National University of Computer & Emerging Sciences, Karachi
Fall-2020 Department of Computer Science
Mid Term-1
20th October 2020, 10:30 AM – 11:30 AM

Course Code: CS302	**Course Name:** Design and Analysis of Algorithm
Instructor Name / Names: Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Zeshan Khan, Waqas Sheikh, Sohail Afzal	
Student Roll No:	**Section:**

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **6 questions** on **3 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 60 minutes. **Max Marks: 12.5**

Question # 1 **[0.5*4 = 2 marks]**

Are these following statements True or False? Prove your answer by computing the values of n_0, c_1, c_2 or by contradiction. [Θ is Theta and Ω is Omega]

Solution

$a)\ 4^3n + 4^2n + 4^1n = \Theta(n^2)$

False

$c_1 n^2 \leq 4^3 n + 4^2 n + 4^1 n \leq c_2 n^2$

$c_1 \leq 4^3/n + 4^2/n + 4^1/n \leq c_2$

$for\ n_0 = 1$

$c_1 \leq 64 + 16 + 4 \leq c_2$

$c_1 \leq 84 \leq c_2$

$for\ n = \infty$

$c_1 \leq 0 + 0 + 0 \leq c_2$

$c_1 \leq 0 \leq c_2$

$\boldsymbol{b)\ 2^n + n^2 = \Omega(2^n)}$

True

$c_1 2^n \leq 2^n + n^2$

$c_1 \leq 1 + n^2/2^n$

$for\ n_0 = 1$

$c_1 \leq 1 + 1/2$

$c_1 \leq 3/2$

$for\ n = \infty$

$c_1 \leq 1 + 0$

$c_1 \leq 1$

$\boldsymbol{c)\ 4^{\log_2 n} + n = \Theta(2^n)}$

False

$c_1 2^n \leq 4^{\log_2 n} + n \leq c_2 2^n$

$c_1 2^n \leq n^2 + n \leq c_2 2^n$

$c_1 \leq n^2/2^n + n/2^n \leq c_2$

$for\ n_0 = 1$

$c_1 \leq 1/2 + 1/2\ \leq c_2$

$c_1 \leq 1 \leq c_2$

$for\ n = \infty$

$c_1 \leq 0 + 0 \leq c_2$

$c_1 \leq 0 \leq c_2$

$\boldsymbol{d)\ \omega(f(n)) + o(f(n)) = \Omega(f(n))}$

True

Let: $f(n) = n^2$

$\omega(f(n)) = n^3$

$o(f(n)) = n$

$\omega(f(n)) + o(f(n)) = \Omega(f(n))$

$n^3 + n = \Omega(n^2)$ which is true

Question # 2 **[0.5*4 = 2 marks]**

Solve the following recurrences using **Master's Method.** Give argument, if the recurrence cannot be solved using Master's Method. [Master's method 4th case is given at last page]

$\boldsymbol{a)\ T(n) = 2T\left(\frac{n}{2}\right) + nlogn}$

$\boldsymbol{b)\ T(n) = 4T\left(\frac{n}{2}\right) + 4}$

$\boldsymbol{c)\ T(n) = 5T\left(\frac{n}{1}\right) + n^2}$

$\boldsymbol{d)\ T(n) = 2T\left(\frac{n}{2}\right) + n * 2^3}$

Solution

a) $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$

According to Masters Theorem 4th case:

if $f(n) \in \left(n^{\log_b a} \log^k n\right)$ then

$$T(n) \in \theta\left(n^{\log_b a} \log^{k+1} n\right)$$

so $a=2, b=2, k=1$

$$T(n) = \theta\left(n^{\log_2 2} \log^{1+1} n\right)$$

$$\boxed{T(n) = \theta(n \log^2 n)}$$

b) $T(n) = 4T\left(\frac{n}{2}\right) + 4$

$f(n) \in \theta(n^d)$ which means $f(n) \in \theta(n^0)$

so $d=0$

$a=4, b=2$

$a > b^d$ as $4 > 2^0$

According to Masters Theorem 3rd case:

if $a > b^d$ then $T(n) = \theta\left(n^{\log_b a}\right)$

$$T(n) = \theta\left(n^{\log_2 4}\right)$$

$$\boxed{T(n) = \theta(n^2)}$$

c) $T(n) = 5T\left(\frac{n}{1}\right) + n^2$

As $b=1$ so Master's Theorem is not applicable because $b \geq 2$ condition should be met to apply masters theorem.

d) $T(n) = 2T\left(\frac{n}{2}\right) + n * 2^3$

$T(n) = 2T\left(\frac{n}{2}\right) + 8n$

$f(n) \in \Theta(n^d)$ means $f(n) \in \Theta(n^1)$

so $d=1$

$a=2$, $b=2$

$a=b^d$ as $2=2^1$

According to second case of master's theorem

if $a=b^d$ then $T(n) = \Theta(n^d \log n)$

$$\boxed{T(n) = \Theta(n \log n)}$$

Question # 3 **[1.5 + 1 = 2.5 marks]**

Compute the time complexity of the following recurrence relations by using **Iterative Substitution Method** or **Recurrence-Tree Method**.

a) $\boldsymbol{T(n) = 3T\left(\frac{n}{3}\right) + n^3}$ **, Assume** $\mathbf{T(1) = 1}$ *(in solution it is assumed as T(1)=0. Please consider it T(1)=1*

b) $\boldsymbol{T(n) = 4T\left(\frac{n}{4}\right) + n^2}$ **, Assume** $\mathbf{T(1) = 1}$

Solution

QUESTION "a":

$T(n) = 3T(n/3) + n^3$, $T(1) = 0$

SOLUTION:

1st Iteration:

$T(n) = 3T(n/3) + n^3$

Calculate $T(n/3)$ by substituting "$n/3$" in place of "n" in the original equation.

$T(n/3) = 3T\left(\frac{n/3}{3}\right) + \left(\frac{n}{3}\right)^3$

$= 3T\left(\frac{n}{9}\right) + \frac{n^3}{27}$

2nd Iteration:

Plug $T\left(\frac{n}{3}\right)$ in original equation.

$T(n) = 3\left[3T\left(\frac{n}{9}\right) + \frac{n^3}{27}\right] + n^3 = 9T\left(\frac{n}{9}\right) + \frac{n^3}{9} + n^3$

Similarly, calculate $T\left(\frac{n}{9}\right)$.

$T\left(\frac{n}{9}\right) = 3T\left(\frac{n/9}{3}\right) + \left(\frac{n}{9}\right)^3$

$= 3T\left(\frac{n}{27}\right) + \frac{n^3}{9^3}$

3rd Iteration:

$$T(n) = 9\left[3T\left(\frac{n}{27}\right) + \frac{n^3}{9^3}\right] + \frac{n^3}{9} + n^3$$

$$= 27T\left(\frac{n}{27}\right) + \frac{n^3}{9^2} + \frac{n^3}{9} + n^3$$

We can see a pattern which can be written as the following general form:

$$T(n) = 3^K T\left(\frac{n}{3^K}\right) + \left(\frac{1}{9^{K-1}} + 1\right.$$

$$T(n) = 3^K T\left(\frac{n}{3^K}\right) + \left(\frac{1}{9^{K-1}} + \frac{1}{9^{K-2}} + \cdots + \frac{1}{9^0}\right) n^3$$

$$= 3^K T\left(\frac{n}{3^K}\right) + \left(\sum_{i=0}^{K-1} \frac{1}{9^i}\right) n^3$$

Note: $\boxed{\sum_{i=0}^{K-1} a^i = \frac{1-a^K}{1-a}}$

$$= 3^K T\left(\frac{n}{3^K}\right) + \left(\frac{1-(1/9)^K}{1-(1/9)}\right) n^3$$

$$= 3^K T\left(\frac{n}{3^K}\right) + \left(\frac{1-(1/9)^K}{8/9}\right) n^3$$

$$= 3^K T\left(\frac{n}{3^K}\right) + \frac{9}{8} n^3 \left(1-(1/9)^K\right)$$

$$= 3^K T\left(\frac{n}{3^K}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{9^K}\right)$$

Since, $T(1) = 0$, in order to equate the base case

$T(n/3^K = 1) = 0$

$$1 = \frac{n}{3^K} \Rightarrow n = 3^K \Rightarrow \log_3 n = K$$

$$T(n) = 3^K T\left(\frac{n}{3^K}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{9^K}\right)$$

Substituting the value of "K" in above equation:

$$T(n) = 3^{\log_3 n} T\left(\frac{n}{3^{\log_3 n}}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{9^{\log_3 n}}\right)$$

~~Since, $3^{\log_3 n}$~~

Since, $3^{\log_3 n}$ can be written as $n^{\log_3 3}$ which is equal to "n". Therefore,

$$T(n) = nT\left(\frac{n}{n}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{n^2}\right)$$

$$= nT(1) + \frac{9}{8}\left(n^3 - \frac{n^3}{n^2}\right)$$

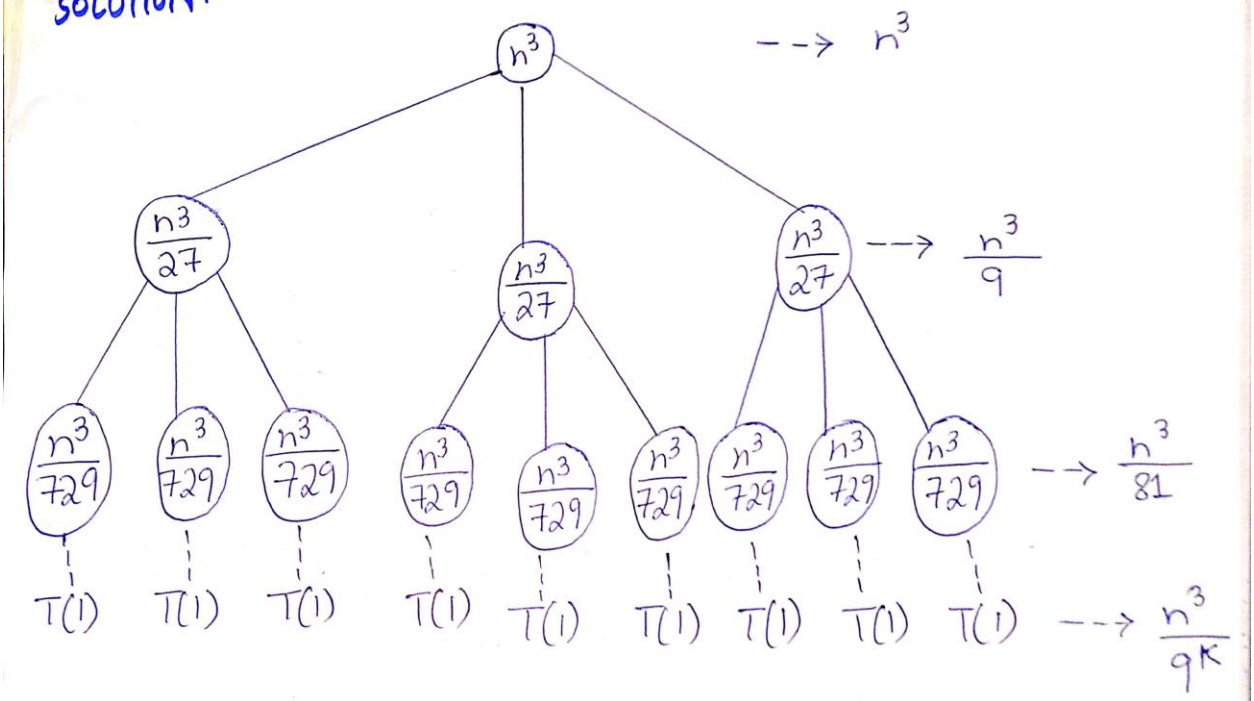
$$= n(0) + \frac{9}{8}(n^3 - n)$$

$$= \frac{9}{8}(n^3 - n)$$

$$= \frac{9}{8} n^3 - \frac{9}{8} n$$, hence $\Theta(n^3)$ is the time complexity.

QUESTION "a": $T(n) = 3T\left(\frac{n}{3}\right) + n^3$

SOLUTION:



- Height of left tree:

For $T(1) \Rightarrow \frac{n}{3^k} = 1$

$\Rightarrow k = \log_3 n$

- Height of right tree:

For $T(1) = \frac{n}{3^k} = 1$

$\Rightarrow k = \log_3 n$

Sum of recurrences:

$$= n^3 + \frac{n^3}{9} + \frac{n^3}{81} + \cdots + \frac{n^3}{9^k}$$

$$= n^3 \left[1 + \frac{1}{9} + \frac{1}{81} + \cdots + \frac{1}{9^k} \right]$$

Using the property of Geometric Series:

$$= n^3 \left[\frac{1}{1 - \frac{1}{9}} \right] \Rightarrow \frac{9}{8} n^3$$

Hence, $\Theta(n^3)$ is the time complexity.

QUESTION "b": ☆

$T(n) = 4T\left(\frac{n}{4}\right) + n^2$, $T(1) = 1$

SOLUTION USING ITERATIVE SUBSTITUTION:

1st Iteration:

$$T(n) = 4T\left(\frac{n}{4}\right) + n^2$$

Calculate $T\left(\frac{n}{4}\right)$ by substituting "$\frac{n}{4}$" in place of "n" in the original equation:

$$T\left(\frac{n}{4}\right) = 4T\left(\frac{n/4}{4}\right) + \left(\frac{n}{4}\right)^2$$

$$= 4T\left(\frac{n}{16}\right) + \frac{n^2}{16}$$

2nd Iteration:

Plug $T\left(\frac{n}{4}\right)$ in original equation:

$$T(n) = 4\left[4T\left(\frac{n}{16}\right) + \frac{n^2}{16}\right] + n^2$$

$$= 16T\left(\frac{n}{16}\right) + \frac{n^2}{4} + n^2$$

Similarly, calculate $T\left(\frac{n}{16}\right)$,

$$T\left(\frac{n}{16}\right) = 4T\left(\frac{n/16}{4}\right) + \left(\frac{n}{16}\right)^2$$

$$= 4T\left(\frac{n}{64}\right) + \frac{n^2}{256}$$

3rd Iteration:

$$T(n) = 16\left[4T\left(\frac{n}{64}\right) + \frac{n^2}{256}\right] + \frac{n^2}{4} + n^2$$

$$= 64T\left(\frac{n}{64}\right) + \frac{n^2}{16} + \frac{n^2}{4} + n^2$$

We can see a pattern which can be written as follows:

$$T(n) = 4^K T\left(\frac{n}{4^K}\right) + n^2\left(1 + \frac{1}{4} + \frac{1}{16}\right)$$

Using Geometric Series:

$$T(n) = 4^K T\left(\frac{n}{4^K}\right) + n^2\left[\frac{1}{1-\frac{1}{4}}\right]$$

$$= 4^K T\left(\frac{n}{4^K}\right) + \frac{4}{3}n^2$$

Since, $T(1) = 1 \Rightarrow \frac{n}{4^K} = 1 \Rightarrow K = \log_4 n$

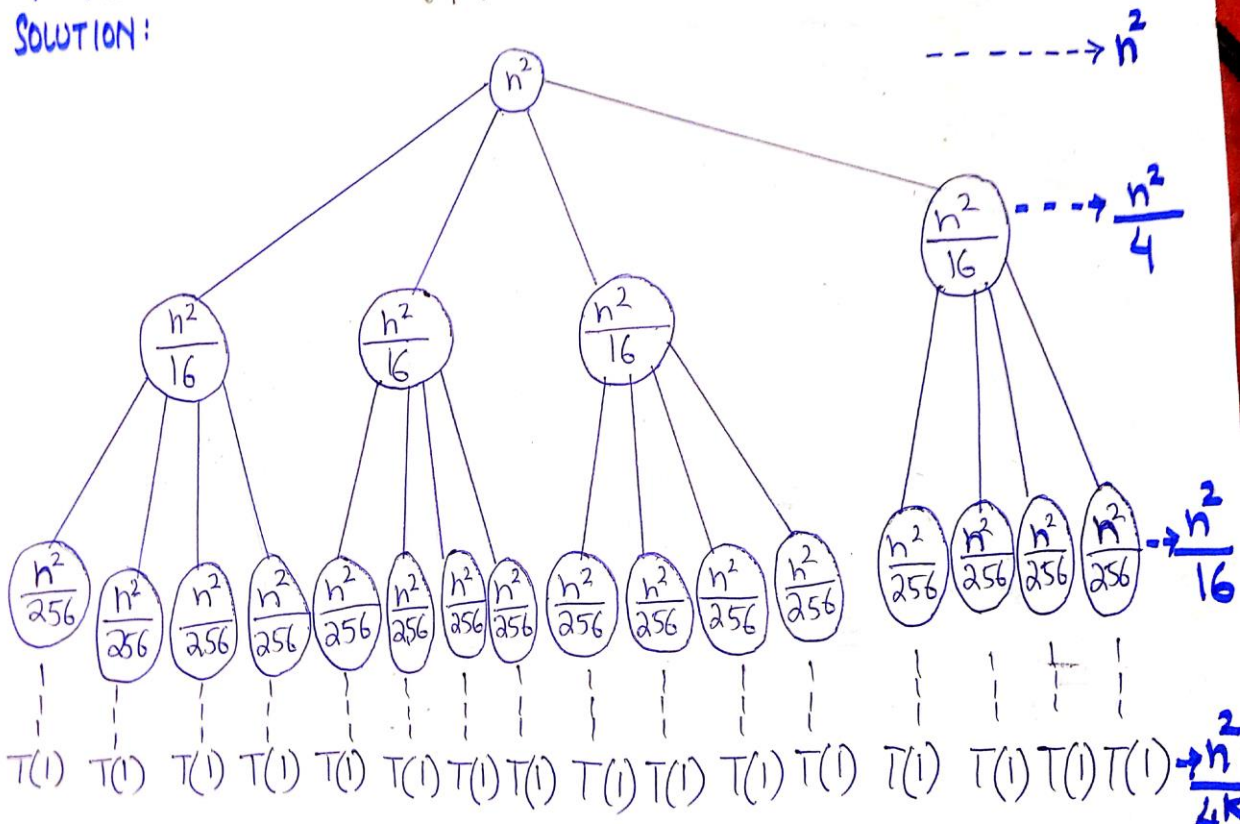
$$T(n) = 4^{\log_4 n} \cdot T(1) + \frac{4}{3}n^2$$

$$T(n) = n + \frac{4}{3}n^2$$

Hence, $\Theta(n^2)$ is the time complexity.

QUESTION "b": $T(n) = 4T\left(\frac{n}{4}\right) + n^2$

SOLUTION:



- Height of left tree:

For ~~$T(n) = 1$~~ $T(1) = \frac{n}{4^K} = 1$

$\Rightarrow K = \log_4 n$

~~$\Rightarrow K = \log_{16} n$~~

- Height of right tree:

For ~~$T(1) \Rightarrow \frac{n}{16^K} = 1$~~

~~$\Rightarrow K = \log_{16} n$~~

- Height of left tree:

For $T(1) \Rightarrow \frac{n}{4^K} = 1$

$\Rightarrow K = \log_4 n$

- Height of right tree:

For $T(1) \Rightarrow \frac{n}{4^K} = 1$

$\Rightarrow K = \log_4 n$

Sum of recurrences:

$$= n^2 + \frac{n^2}{4} + \frac{n^2}{16} + \cdots + \frac{n^2}{4^K}$$

$$= n^2 \left[1 + \frac{1}{4} + \frac{1}{16} + \cdots + \frac{1}{4^K}\right]$$

Using the property of Geometric Series:

$$= n^2 \left[\frac{1}{1 - \frac{1}{4}}\right] \Rightarrow \frac{4}{3} n^2$$

Hence, $\Theta(n^2)$ is the time complexity.

Question # 4 **[1 mark]**

Consider the given recurrence relation. You need to apply **Substitution Guess and Test method** by assuming given guesses to find correct one.

$$\boldsymbol{T(n) = 3T\left(\frac{n}{2}\right) + n^2}$$

Guess 1 : $\boldsymbol{T(n) = O(n)}$ **, Guess 2 :** $\boldsymbol{T(n) = O(n^2)}$

Solution

Guess 1: $\boldsymbol{T(n) = O(n)}$

$$T(n) \leq cn$$

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)$$

$$T(n) = \leq 3\frac{cn}{2} + n^2 \leq 1.5\ cn + n^2$$

Our Guess was wrong There is no C>1 exist to prove $1.5\ cn + n^2 = O(n)$

Guess 2: $\boldsymbol{T(n) = O(n^2)}$

$$T(n) \leq cn^2$$

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 \quad so\ we\ get: T\left(\frac{n}{2}\right) \leq cn^2/4$$

$$T(n) \leq 3cn^2/4 + n^2$$

$$T(n) \leq 0.75cn^2 + n^2 \leq cn^2\ for\ c \geq 4$$

So $O(n^2)$ guess is correct for $c \geq 4$

For finding c, you can use this equation:

$$0.75cn^2 + n^2 \leq cn^2$$

$$n^2 \leq cn^2 - 0.75cn^2$$

$$n^2 \leq 0.25cn^2$$

$$c \geq 4$$

Question # 5 **[1.5 marks]**

Consider below algorithm which solves the following :
"Find the middle of the linked list while only going the linked once"

```
GetMiddle (List l){
  pSlow = pFast = l;
  while ((pFast->next)&&(pFast->next->next)){
    pFast = pFast->next->next
    pSlow = pSlow->next
  }
  return pSlow
}
```

Invariant Property: At the start of the i-th iteration of the while loop, "pSlow" points to the i-th element in the list and "pFast" points to the 2i-th element.

Prove given invariant property for above pseudocode.

Solution

- *Invariant: At the start of the i-th iteration of the while loop, pSlow points to the i-th element in the list and pFast points to the $2i$-th element*
- **Initialization:** True when $i = 0$ since both pointers are at the head
- **Maintenance:** if pSlow, pFast are at positions i and $2i$ respectively before i-th iteration, they will be at positions $i+1$, $2(i+1)$ respectively before the $i+1$-st iteration
- **Termination:** When the loop terminates, pFast is at element $n-1$. Then by the loop invariant, pSlow is at element $(n-1)/2$. Thus pSlow points to the middle of the list

Question # 6 **[1 + 0.5 + 0.5 + 1.5 = 3.5 marks]**

Let $A[1 \ldots . . n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$ then the pair (i, j) is called an **inversion** of A.

a) List the five inversions of the array ⟨2, 3, 8, 6, 1⟩. (Recall that inversions are specified by indices rather than array values)
b) What array with elements from the set {1, 2, ...,n} has the most inversions? How many does it have?
c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.
d) Give an algorithm *(you may write algorithm in english)* that determines the number of inversions in any permutation on n elements in $\Theta(nlogn)$ worst-case time. (Hint : Modify merge sort)

Solution

Part (a): Given the list $< 2, 3, 8, 6, 1 >$ we find that the inversions in it are

$$(1, 5)\,, (2, 5)\,, (3, 4)\,, (3, 5)\,, (4, 5)\,.$$

Recall that inversions are specified by the *indices* rather than the array values.

Part (b): The list with the most inversions is $< n, n-1, n-2, \cdots, 3, 2, 1 >$. This list has

$$\begin{aligned}(n-1)+(n-2)+\cdots+2+1 &= \sum_{k=1}^{n-1} k = \sum_{k=1}^{n} k - n \\ &= \frac{n(n+1)}{2} - n = \frac{n}{2}(n+1-2) \\ &= \frac{n(n-1)}{2} = \theta(n^2)\,.\end{aligned}$$

Part (c): Every inversion must be "undone" by a step in the insertion sort inner while loop and thus we expect that the running time of insertion sort to be proportional to the number of inversions.

Part (d): Following the hint given in the book we will attempt to implement a divide-and-conquer merge sort type algorithm to count the number of inversion in an input array. If we take the merge sort algorithm as a starting point then when after splitting the array A into two parts we see that the number of inversion in the original array A will be the number of inversions found in the left-half plus the number of inversions found in the right-half plus the number of inversions involving elements in the left and right halves.

In each of the recursive calls lets assume that our algorithm returns the number of inversions in the left and right subarrays. We then have to implement a merge like procedure that counts the number of inversions between the two left and right arrays. The big leap to understanding this problem is to notice that if the left and right subarrays are returned *sorted* (along with the number of inversions each subarray contained) we can easily count the number of "cross inversions" when we merge the two arrays.

Assume that we have two sorted arrays L (for "left") and R (for "right") and we will merge these two into an output sorted array as in merge sort. In that procedure if we are placing an element of L into the output array then this element is smaller than all other elements in L and is also smaller than all other elements in R. Thus this operation does not reveal an inversion between elements in L and R. On the other hand, if we place an element from R into the output array then this element must be smaller than all the remaining unplaced elements in the L array.

If there are m remaining elements in L we have "found" m cross inversions. As we continually place elements from L and R onto the output array each time we place one from R we add m to the number of cross inversions found. At this point the pseudocode for this procedure can be written as

```
COUNT-INVERSIONS-N-SORT(A, p, r)
1  ▷ Returns the number of inversions and A sorted between the two indices p and r
2  if p ≥ r
3      then return 0
4  q ← floor((p + r)/2)
5  inversions ← COUNT-INVERSIONS-N-SORT(A, p, q)
6  inversions ← inversions + COUNT-INVERSIONS-N-SORT(A, q + 1, r)
7  inversions ← inversions + COUNT-INVERSIONS-N-MERGE(A, p, q, r)
8  return inversions
```

```
COUNT-INVERSIONS-N-MERGE(A, p, q, r)
 1  ▷ the number of elements in the left array
 2  n_1 ← q − p + 1
 3  ▷ the number of elements in the right array
 4  n_2 ← r − q
 5  create arrays L[1 . . . n_1] and R[1 · · · n_2]
 6  for i ← 1 to n_1
 7        do L[i] ← A[p + i − 1]
 8  for j ← 1 to n_2
 9        do R[j] ← A[q + j]
10  i ← 1 ▷ holds the index into L
11  j ← 1 ▷ holds the index into R
12  k ← 1 ▷ holds the index into A
13  inversions ← 0
14  while i ≤ n_1 and j ≤ n_2
15        do if L[i] ≤ R[j]
16              then A[k] ← L[i]
17                   i ← i + 1
18              else  A[k] ← R[j]
19                    j ← j + 1
20                    ▷ Add in the inversion count
21                    inversions ← inversions +(n_1 − i + 1)
22           k ← k + 1
23  ▷ at this point one of L or R is now empty
24  while i ≤ n_1 ▷ never executed if the left array "runs out first"
25        do
26           A[k] ← L[i]
27           i ← i + 1
28           k ← k + 1
29  while j ≤ n_2 ▷ never executed if the right array "runs out first"
30        do
31           A[k] ← R[j]
32           j ← j + 1
33           k ← k + 1
```

From the above code we see that the running time of this algorithm on an input of size n can be written as

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)\,.$$

This has the solution $T(n) = n\lg(n)$ as we were to show.

Appendix

Masters Theorem 4th Case

If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$

Mathematical Properties :

$$\sum_{n=1}^{\infty} a_1(r)^{n-1} = \frac{a_1}{1-r}, \quad \sum_{i=0}^{k-1} a^i = \frac{1-a^i}{1-a}$$

BEST OF LUCK