Cairo University
Faculty of Computers and Artificial Intelligence

SCS316-Algorithms Analysis and Design
Assignment - 1

Supervised By:
DR. Zeinab Abd ElHaliem
TA. Dalia Maher Mohamed

Prepared & Implemented By:

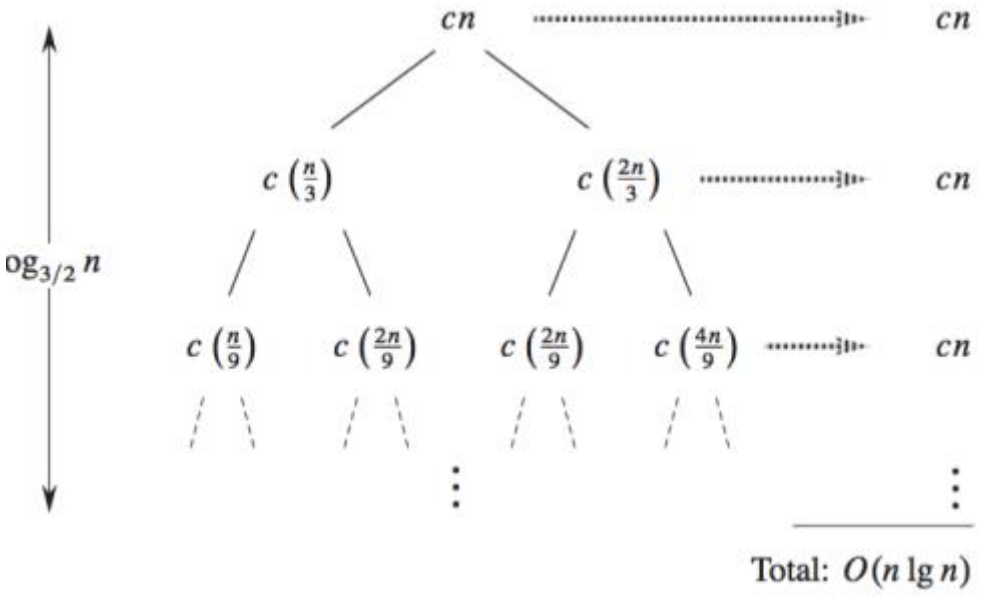
Mootaz Medhat Ezzat Abdelwahab (20206074)

Academic Year: 2023/2024

Second Semester

SCS316-Algorithms Analysis and Design **[Assignment - 1]**

➢ Problem 1



Let c represent the constant factor in the O(n) term.

■ **Write Recurrence relation:**

$$T(n) = T(n/3) + T(2n/3) + cn$$

■ **Verify that the upper bound is O(n log n) using the substitution method:**

$T(n) = T(n/3) + T(2n/3) + cn$ ⇒ **First iteration**

$\because T(n/3) = T(n/9) + T(2n/9) + \frac{1}{3}cn$

$\because T(2n/3) = T(2n/9) + T(4n/9) + \frac{2}{3}cn$

$\therefore T(n) = T(n/9) + T(2n/9) + \frac{1}{3}cn + T(2n/9) + T(4n/9) + \frac{2}{3}cn + cn$

$\therefore T(n) = T(n/9) + 2T(2n/9) + T(4n/9) + 2cn$ ⇒ **Second iteration**

$\because T(n/9) = T(n/27) + T(2n/27) + \frac{1}{9}cn$

$\because T(2n/9) = T(2n/27) + T(4n/27) + \frac{2}{9}cn$

$\because T(4n/9) = T(4n/27) + T(8n/27) + \frac{4}{9}cn$

$$\therefore T(n) = T(n/27) + T(2n/27) + \frac{1}{9}cn + 2T(2n/27) + 2T(4n/27) + \frac{4}{9}cn +$$

$$T(4n/27) + T(8n/27) + \frac{4}{9}cn + 2cn$$

$$\therefore T(n) = T(n/27) + 3T(2n/27) + 3T(4n/2\ 7) + T(8n/27) + 3cn$$

⇒ **Third iteration**

⇒ From iteration one, two and three and because we want to get the upper bound, we will take the recursion trees with the longest height $T(2n/3)$, $T(4n/9)$, $T(8n/27)$:

$$\therefore T(n) = T(2n/3) + cn \quad \Rightarrow \quad T(n) = T(n/1.5) + cn \quad \Rightarrow First\ iteration$$

$$\therefore T(n) = T(4n/9) + 2cn \quad \Rightarrow \quad T(n) = T(n/2.25) + 2cn \quad \Rightarrow Second\ iteration$$

$$\therefore T(n) = T(8n/27) + 3cn \quad \Rightarrow \quad T(n) = T(n/3.375) + 3cn \quad \Rightarrow Third\ iteration$$

$$\therefore T(n) = T\left(n/1.5^{k}\right) + kcn \quad \Rightarrow \quad \frac{n}{1.5^{k}} = 1 \quad \Rightarrow \quad k = log_{1.5}n$$

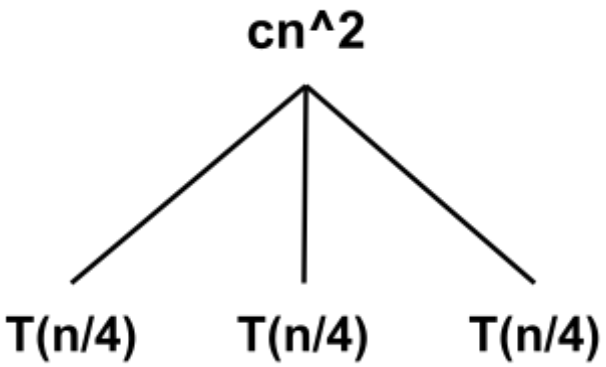
$$\therefore T(n) = T\left(n/1.5^{log_{1.5}n}\right) + log_{1.5}n\, cn \quad \Rightarrow \quad T(n) = T(1) + n\, log_{1.5}n$$

$$\therefore T(n) = n\, log_{1.5}n \quad \Rightarrow \quad O\left(n\, log_{1.5}n\right)$$

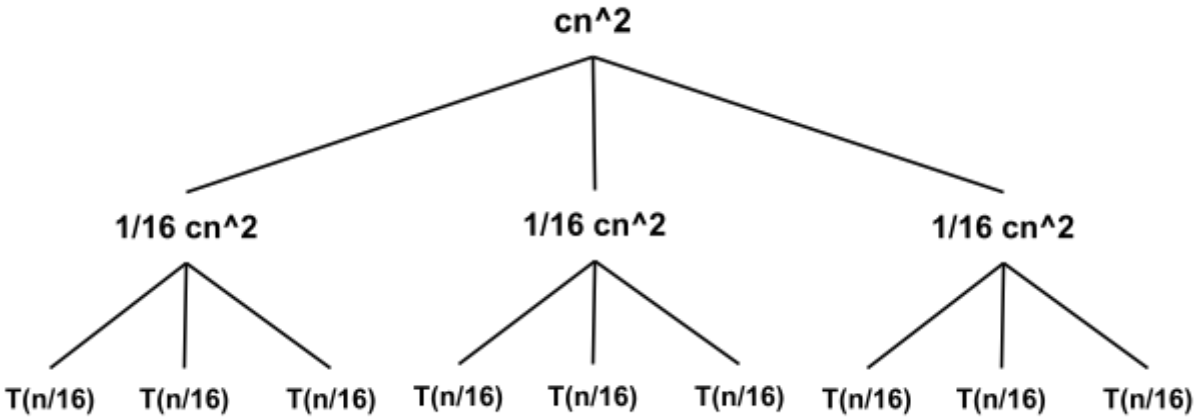
➢ Problem 2

Construct a recursion tree for the recurrence T(n) = 3T(n/4) + C^2 and solve it using the iterative method.

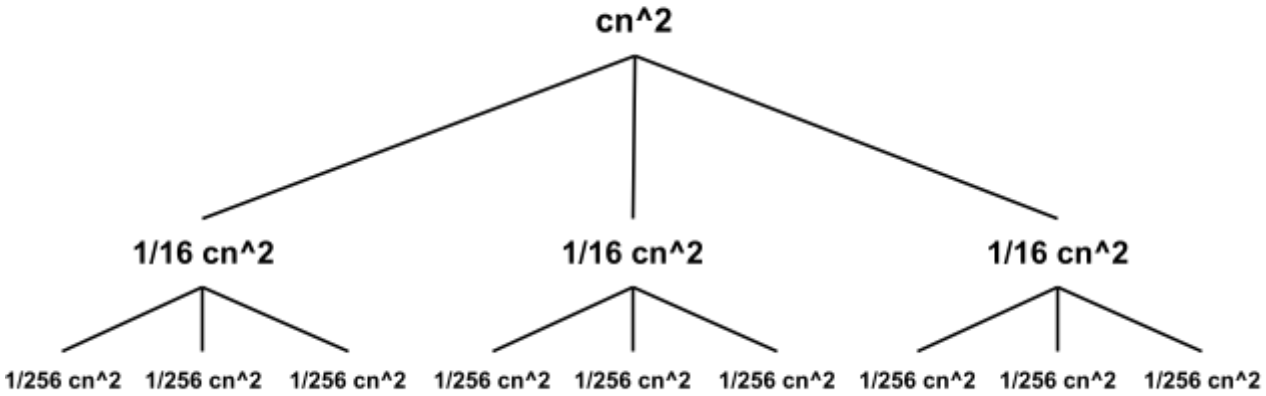
■ Recursion Tree:



T(n/4) = 3 T(n/16) + 1/16 cn^2



T(n/16) = 3 T(n/64) + 1/256 cn^2



$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots$$

$$height\ of\ tree = \log_4 n$$

■ **Solving using the iterative method:**

$T(n) = 3\,T(n/4) + cn^2$ ⇒ **First iteration**

$\because T(n/4) = 3\,T(n/16) + \frac{1}{16}cn^2$

$\therefore T(n) = 3 * (3\,T(n/16) + \frac{1}{16}cn^2) + cn^2$

$\therefore T(n) = 9\,T(n/16) + \frac{3}{16}cn^2 + cn^2$

$\therefore T(n) = 9\,T(n/16) + (\frac{3}{16} + 1)\,cn^2$ ⇒ **Second iteration**

$\because T(n/16) = 3\,T(n/64) + \left(\frac{1}{16}\right)^2 cn^2$

$\therefore T(n) = 9 * (3\,T(n/64) + \left(\frac{1}{16}\right)^2 cn^2) + cn^2$

$\therefore T(n) = 27\,T(n/64) + \left(\frac{3}{16}\right)^2 cn^2 + cn^2$

$\therefore T(n) = 27\,T(n/64) + \left(\left(\frac{3}{16}\right)^2 + 1\right)cn^2$ ⇒ **Third iteration**

⇒ From iteration one, two and three: $T(n) = 3^k\,T\left(n/4^k\right) + \left(\left(\frac{3}{16}\right)^k + 1\right)cn^2$

$\because \frac{n}{4^k} = 1 \quad \Rightarrow \quad k = log_4 n$

$\therefore T(n) = 3^{log_4 n}\,T(1) + \left(\left(\frac{3}{16}\right)^{log_4 n} + 1\right)cn^2$

$\therefore T(n) = n^{log_4 3} * 1 + \left(n^{log_4 \frac{3}{16}} + 1\right)cn^2$

$\therefore T(n) = n^{log_4 3} + \left(n^{log_4 \frac{3}{16}} + 1\right)cn^2 \quad \Rightarrow \quad O\left(n^2\right)$

➢ Problem 3

Solve the following recurrence relation using the master method:

$$\text{T(0) = c1 , T(1) = c2 , } T(n) = 2^k\, T\left(n/2^k\right) + \left(2^k - 1\right) c3$$

■ **We will solve this recurrence relation using master method and by substitution with T(0) = c1, T(1) = c2 in** $T(n) = 2^k\, T\left(n/2^k\right) + \left(2^k - 1\right) c3$:

➥ **Using master method :-**

$$\because a = 2^k, \quad b = 2^k, \quad d = 0$$

$$\therefore n^{\log_b a} \; [\;] \; n^d \;\Rightarrow\; n^{\log_{2^k} 2^k} > n^0 \;\Rightarrow\; n^1 > n^0 \;\Rightarrow\; T(n) \in O(n)$$

➥ **By substitution with T(0) = c1, T(1) = c2 in** $T(n) = 2^k\, T\left(n/2^k\right) + \left(2^k - 1\right) c3$

$$\because \frac{n}{2^k} = 1 \;\Rightarrow\; n = 2^k \;\Rightarrow\; Take\ the\ logarithm\ to\ base\ 2\ on\ both\ sides$$

$$\therefore \log_2 n = \log_2 2^k \;\Rightarrow\; \log_2 n = k \;\Rightarrow\; k = \log n$$

$$\therefore T(n) = 2^{\log_2 n}\, T\left(n/2^{\log_2 n}\right) + \left(2^{\log_2 n} - 1\right) c3$$

$$\therefore T(n) = n^{\log_2 2}\, T\left(n/n^{\log_2 2}\right) + \left(n^{\log_2 2} - 1\right) c3$$

$$\therefore T(n) = n\, T(n/n) + (n - 1)\, c3$$

$$\therefore T(n) = n\, T(1) + (n - 1)\, c3$$

$$\therefore T(n) = nc2 + (n - 1)\, c3$$

$$\therefore T(n) \in O(n)$$

Divide and Conquer Problems:
Solve all the following problems using divide and conquer strategy.

➢ Problem 4

Given an array of n elements A[0: n-1] that may contain positive and negative numbers.

■ Find the contiguous subarray of numbers which has the largest sum:

For this problem there are 2 subarrays of size $(n/2)$ that are been divided recursively in each iteration. Then there is $\Theta(n)$ work of eliminating the values that do not satisfy the largest sum range of the given array.

■ Using the previous data will lead us to the recurrence relation below:

⇒ **2 subarrays.**

⇒ $(\frac{n}{2})$ **subarray size.**

⇒ $\Theta(n)$ **eliminating the values.**

$$T(n) = 2T(n/2) + \Theta(n)$$

And using master method we can solve this relation and get time complexity:-

$$\because a = 2, \quad b = 2, \quad d = 1$$

$$\therefore n^{\log_2 2} = n^1 \Rightarrow n^1 = n^1 \Rightarrow O(n^d \log_b n) \Rightarrow O(n \log n)$$

➢ Problem 5

Find the multiplication matrix of 2 square matrices A and B of size n*n each.

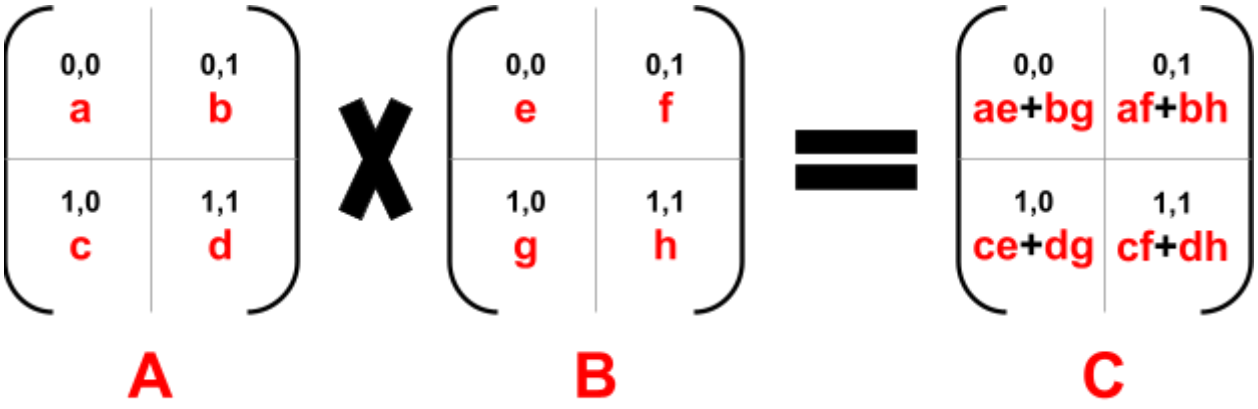
Using simple divide and conquer strategy to multiply two 2 square matrices

■ **First :-**

We will divide the 2 square matrices A and B into 4 sub-matrices of size n/2 * n/2

⇒ a, b, c and d are sub-matrices of A, of size n/2 * n/2

⇒ e, f, g and h are sub-matrices of B, of size n/2 * n/2



■ **Second:-**

We will calculate the values recursively

⇒ ae + bg, af + bh, ce + dg and cf + dh

➥ **To get multiplication matrix of 2 square matrices A and B**

⇒ we do 8 multiplications for matrices of size n/2 * n/2

⇒ 4 additions for matrices of size n/2 * n/2.

⇒ addition of two matrices takes $O(n^2)$ time.

➥ **So T(n) can be expressed as follows:-**

$$T(n) = 8\,T(n/2) + O(n^2)$$

And using master method we can solve this relation and get time complexity:-

$$\because a = 8, \quad b = 2, \quad d = 2$$

$$\therefore n^{\log_2 8} > n^2 \quad \Rightarrow \quad n^3 > n^2 \quad \Rightarrow \quad O(n^3)$$

➢ Problem 6

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision.

Recall the following formula for distance between two points p and q.

Euclidean distance d(p , q) = sqrt[(qx − px)^2 + (qy − py)^2]

The Brute force solution is O(n^2), compute the distance between each pair and return the smallest.

■ Find the smallest distance in O(nLogn) time using Divide and Conquer strategy

To use divide and conquer strategy and make algorithm complexity O(nLogn)

➥ We will divide all points in two sets and recursively call two sets.
➥ After dividing all points in two sets, it finds the strip in O(n) time.
➥ It takes O(n) time to divide the Py array around the mid vertical line
➥ Finally finds the closest points in strip [ ] in O(n) time.

■ So T(n) can be expressed as follows:-

$$T(n) = 2\,T(n/2) + O(n) + O(n) + O(n)$$
$$T(n) = 2\,T(n/2) + 3\,O(n)$$

And using master method we can solve this relation and get time complexity:-

$$\because a = 2, \quad b = 2, \quad d = 1$$
$$\therefore n^{\log_2 2} = n^1 \quad \Rightarrow \quad O(n^d \log_b n) \quad \Rightarrow \quad O(n \log n)$$

➢ Problem 7

Consider an unsorted array with N number of elements and given number k value in range from 1 to N; we have to find the kth largest element in the best possible way which take time complexity O(n)

■ **Example:**

Input:

K = 3

A[ ] = {14, 5, 6, 12, 14, 8, 10, 6, 25}

Output: Kth largest element = 12

Input:

K = 5

A[ ] = {18, 15, 3, 1, 2, 6, 2, 18, 16}

Output: Kth largest element = 3

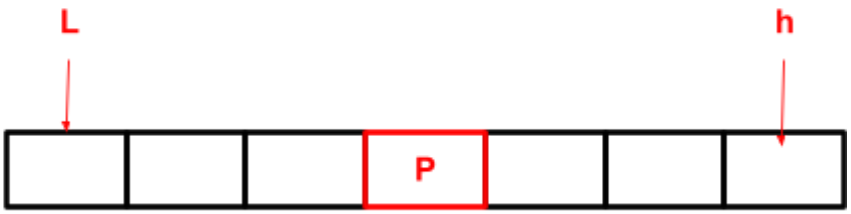
■ **In this problem we will use a method similar to quickSort ( ). But we get worst-case linear time by selecting a pivot that divides the array in a balanced way.**

■ **After the array is divided in a balanced way, we will apply the same steps as used in quickSort() to decide which to go left or right of the pivot.**

```
int partition(int arr[], int l, int h, int x)
{
    // Search for x in arr[l..h] and move it to end
    int i;
    for (i = l; i < h; i++)
        if (arr[i] == x)
          break;

    swap(arr[i], arr[h]);

    i = l;
    for (int j = l; j <= h - 1; j++)
        if (arr[j] <= x)
        {
            swap(arr[i], arr[j]);
            i++;
         }
    }
    swap(arr[i], arr[h]);
    return i;
}
```



➢ Problem 8

Given an array of integers. Find the Inversion Count in the array.

Inversion Count: For an array, inversion count indicates how far (or close) the array is from being sorted. If array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum. Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

Input: N = 5, arr[ ] = {2, 4, 1, 3, 5}

Output: 3

Explanation: The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

Input: N = 5, arr[ ] = {2, 3, 4, 5, 6}

Output: 0

Explanation: As the sequence is already sorted so there is no inversion count.

Input: N = 3, arr[ ] = {10, 10, 10}

Output: 0

Explanation: As all the elements of array are same, so there is no inversion count.

➥ In this problem we will use merge sort as divide and conquer based algorithm to get O(n log n) time complexity as follow:-

```
while (i < sizeOfLeftSubArray && j < sizeOfRightSubArray)
{
   if (leftSubArray[i] <= rightSubArray[j])
   {
      arr[k] = leftSubArray[i];
      i++
   }
   else
   {
      arr[k] = rightSubArray[j];
      j++;
      inversionCount += (middleIndex - i + 1);
   }
   k++;
}
```