Analysis Report

Question 1:

The time complexity in the iterative method is O(n) as it has a loop that runs n time so the asymptotic big theta notation is θ(n)

However, in the divide and conquer method the time complexity is O(log(n)) so the asymptotic big theta notation is θ(log n) as it constantly divides the problem into halves(subproblems)

Recurrence of divide and conquer:

$T(n) = aT(n/b) + f(n)$

Since we only call (recur) on the function on time inside it then a = 1 and when we call it we divide the result into half therefore the b = 2. Then the remaining function is the best case or when the 'y' is either odd or even

$T(n) = T(n/2) + O(1)$

To solve the recurrence: using Master theorem

a = 1, b =2 ,

f(n) = O(1)

n^(log b a) = n^(0) = 1

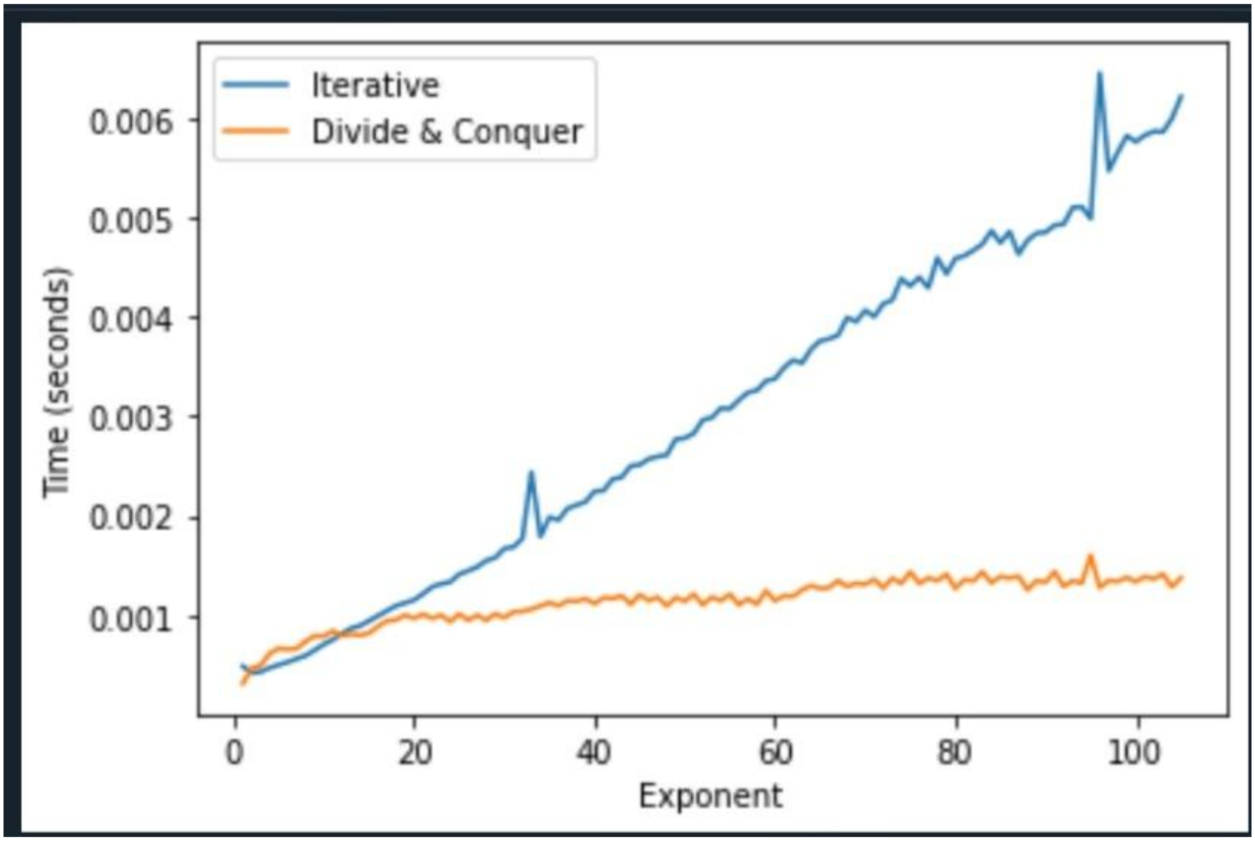
therefore f(n)=   n^(log b a)

then we will use case 2

then T(n) = θ(n^(log b alogn)

T(n) = θ(n^(0)logn) = θ(logn)

-The empirically observed running time matches the running time predicted by the analysis as shown in the graph below



Question 2:

The two major methods here to determine the time complexity is the merge sort and binary search where binary search time complexity takes O(log(n)) in the worst case however the merge sort takes O(nlog(n)) so the time complexity of the whole function is O(nlog(n)) so the asymptotic big theta notation is θ(nlog(n))

Recurrence of divide and conquer:

T(n) = aT(n/b) + f(n)

Since the slower function is merge sort we will have its values;

We call it 2 times (recur) in the function so a =2

And we call only its half so b = 2

While the remaining of the function calls loops on a value therefore O(n)

T(n) = 2T(n/2) + O(n)

To solve the recurrence: using Master theorem

a = 2, b =2 ,

f(n) = O(n)

n^(log b a) = n^(log 2 2) = n^(1) = n

therefore f(n)= n^(log b a)

then we will use case 2

then T(n) = θ(n^(log b alogn)

T(n) = θ(n^(1)log) = θ(nlogn)

-The empirically observed running time matches the running time predicted by the analysis as shown in the graph below