["flu", "ekr", "y"]
# Even with multiple matching letters (e.g. 3 f's), there should
# only exist a single "f" in your first element.
letters("match", "ham") ➞ ["ahm", "ct", ""]
# "ham" does not contain any letters that are not found already
# in "match".

**Ans**:

In [22]:

```
def letters(word1,word2):
    set1 = set(word1)
    set2 = set(word2)
    output = []
    output.append(''.join(sorted(set1.intersection(set2))))
    output.append(''.join(sorted(set1.difference(set2))))
    output.append(''.join(sorted(set2.difference(set1))))
    print(f'letters{word1,word2} → {output}')
letters("sharp", "soap")
letters("board", "bored")
letters("happiness", "envelope")
letters("kerfuffle", "fluffy")
letters("match", "ham")
```

```
letters('sharp', 'soap') → ['aps', 'hr', 'o']
letters('board', 'bored') → ['bdor', 'a', 'e']
letters('happiness', 'envelope') → ['enp', 'ahis', 'lov']
letters('kerfuffle', 'fluffy') → ['flu', 'ekr', 'y']
letters('match', 'ham') → ['ahm', 'ct', '']
```

3. Write a function that pairs the first number in an array with the last, the second number with the second to last, etc.

Examples:
pairs([1, 2, 3, 4, 5, 6, 7]) → [[1, 7], [2, 6], [3, 5], [4, 4]]
pairs([1, 2, 3, 4, 5, 6]) → [[1, 6], [2, 5], [3, 4]]
pairs([5, 9, 8, 1, 2]) → [[5, 2], [9, 1], [8, 8]]
pairs([]) → []

**Ans**:

In [23]:

```
def pairs(in_list):
    listcopy = in_list.copy()
    output = []
    while True:
        if len(in_list) > 0:
            if len(in_list) == 1:
                output.append([in_list[0],in_list.pop(0)])
            else:
                output.append([in_list.pop(0),in_list.pop(-1)])
        else:
            break
    print(f'pairs({listcopy}) → {output}')
pairs([1, 2, 3, 4, 5, 6, 7])
pairs([1, 2, 3, 4, 5, 6])
pairs([5, 9, 8, 1, 2])
pairs([])
```

```
pairs([1, 2, 3, 4, 5, 6, 7]) → [[1, 7], [2, 6], [3, 5], [4, 4]]
pairs([1, 2, 3, 4, 5, 6]) → [[1, 6], [2, 5], [3, 4]]
pairs([5, 9, 8, 1, 2]) → [[5, 2], [9, 1], [8, 8]]
pairs([]) → []
```

4. Write a function that adds two numbers. The catch, however, is that the numbers will be strings.

Examples:
add_str_nums("4", "5") → "9"
add_str_nums("abcdefg", "3") → "-1"
add_str_nums("1", "") → "1"

add_str_nums("1874682736267235927359283579235789257", "32652983572985729") ➞ "1874682736267235927391936562808774986"

**Ans**:

In [24]:
```
def add_str_nums(num1,num2):
    num1 = num1 if len(num1) > 0 else "0"
    num2 = num2 if len(num2) > 0 else "0"
    if num1.isdigit() == False or num2.isdigit() == False:
        output = -1
    else:
        output = int(num1)+int(num2)
    print(f'add_str_nums{num1,num2} ➞ {str(output)}')
add_str_nums("4", "5")
add_str_nums("abcdefg", "3")
add_str_nums("1", "")
add_str_nums("1874682736267235927359283579235789257", "32652983572985729")
```

```
add_str_nums('4', '5') ➞ 9
add_str_nums('abcdefg', '3') ➞ -1
add_str_nums('1', '0') ➞ 1
add_str_nums('1874682736267235927359283579235789257', '32652983572985729') ➞ 1874682736267235927391936562808774986
```

5. lPaeesh le pemu mnxit ehess rtnisg! Oh, sorry, that was supposed to say: Please help me unmix these strings!

Somehow my strings have all become mixed up; every pair of characters has been swapped. Help me undo this so I can understand my strings again.

Examples:
unmix("123456") ➞ "214365"
unmix("hTsii  s aimex dpus rtni.g") ➞ "This is a mixed up string."
unmix("badce") ➞ "abcde"

**Ans**:

In [25]:
```
def unmix(in_string):
    output = ''
    for i in range(0,len(in_string)-1,2):
        output += in_string[i+1]+in_string[i]
        if (len(in_string)%2 != 0 and i == len(in_string)//2 ):
            output += in_string[-1]
    print(f'unmix({in_string}) ➞ {output}')
unmix("123456")
unmix("hTsii  s aimex dpus rtni.g")
unmix("badce")
```

```
unmix(123456) ➞ 214365
unmix(hTsii  s aimex dpus rtni.g) ➞ This is a mixed up string.
unmix(badce) ➞ abcde
```