

Course Code: CS-2009	Course Name: Design and Analysis of Algorithm
Instructors: Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Dr. Farrukh Salim, Dr. Nasir Uddin, Mr. Faisal Ahmed, Mr. Sandesh Kumar, Ms. Ansum Hamid, Mr. Syed Faisal Ali& and Mr. Minhaz Raza	
Student Roll No:	Section: BCS-5A, B, C, D, E, F, G, H, J, K, BSE-5A, B, BAI-5A, B, BCY-5A, B

Instructions:

- Return the question paper
- Read each question completely before answering it. There are **5 questions** on **2 pages**
- In case of any ambiguity, you may make assumptions. But your assumption should not contradict any statement in the question paper

Time: 60 Minutes

Max Marks: 12.5

Question#1

(2 Marks, CLO-1)

Imagine a scenario where we have two different computers, Computer A and Computer B, both tasked with sorting a large dataset of one million records. Computer A has a processing speed of 500 million instructions per second (500 MIPS), while Computer B operates at an impressive 2 billion instructions per second (2 GIPS).

Now, let's add an element of friendly competition to the mix. Computer A and Computer B are participating in a sorting challenge. Computer A decides to use Bubble Sort, an algorithm known to require " n^2 " instructions for sorting n records. On the other hand, Computer B chooses Heap Sort but uses an implementation with a high-level language and an inefficient compiler, resulting in code that demands " $30 \times n \times \log n$ " instructions for sorting. With this competition set up, let's explore which computer and sorting algorithm combination outperforms the other when sorting this substantial dataset of one million records.

Q1

Marking Scheme: Correct 2 Points with clear formulas (Calculation mistakes can be ignored); Partially Correct with good attempt e.g numerator is correct but mistake in denominator: 1.0-1.75 ; Answer given in words only: 0- 0.75;

Solution:

Computer A:-

$$\frac{(10^6)^2}{560 \times 10^6} = 2000 \text{ Second} \times \frac{\frac{1}{2} \text{ hour}}{0.5 \text{ hour}}$$

Computer B:-

$$\frac{30 \cdot 10^4 \log 10^8}{2 \times 10^7} = 0.2989 \approx 0.3 \text{ Seconds.}$$

Computer B 6691 times faster than Computer A.

$$\text{Ratio} = \frac{2000}{0.2989} = 6691.20$$

Question#2**(4 Marks, CLO-1)**

You're consulting for a small computation-intensive investment company, and they have the following type of problem that they want to solve over and over. A typical instance of the problem is the following. They're doing a simulation in which they look at n consecutive days of a given stock, at some point in the past. Let's number the days $i = 1, 2, \dots, n$; for each day i , they have a price $p(i)$ per share for the stock on that day. (We'll assume for simplicity that the price was fixed during each day.) Suppose during this time period, they wanted to buy 1,000 shares on someday and sell all these shares on some (later) day. They want to know: When should they have bought and when should they have sold in order to have made as much money as possible? (If there was no way to make money during the n days, you should report this

instead.) For example, suppose $n = 4$, $p(1) = 9$, $p(2) = 1$, $p(3) = 5$, $p(4) = 3$. Then you should return "buy on 2, sell on 3" (buying on day 2 and selling on day 3 means they would have made \$4 per share, the maximum possible for that period). Design

- (a) Algorithm to solve about a problem using $O(n^2)$ [1 Point]
- (b) Algorithm to solve the above problem using $O(n \log n)$ [1.5 Points]

You can explain designing in simple sentences but must be clear and all scenarios should be covered. Use $n = 8$, $p(1) = 2$, $p(2) = 9$, $p(3) = 3$, $p(4) = 8$, $p(5) = 11$, $p(6) = 1$, $p(7) = 6$, $p(8) = 6$ to check whether result is correct from your design.

Q2:

1.5 Points for $O(n^2)$ correct solution i.e. by using Brute Force with example; 1.25 Points without Dry Run; 0-1 for partial solution

2.5 Points $O(n \log n)$ for correct solution using Maximum Subarray Solution with Dry Run

2.0 Points for correct solution without Dry Run

1-1.75 for Partial Correct Solution

0-1 for some attempt

Solution:

Brute and Force Approach $O(n^2)$

Maximum Subarray i.e. Divide & Conquer Solution $O(n \log n)$

A natural approach would be to consider the first $n/2$ days and the final $n/2$ days separately, solving the problem recursively on each of these two sets, and then figure out how to get an overall solution from this in $O(n)$ time. This would give us the usual recurrence $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n)$, and hence $O(n \log n)$.

Also, to make things easier, we'll make the usual assumption that n is a power of 2. This is no loss of generality: if n' is the next power of 2 greater than n , we can set $p(i) = p(n')$ for all i between n and n' . In this way, we do not change the answer, and we at most double the size of the input (which will not affect the $O()$ notation).

Now, let S be the set of days $1, \dots, n/2$, and S' be the set of days $n/2 + 1, \dots, n$. Our divide-and-conquer algorithm will be based on the following observation: either there is an optimal solution in which the investors are holding the stock at the end of day $n/2$, or there isn't. Now, if there isn't, then the optimal solution is the better of the optimal solutions on the sets S and S' . If there is an optimal solution in which they hold the stock at the end of day $n/2$, then the value of this solution is $p(j) - p(i)$ where $i \in S$ and $j \in S'$. But this value is maximized by simply choosing $i \in S$ which minimizes $p(i)$, and choosing $j \in S'$ which maximizes $p(j)$.

Thus our algorithm is to take the best of the following three possible solutions.

- The optimal solution on S .
 - The optimal solution on S' .
 - The maximum of $p(j) - p(i)$, over $i \in S$ and $j \in S'$.
-

The first two alternatives are computed in time $T(n/2)$, each by recursion, and the third alternative is computed by finding the minimum in S and the

maximum in S' , which takes time $O(n)$. Thus the running time $T(n)$ satisfies

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n),$$

as desired.

Question#3**(1.5 Marks, CLO-2)**

Compute the running time for the following code fragment. Express it in the term of big O notation.

Solution:

Marking Scheme:

1.5: Correct Solution (Some mistakes in individual steps can be ignored e.g. n instead of n+1)

1.0 – 1.25: Partially current solution in most of the lines

0 – 0.75: Some attempt for some line of code

Direct Correct Answer without steps: 0.75 only

Ans 1:

The running time is:

- 1 declaration (int arr[n])
- 2 assignment (int i=0)
- $2(n+1)$ comparisons ($i \leq n$)
- $2n$ increments ($i++$)
- n indirect assignment ($arr[i]=i$)
- n indirect assignment ($int j=n*n$)
- $n(\log_2 n + 2) = 2n \log(n) + 2n$ comparison ($j > 0$)
- $n(2 \log(n) + 1)$ indirect assignment ($arr[i]=arr[i]+1$)
- $n(2 \log(n) + 1)$ indirect assignment ($j=(int)(j/2)$)

Hence:

$$T(n) = 1(a) + 2(b) + 2(n+1)(c) + 2n(d) + n(e) + n(f) + (2n \log n + 2n)(g) + (2n \log(n) + n)(h) + (2n \log(n) + n)(i)$$

Where a, b, c, d, e, f, g, h and i are machine-dependent constants representing the different

operations mentioned above. Since the highest order the term in above expression is $n \log(n)$. Therefore $T(n)$ is $O(n \log n)$

Question#4**[1 Mark, CLO-2]**

Solve the following recurrence equations.

PART 4A) Solve the following using the master method

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

[0.5 Mark]**Solution:****Marking Scheme: Strict Marking. Case is correctly identified Either 0 or 0.5 for both**We can see that $T(n)$ is of the form $T(n) = aT(n/b) + f(n)$, $a = 3$, $b = 3$, $f(n) = n/2$ $\Rightarrow T(n) = \Theta(n \log n)$ (Case 2)

B) $T(n) = 4T\left(\frac{n}{2}\right) + n^3(\log_2 n)$

[0.5 Mark]**Solution:**

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3(\log_2 n)$$

$$\text{in the form } T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$a = 4, b = 2, \text{ and } f(n) = n^3(\log_2 n)$$

$$\text{compare } n^{\log_b a} \text{ with } f(n)$$

$$n^{\log_b a} = n^{\log_2 4} = n^2 << f(n)$$

so, using case 3 of master's theorem

$$T(n) = \Theta(n^3 \log_2 n)$$

PART 4B) Solve the following using iterative substitution

[3 Marks, CLO-2]

$$A) T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$$

[1.5 Mark]

Marking Scheme:

1.5 for correct answer with some minor errors

0.5-1.25 for partial attempt

ITERATIVE SUBSTITUTION

$$1) T(n) = \begin{cases} 1 & n=0 \\ 2T(n-1)+1 & n>0 \end{cases}$$

$$T(n) = 2T(n-1) + 1 \rightarrow \textcircled{1}$$

$$T(n-1) = 2T(n-2) + 1$$

$$T(n) = 2[2T(n-2) + 1] + 1 -$$

$$= 2^2 T(n-2) + 2 + 1 \rightarrow \textcircled{2}$$

$$T(n-2) = 2T(n-3) + 1$$

$$\begin{aligned} T(n) &= 2^2 [2T(n-3) + 1] + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \rightarrow \textcircled{3} \end{aligned}$$

Generalizing eq (3)

$$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 1 \rightarrow (4)$$

Assume $n-k=0$ Since it is a decreasing function.

$$\Rightarrow n-k=0, \quad n=k$$

Substituting in eq (4)

$$T(n) = 2^n T(0) + 1 + 2 + \dots + 2^{k-1}$$

$$T(n) = 2^n (1) + 2^k - 1$$

$$= 2^n (1) + 2^n - 1$$

$$= 2^{n+1} - 1$$

$$= O(2^n)$$

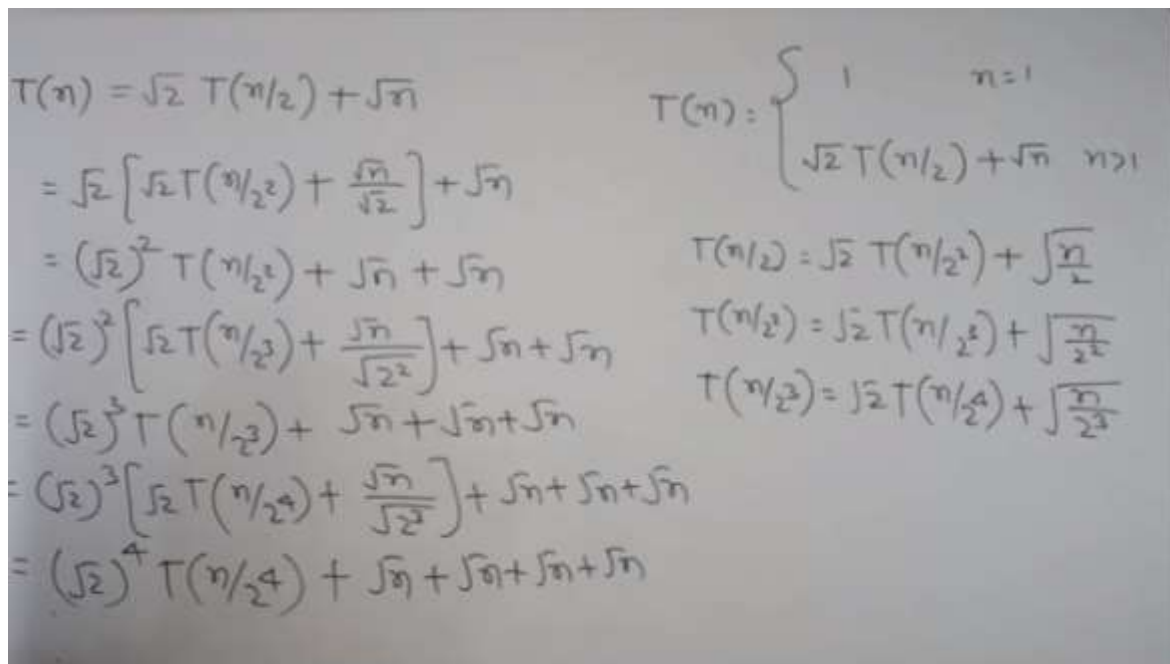
$$B) T(n) = \begin{cases} 1 & n = 1 \\ \sqrt{2}T\left(\frac{n}{2}\right) + \sqrt{n} & n > 1 \end{cases}$$

[1.5 Mark]

Proof for $O(n \log n)$ and for $O(1)$

Marking Scheme:

Since iterative method was also allowed, students can also attempt through this
Must show all steps of substitution method (Not shown below)



$$T(n) = \sqrt{2} T(n/2) + \sqrt{n}$$

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

$$= (\sqrt{2})^2 T(n/4) + \sqrt{n} + \sqrt{n}$$

$$= (\sqrt{2})^2 \left[\sqrt{2} T(n/8) + \frac{\sqrt{n}}{\sqrt{2^2}} \right] + \sqrt{n} + \sqrt{n}$$

$$= (\sqrt{2})^3 T(n/8) + \sqrt{n} + \sqrt{n} + \sqrt{n}$$

$$= (\sqrt{2})^3 \left[\sqrt{2} T(n/16) + \frac{\sqrt{n}}{\sqrt{2^3}} \right] + \sqrt{n} + \sqrt{n} + \sqrt{n}$$

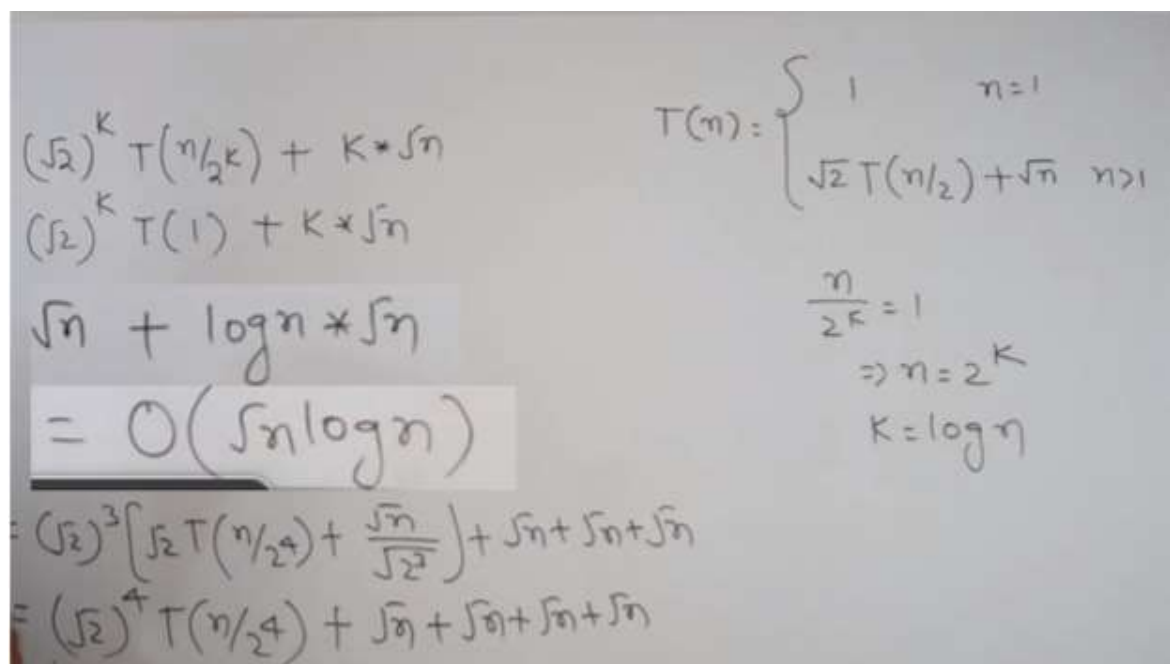
$$= (\sqrt{2})^4 T(n/16) + \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n}$$

$$T(n) = \begin{cases} 1 & n=1 \\ \sqrt{2} T(n/2) + \sqrt{n} & n>1 \end{cases}$$

$$T(n/2) = \sqrt{2} T(n/4) + \sqrt{\frac{n}{2}}$$

$$T(n/4) = \sqrt{2} T(n/8) + \sqrt{\frac{n}{4}}$$

$$T(n/8) = \sqrt{2} T(n/16) + \sqrt{\frac{n}{8}}$$



$$(\sqrt{2})^k T(n/2^k) + K * \sqrt{n}$$

$$(\sqrt{2})^k T(1) + K * \sqrt{n}$$

$$\sqrt{n} + \log n * \sqrt{n}$$

$$= O(\sqrt{n} \log n)$$

$$= (\sqrt{2})^3 \left[\sqrt{2} T(n/16) + \frac{\sqrt{n}}{\sqrt{2^3}} \right] + \sqrt{n} + \sqrt{n} + \sqrt{n}$$

$$= (\sqrt{2})^4 T(n/16) + \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n}$$

$$T(n) = \begin{cases} 1 & n=1 \\ \sqrt{2} T(n/2) + \sqrt{n} & n>1 \end{cases}$$

$$\frac{n}{2^k} = 1$$

$$\Rightarrow n = 2^k$$

$$k = \log n$$

$O(n \log n)$ is True and $O(1)$ is False

Question#5**[1 Mark, CLO-2]**

For each of the following questions, indicate whether it is T (True) or F (False).

Please note that you must have to justify your answer using some example e.g. assuming a function (where possible), otherwise, no marks will be rewarded.

Marking Scheme:

Correct Answer with steps: 0.5 each

Direct correct Answer without steps: 0.25 each

Incorrect Answer but with steps and correct understanding of Asymptotic Notations: 0.25 each

Incorrect Answer and without steps: 0 each

1. $5n \log_2 n + 9n = O(n \log_2 n)$ for $c = 8$, and $n_0 = 2$

[0.5 Mark]

Solution: False

find a c & n_0 such that for all $n \geq n_0$:

$$5n \log_2 n + 9n \leq c \cdot n \log_2 n$$

While finding c and n_0 , in

$$5n \log_2 n + 9n \leq 5n \log_2 n + 9n \log_2 n \quad \text{for all } n \geq 2$$

$$5n \log_2 n + 9n \leq 14n \log_2 n$$

So,

$$f(n) = 5n \log_2 n + 9n = O(n \log_2 n) \text{ [for } c \geq 14, \text{ for all } n \geq n_0 = 2]$$

thus, with $c=8$, this is not true.

2. $3n^2 + 5n + 7 = O(n^2 \log_2 n)$

[0.5 Mark]

Solution: True

As $3n^2 + 5n + 7 \leq cn^2 \leq n^2 \log_2 n$

So, let it prove for $3n^2 + 5n + 7 \leq cn^2$

Divide both sides by n^2 , we get:

$$3n^2/n^2 + 5n/n^2 + 7/n^2 \leq c \cdot n^2/n^2$$

$$3 + 5/n + 7/n^2 \leq c$$

If we choose n_0 equal to 1 then we have value of c

$$3 + 5 + 7 \leq c$$

$$c \geq 15$$

$$3n^2 + 5n + 7 \leq 15n^2 \text{ for } n \geq 1$$

Proved that $f(n) = 3n^2 + 5n + 7 = O(n^2)$ [for $c = 15$, $n_0 = 1$]

Since the above is proved for $O(n^2)$, it is true for higher complexity as well, that is $O(n^2 \log_2 n)$