CSC 226
Zach Serviss
V00950002

Assignment 1

1. 3Sum

Input: Array of n integers A

Output: numbers which sum to zero

```
3SUM(array A)
quickSort(A) ———> O(n log n)
n = A.length
For int i=0 to n-2   ———> O(n)
        x = A[I]
        s = I+1
        e = n-1
        while s < e ———> O(n)
                y = A[s]
                z = A[e]
                if x+y+z == 0
                        output x, y, z
                        s++
                        e—
                else if (x+y+z > 0)
                        e—
                else
                        s++
```

Adding the significant time complexities = O(n log n) + O(n*n) c O(n^2)

Therefor the time complexity of this algorithm is O(n^2)

2. So well divide our array into 3 groups

(Because constants don't really matter I'll be using 5n as see in the 5 and 7 grouping linear select examples done in class)

So with our array into 3 equal groups, at the absolute maximum, our median of medians will be above or below 2n/3 where n is the number of elements in the array.

Therefore our equation for a 3 grouping linear select will be:

T(n) = 5n+T(n/3)+t(2n/3)

Lets guess T(n) <= cn

T(n) = 5n+T(n/3)+t(2n/3) <= cn

T(n) = 3(5n+T(n/3)+t(2n/3)) <= 3(cn)

T(n) = 15n+cn+2cn<=3cn

T(n) = 15n<=0

Therefore T(n) cannot be O(n)

3. To solve this inversion count problem in O(n log n) we must use a modified merge sort algorithm to break the array into smaller sub-problems.

-First we call mergeSort on the left half (i) and right half (j) of the given array

-We then have two fully sorted arrays

-We then count the number of inversions starting at the first indexes of the two subarrays.

-We count the inversions by looking at a[i] and a[j]. If a[i]>a[j] then there are mid-1 inversions because the left and right subarrays are sorted and all remaining subelements of the left subarray will be guaranteed to be greater than the right.

-We then recursively call to divide the array into halves, and sum the number of inversions of in the first half, second half, as well as the number of inversions when merging the two arrays together

-This gives us the total number of inversions of a given array with a runtime of O(n log n) because of the use of mergeSort.

4.

a.T(n) = 16T(n/4) + n^4, a=16,b=4,c=4

4>log(base 4) 16 (2)

Therefore T(n) is Theta(n^4)

b.T(n) = 125T(n/5) + n^2, a=125,b=5,c=2

2<log(base 5) 125 (3)

Therefore T(n) is Theta(n^(log(base 5)125))

c.T(n) = 64T(n/8) + n^2, a=64,b=8,c=2

2=log(base 8) 64 (2)

Therefore T(n) is Theta(n^2 log n)

5. For this algorithm we will simply XOR all the numbers in the array and the index

Because the values are from 0 to n and the index runs from 0 to n-1, XORing all these numbers against each other will give us the missing number.

Ex array b = [0,1,3,4] indexes = 0,1,2,3

0 XOR 0 XOR 1 XOR 1 XOR 2 XOR 3 XOR 3 XOR 4 = 2(the missing number)

```
missingNumber(array a)
        int missing = a.length
        for(int i =0; i< a.length; i++)
                missing  XOR= (i XOR a[i])
        return missing
```

This algorithm runs in O(n) time because we go through the whole array and the number of bits needed for the variable missing is just n bits where n is the number of elements in the array. Then we just xor the bits with the resulting xor between the array position and value. Because of this we actually only use a constant amount of extra bits, n so our space complexity in terms of bits is constant, therefore O(1).