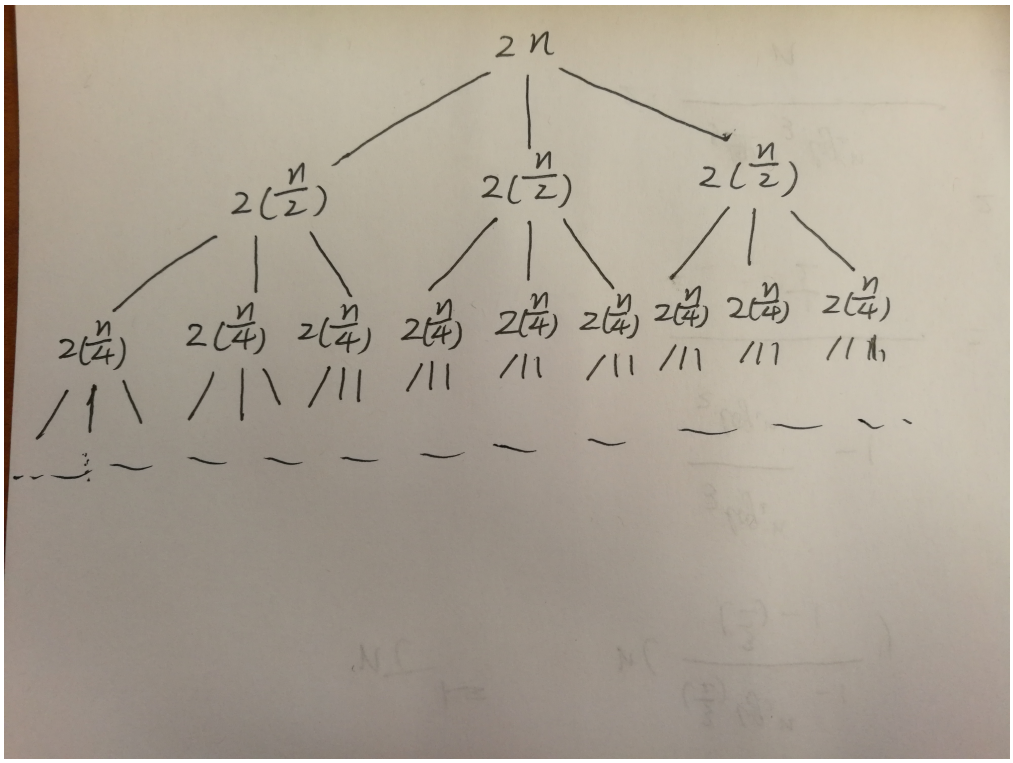
# CS580 Homework 1

Sihao Yin, Yuxuan Jiang (0028234022, 0028440468)

December 9, 2022

**Question 1.**

1. The following is how the recursion tree would look like



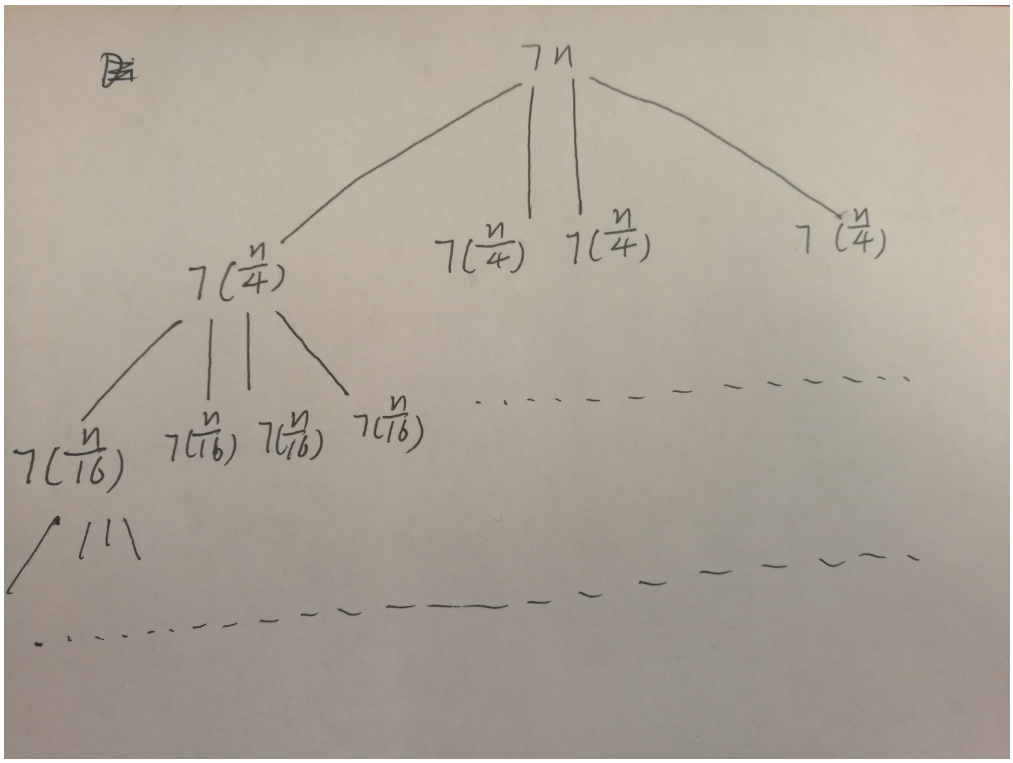

Sum of second level: 3 * $2(\frac{n}{2}) = 3n$
Sum of third level: 9 * $2(\frac{n}{4}) = \frac{9}{2}n$
Sum of fourth level: 27 * $2(\frac{n}{8}) = \frac{27}{4}n$

Since we are dividing by 2, the height of the recursion tree will be $log_2^n$
Since the number of nodes at height k is $3^k$, the number of nodes at the base level is $3^{log_2^n}$.
Hence T(n) = $2n + 2 * \frac{3}{2} n + 2 * (\frac{3}{2})^2 n + 2 * (\frac{3}{2})^3 n + ... + 2 * (\frac{3}{2})^{log_2^n - 1} n + 6 * 3^{log_2^n}$
= $2 * (n + \frac{3}{2} n + (\frac{3}{2})^2 n + (\frac{3}{2})^3 n + ... + (\frac{3}{2})^{log_2^n - 1} n) + 6 * n^{log_2^3}$

Since $n + \frac{3}{2} n + (\frac{3}{2})^2 n + (\frac{3}{2})^3 n + ... + (\frac{3}{2})^{log_2^n - 1} n$ is a geometric series, with a = n and r = $\frac{3}{2}$, the sum of it will be $\sum_{k=0}^{log_2^n - 1} n * ((\frac{3}{2})^k) = n * \frac{(\frac{3}{2})^{log_2^n} - 1}{(\frac{3}{2}) - 1} = 2n * ((\frac{3}{2})^{log_2^n} - 1) = 2(n^{log_2^3}) - 2n$

Hence T(n) = $4*(n^{log_2^3}) - 4n + 6 * n^{log_2^3} = \Theta(n^{log_2^3})$

2. The following is how the recursion tree would look like

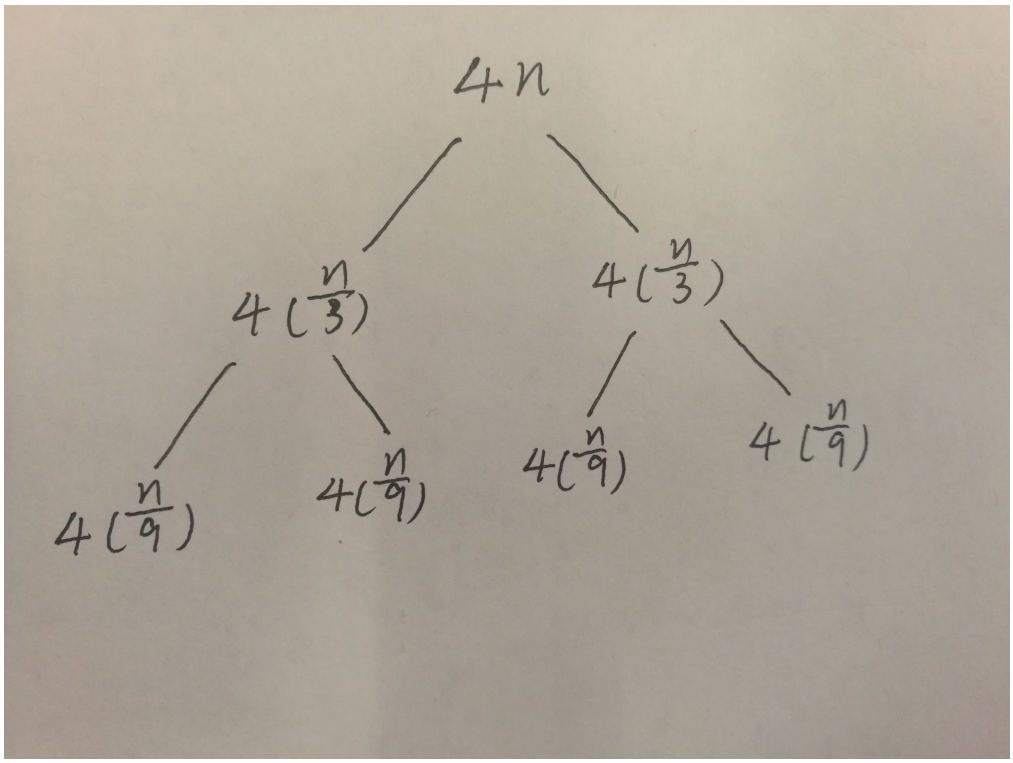



Sum of the second level: 7*($4 * \frac{n}{4}$)
Sum of the third level: 7*($4^2 * \frac{n}{4^2}$)
Sum of the fourth level: 7*($4^3 * \frac{n}{4^3}$)

Since we are dividing by 4, the height of the tree will be $log_4^n$
Since the number of nodes at height k is $4^k$, the number of nodes at the base level is $4^{log_4^n} =$ n.
Hence T(n) = 7n + 7n +....+7n
= 7n * ($log_4^n$) = 7n * $\frac{1}{2} * log_2^n$

Hence T(n) = 7n * $\frac{1}{2} * log_2^n = \Theta(n * log_2^n)$

3. The following is how the recursion tree would look like




Sum of the second level: $4*(2 * \frac{n}{3})$
Sum of the third level: $4*(2^2 * \frac{n}{3^2})$
Sum of the fourth level: $4*(2^3 * \frac{n}{3^3})$

Since we are diving by 3, the height of the recursion tree will be $log_3^n$
Since the number of nodes at height k is $2^k$, the number of nodes at the base level is $2^{log_3^n}$.
Hence T(n) = 4n + 4 * $\frac{2}{3}$ n + 4 * $(\frac{2}{3})^2$n + 4 * $(\frac{2}{3})^3$n + ...+ 4 * $(\frac{2}{3})^{log_3^n}$n + $2^{log_3^n-1}$
= 4 * (n + $\frac{2}{3}$ n + $(\frac{2}{3})^2$n + $(\frac{2}{3})^3$n + ... + $(\frac{2}{3})^{log_3^n-1}$n) + $n^{log_3^2}$

Since n + $\frac{2}{3}$ n + $(\frac{2}{3})^2$n + $(\frac{2}{3})^3$n +...+ $(\frac{2}{3})^{log_3^n}$n is a geometric series, with a = n and r = $\frac{2}{3}$, the sum of it will be $\sum_{k=0}^{log_3^n-1}$ n*$((\frac{2}{3})^k)$ = n * $\frac{1-(\frac{2}{3})^{log_3^n}}{(1-\frac{2}{3})}$ = $3n - 3(n^{log_3^2})$

Hence T(n) = $4 * (3n - 3(n^{log_3^2})) + n^{log_3^2} = 12n - 11(n^{log_3^2})$= $\Theta$(n)

**Question 2.**

Below is the pseudocode, we keep a global counter for number of inversions. This is essentially merge sort

```
numOfInver = 0
function mergeSort(A[1...N]):
    if N > 1:
        m = floor(N/2)
        mergeSort(A[1...m])
        mergeSort(A[m+1...N])
        merge(A[1...N],m)

function merge(A[1...N],m):
        B = a new array with length equals to left.length+right.length
        i = 1
        j = m+1
        for k from 1 to N:
            if j > N:
                B[k] = A[i]
                i = i + 1
            else if i > m:
                B[k] = A[j]
                j = j + 1
            else if A[i] <= A[j]:
                B[k] = A[i]
                i = i + 1
            else:
                numOfInver = numOfInver + (j-m)
                j = j + 1

        for k from 1 to N:
            A[k] = B[k]

        return B
```

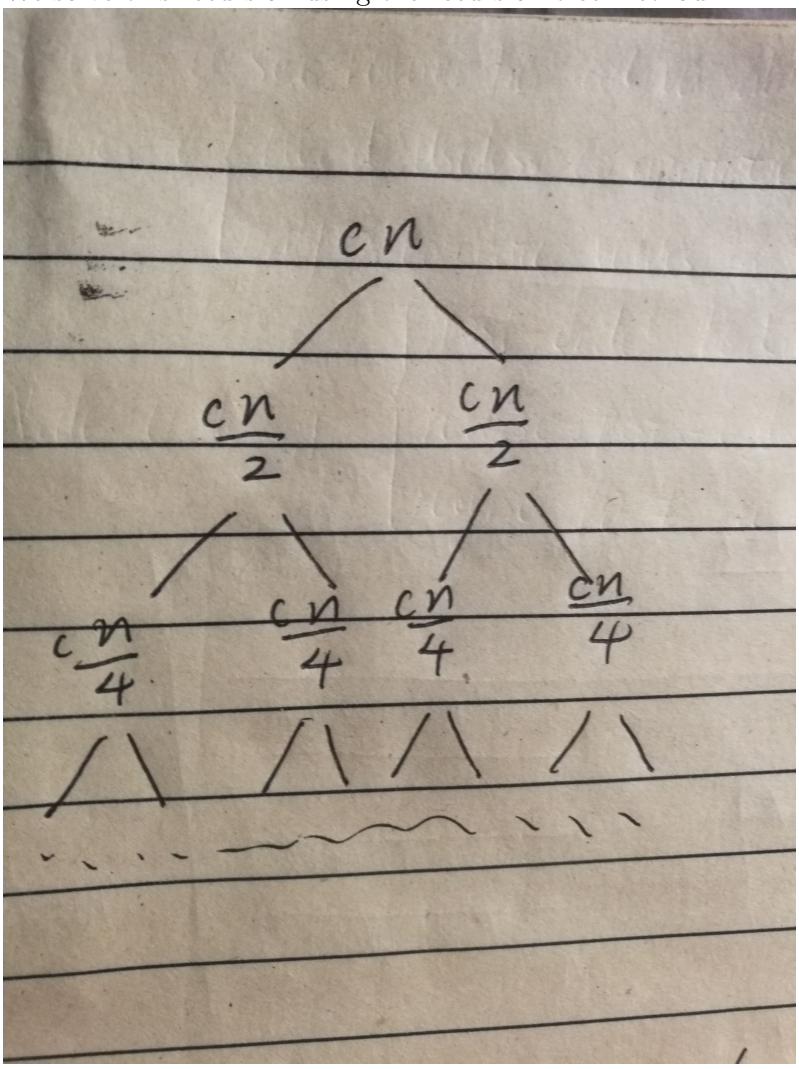
**Analysis:**

1. this algorithm is correct because in merge sort, the two subarrays are sorted when merging. Hence, when we check if A[i] > A[j], such an ith element must be on the left subarray, and jth element must be on the right. Since the right subarray is sorted, the ith element is also larger than any elements on the right subarray which are smaller than the jth element. Hence, numOfInver = numOfInver + (j-m)

2. In function mergesort, we divide the input array in half and recursively call the function

mergesort for each half. Then we call the function merge to merge the two halves.

3. In function merge, we loop through the two input arrays and increment the inversion counter accordingly. This takes linear time
4. With above, we have the this recurrence.: T(n) = $2(\frac{n}{2})$ + O(n).
5. We solve this recursion using the recursion tree method




   (a) Sum of the second level: cn
   (b) Sum of the third level: cn
   (c) Sum of the fourth level: cn

   Since we are dividing the input array by half each time in mergeSort, the height of the recursion tree is $log_2^n$. Hence T(n) = cn + cn + ... + cn = cn * $log_2^n$ = O(n $log_2^n$)