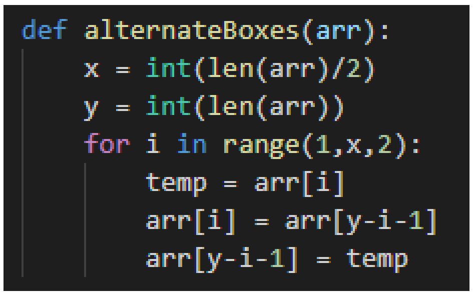
CSE 321 HOMEWORK 3
REPORT

1-) There are 2n boxes standing in a row; the first n of them are black and the remaining n boxes are white. Designed a decrease-and-conquer algorithm that makes the boxes alternate in a black-white-black-white pattern using minimum number of moves.

Black	Black	Black	Black	Black	White	White	White	White	White
0	1	2	3	4	5	6	7	8	9

Black	White	Black	White	Black	White	Black	White	Black	White
0	1	2	3	4	5	6	7	8	9

In order to change the array as shown above with the minimum number of moves,the boxes that will remain unmoved should be considered first.The boxes with even index numbers keep their place and only boxes with odd index numbers will be swapped with their matching box symmetricaly from head and from end of the list. For instance there are 10 boxes as first 5 of them are black and rest of 5 are White.List starts with 0,which means no move required.Then for index 1,box number 1 should be swapped with number 8(symmetric box of box 1).Then goes on until reaching the middle of the boxes.



```python
def alternateBoxes(arr):
    x = int(len(arr)/2)
    y = int(len(arr))
    for i in range(1,x,2):
        temp = arr[i]
        arr[i] = arr[y-i-1]
        arr[y-i-1] = temp
```

Since the number of boxes is considered to be 2n,this decrease and conquer algorithm executes (n-1)/2 swap exactly whether in the worst case,best case or average case.There is no conditional situation,which makes the time complexity of program:

Cworst = Cbest = Caverage = Θ(n)

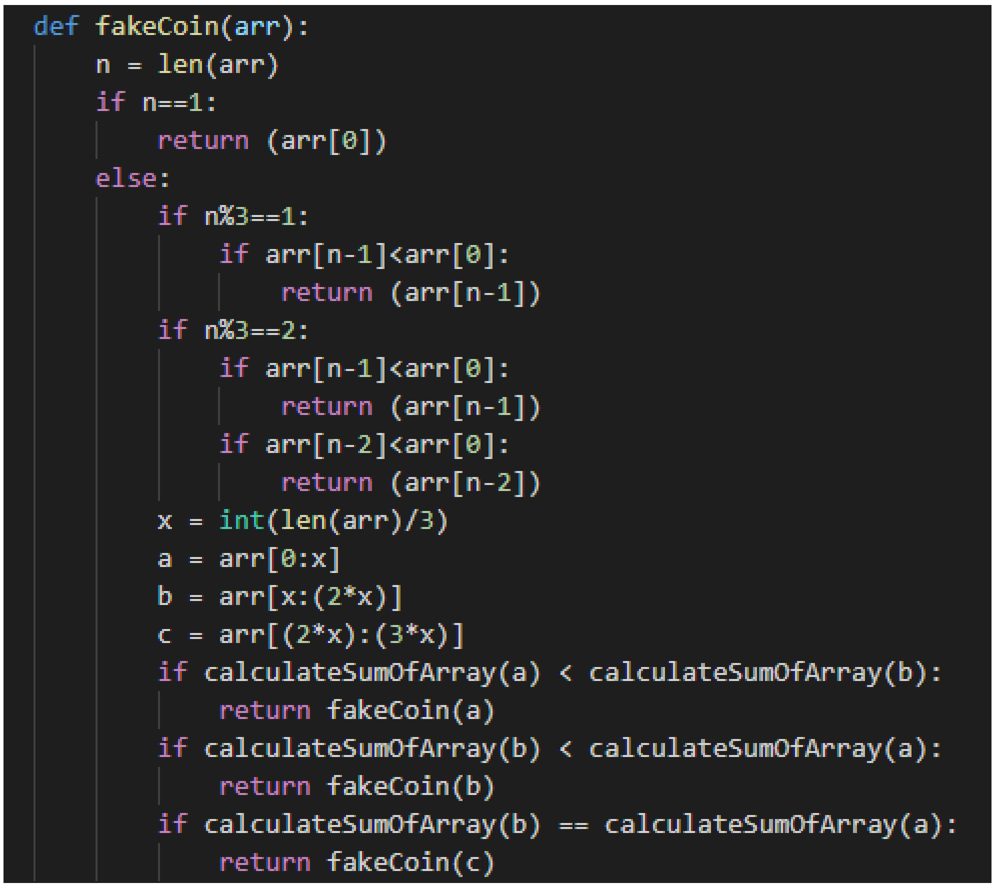
2) There are n coins which look exactly the same but one of them is fake. The weighbridge can be used in this problem which has two scales to find the fake coins.

In this algorithm,approach to solve the problem is below:

i) Set of coins should be seperated into 3 subgroups(even if the number is not 3n).
ii) Compare subgroup1 to subgroup2.
iii) If they have same weight,then the subgroup3 contains the fake coin,If subgroup1 is heavier than subgroup2,then subgroup2 contains the fake coin and vice versa.
iv) Recur the algorithm for the subgroup which contains the fake coin.
v) If size equal to 1,then return that element as the fake coin.

However,the number of inputs might not be 3n,it should be considered before starting recurrence.

Input size is considered to be k:
If k%3(k mod 3) is equal to 1: The last element should be compared with the first element of the list to decide whether it is fake or not.
If k%3 is equal to 2: The last 2 elements should be compared to each other in order to determine whether any of them is fake or not.



```python
def fakeCoin(arr):
    n = len(arr)
    if n==1:
        return (arr[0])
    else:
        if n%3==1:
            if arr[n-1]<arr[0]:
                return (arr[n-1])
        if n%3==2:
            if arr[n-1]<arr[0]:
                return (arr[n-1])
            if arr[n-2]<arr[0]:
                return (arr[n-2])
        x = int(len(arr)/3)
        a = arr[0:x]
        b = arr[x:(2*x)]
        c = arr[(2*x):(3*x)]
        if calculateSumOfArray(a) < calculateSumOfArray(b):
            return fakeCoin(a)
        if calculateSumOfArray(b) < calculateSumOfArray(a):
            return fakeCoin(b)
        if calculateSumOfArray(b) == calculateSumOfArray(a):
            return fakeCoin(c)
```

This algorithm always recurs until the reaching size 1 in every case except the case that input is not 3n and the fake coin is in the last 2 spaces.However it does not affect the average case considering that the probability of it is too small to change the average case.

In the end,if calculating the sum of array(calculating total weight of coins) is considered to be O(n) then the average and worst case of the algorithm is Θ(n*logn)

3) Average Case of QuickSort:

The input size is considered to be n

A partition can happen in any position S ($0 <= s <= n-1$)
after n+1 comp are made to achieve partition

0	…..	s	……..	n-1

After the partition,left and right subarrays will have s and (n-1-s) elements.
Assuming that the partition split can happen in each position S with probablity 1/n,following recurrence generated:

Cavg = 1/n * Σ [Cavg(s) + Cavg(n-1-s) + (n+1)] for n>1 (j is from 0 to n-1)
Cavg(0) = 0
Cavg(1) = 0
Cavg(n) = 2n*ln(n)= 1.39*n*log(n)

Average Case of Insertion Sort:

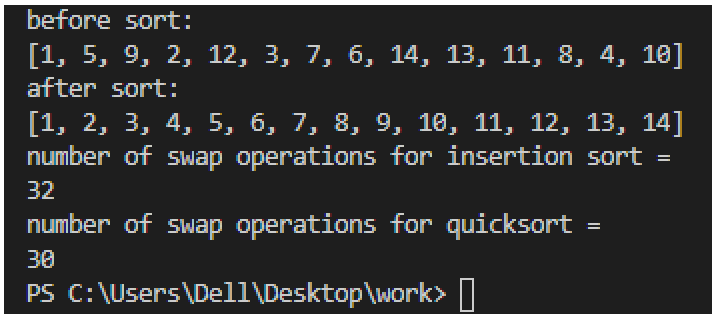
On each iteration:

1 comparison will occur with probablity 1/i+1
2 comparison will occur with probablity 1/i+1
3 comparison will occur with probablity 1/i+1

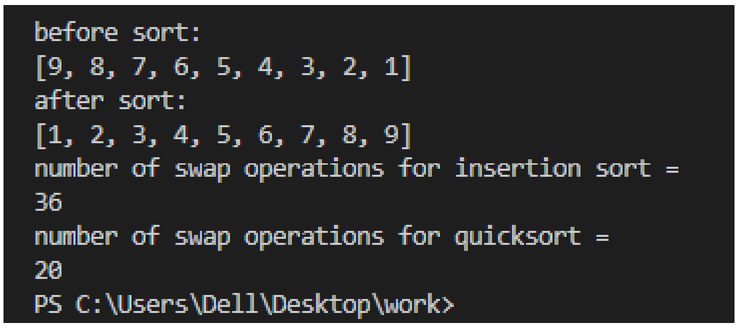
i comparison will occur with probablity 2/i+1

Cavg = (i from 1 to n-1) Σ (j from 1 to i-1) Σ (j*1/i+1)
= n*(n-1)/4 + (n-1) = Θ(n^2)

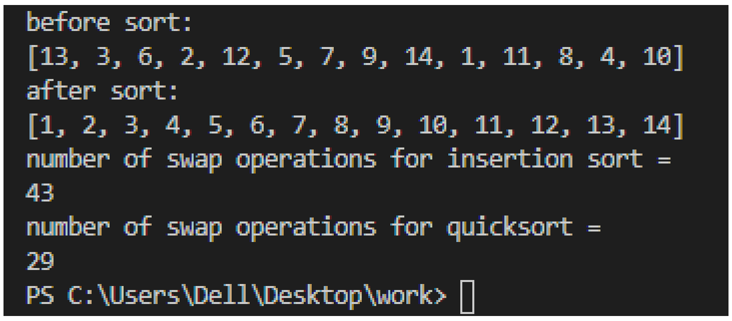
By the theoretical average case analysis,it is shown that for the average case quick sort is better choice.
In order to see the result on practice,some test cases executed:



```
before sort:
[1, 5, 9, 2, 12, 3, 7, 6, 14, 13, 11, 8, 4, 10]
after sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
number of swap operations for insertion sort =
32
number of swap operations for quicksort =
30
PS C:\Users\Dell\Desktop\work>
```



```
before sort:
[9, 8, 7, 6, 5, 4, 3, 2, 1]
after sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9]
number of swap operations for insertion sort =
36
number of swap operations for quicksort =
20
PS C:\Users\Dell\Desktop\work>
```



```
before sort:
[13, 3, 6, 2, 12, 5, 7, 9, 14, 1, 11, 8, 4, 10]
after sort:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
number of swap operations for insertion sort =
43
number of swap operations for quicksort =
29
PS C:\Users\Dell\Desktop\work>
```

The results prove that for the average case quicksort is better choice

4) Designed a decrease by constant factor algorithm that finds the median of an unsorted array.

In order to achieve this goal,The Quickselect Algorithm is used.Quickselect algorithm partition the unsorted list around a pivot value and after partition,list becomes more balanced which allows program to choose which part of list to recur to find Kth smallest element.

< Pivot	Pivot Value	>Pivot

After partitioning on each iteration,the distribution of list is shown above.

i) If K(in this specific algorithm K= size/2 + 1 which means median) is equal to index of pivot + 1, then pivot is median.

ii) If K is smaller than index of pivot + 1,then Kth smallest element of list is same as Kth element of the left sublist.

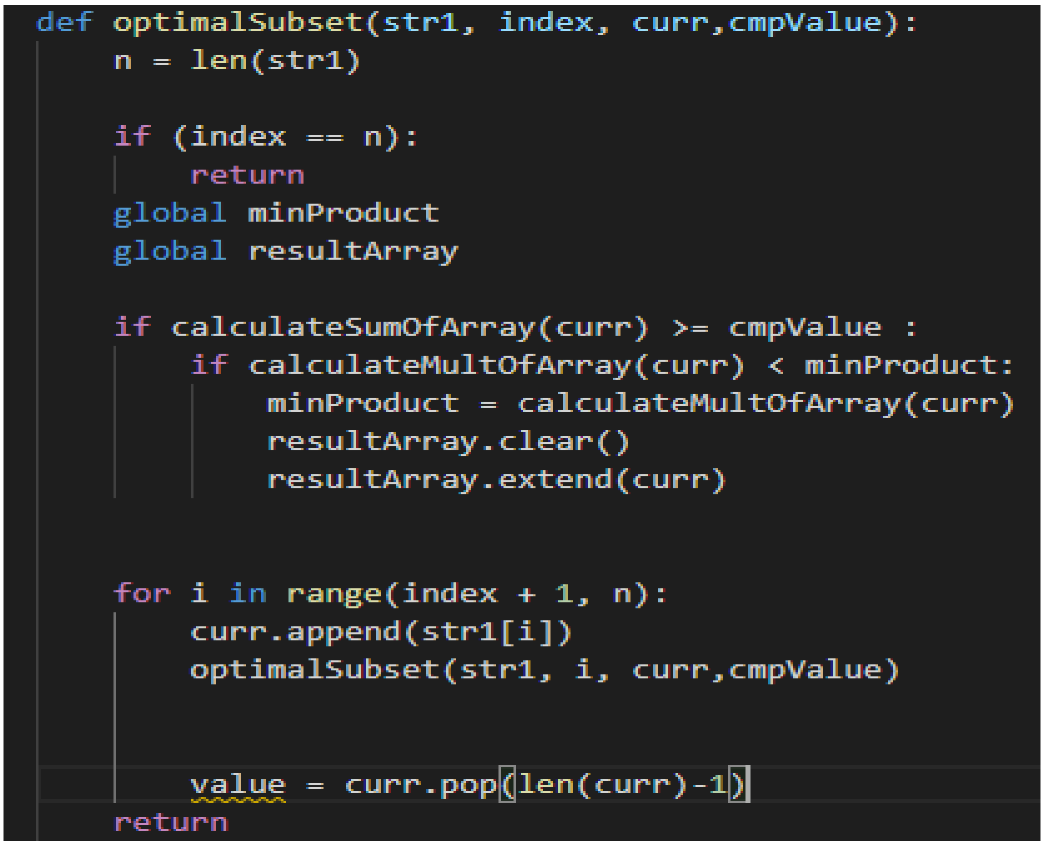
iii) If K is greater than index of pivot+1,then Kth smallest element of list is same as (K-S)th element of the right sublist.

For the worst case,there will be an extremely unbalanced partition on each iteration(n-1 times) which makes the one of the sublists empty and makes the other one with n-1 elements.

Cworst = (n-1)+(n-2)+…….+2+1 = n*(n+1) / 2 which is Θ(n^2)

5)

In order to achieve finding optimal subset which satisfies the given conditions,recursively each subset created and when it created,the calculating sum and multiplication is made.If the current subset is satisfying the condition to become optimal then program fills the result array with that subset.For the start condition minimum product is set to a huge junk value.



```python
def optimalSubset(str1, index, curr,cmpValue):
    n = len(str1)

    if (index == n):
        return
    global minProduct
    global resultArray

    if calculateSumOfArray(curr) >= cmpValue :
        if calculateMultOfArray(curr) < minProduct:
            minProduct = calculateMultOfArray(curr)
            resultArray.clear()
            resultArray.extend(curr)


    for i in range(index + 1, n):
        curr.append(str1[i])
        optimalSubset(str1, i, curr,cmpValue)


        value = curr.pop(len(curr)-1)
    return
```

The algorithm recur all the subsets (2^n) and makes the calculation on each subset which find the total sum and multiplication of the elements of that subset(maximum n).For the worst case the time complexity of this algortihm is Θ(n*2^n)