Question 1 [10 Marks]

Describe an efficient algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. n integers are placed in array S. Algorithm must be written in pseudo-code form.
You are required to do the complexity analysis of your algorithm and if you use any extra memory space, you are required to do the complexity analysis of the extra storage as well.

Sort the element in S using merge sort
- Remove the last element from S. Let y be the value of the removed element.
- If S is nonempty, look for z=x-y in S using binary search.
- Is S contains such an element z, then stop, since we have found y and z such that x=y+z. Otherwise, repeat Step 2.
- If S is empty, then no two elements in S sum to x, return NIL.

The following algorithm solves the problem:

```
DETERMINE_x(S,x)
1. MERGE_SORT(S,1,length(S))
2. y=REMOVE_MAX(S)
3. WHILE (S) DO
4.       z= BINARY_SEARCH(S,x-y,1,length(S))
5.  IF y+z=x THEN
6.       return TURE
7.  ELSE y=REMOVE_MAX(S)
8.      return NIL
```

Note that when we consider an element of S during i-th iteration, we don't need to look at the elements that have already been considered in previous iterations.
Step 1 takes Θ (n lg n) time.
Step 2 takes O (1).
Step 4 requires at most lg n time.
Steps 3-7 are repeated at most n times.
Thus, the total running time of this algorithm is Θ (n lg n). The most time-consuming work in this algorithm is MERGE-SORT, which takes Θ (n lg n) time.

Question 2 [4+4+4+4=16 Marks]

a) Provide a worst-case asymptotic time complexity of the following algorithms by using a suitable asymptotic notation considering a nearest function. Assume that there are no errors/ bugs in the algorithms. Show the meaningful working behind your answer.

Code	Time Complexity

int sum = 0;
for (int i = 0; i < N; i++) {
if (i == N - 1) {
for (int j = 0; j < N; j++) {
sum++;
}
}
}
	O(N)

int sum = 0;
for (int i = 0; i < N * 2; i++) {
for (int j = 0; j < 10000; j++) {
for (int k = 0; k < j*j; k++) {
sum++;
}
}
}
	O(N)

int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
for (j = 2; j <= n; j = j * 2) {
k = k + n / 2;
}
}
	O(n log n)

j = 0;
while (j < n)
{ for (int i = 0; i < n; ++i)
for (int k = 0; k < i; ++k)
sum++;
j = j + 1;
}
	O(n^3)

Question 3 [6+8=14 Marks]

a. For each of the following recurrences, give an expression for the runtime T (n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply. [6 marks]

1. $T(n) = 4T(n/2) + n^2$

2. $T(n) = 16T(n/4) + n$

Q3 Solution

(a)

must apply & mention master theorm case for fcell

marks

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

Sol $n^2 \log n$

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

Sol n^2

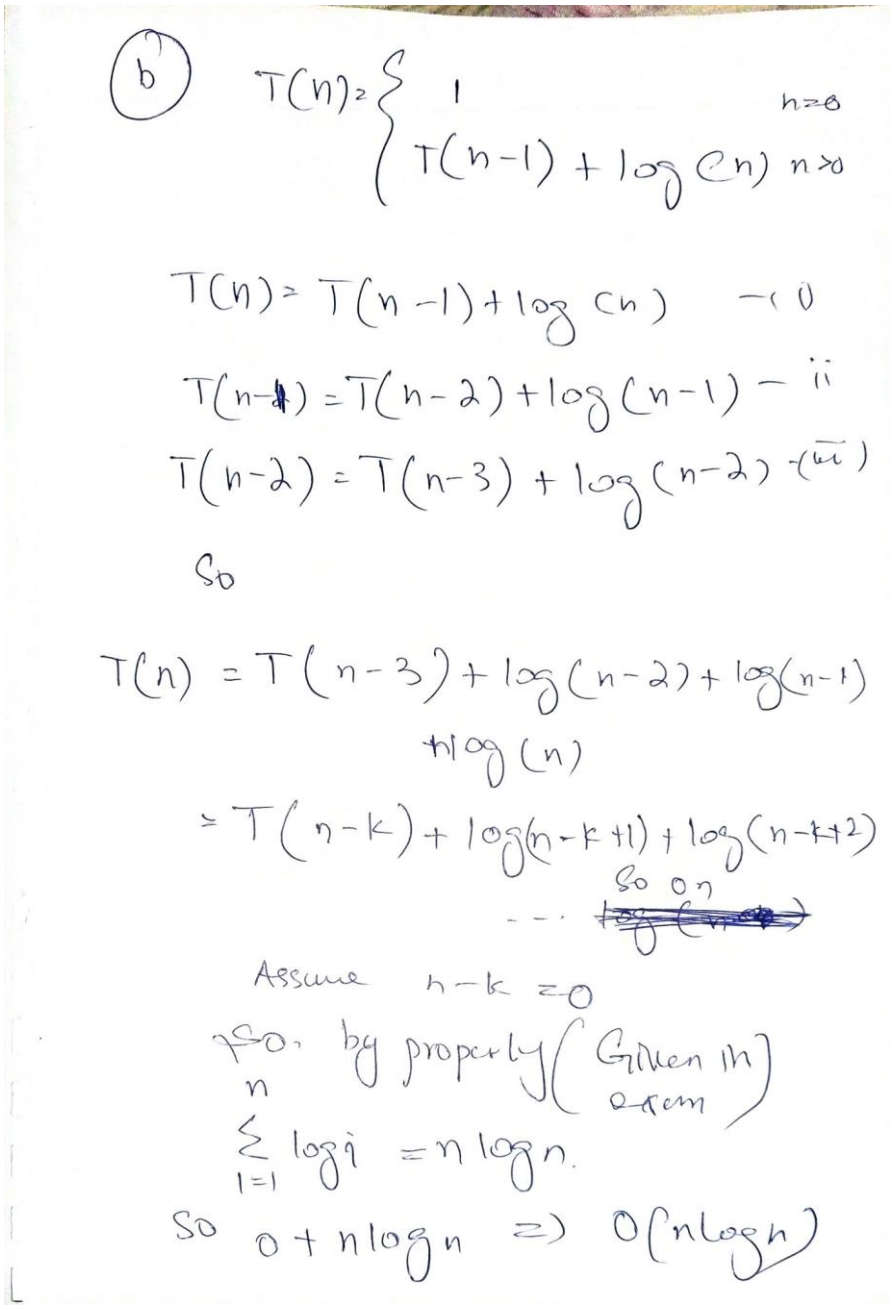
Correct = 3 marks

missing Theorm info = 1 mark

else zero marks

b. Use iteration method/substitution method to solve the following recursive relation [8 Marks]

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



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

$T(n) = T(n-1) + \log(n)$ — (i)

$T(n-1) = T(n-2) + \log(n-1)$ — (ii)

$T(n-2) = T(n-3) + \log(n-2)$ — (iii)

So

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

$= T(n-k) + \log(n-k+1) + \log(n-k+2)$ ... So on

Assume $n-k = 0$

So, by property (Given in question)

$\sum_{i=1}^{n} \log i = n \log n$

So $0 + n \log n \Rightarrow O(n \log n)$

Question 4 [10 Marks]

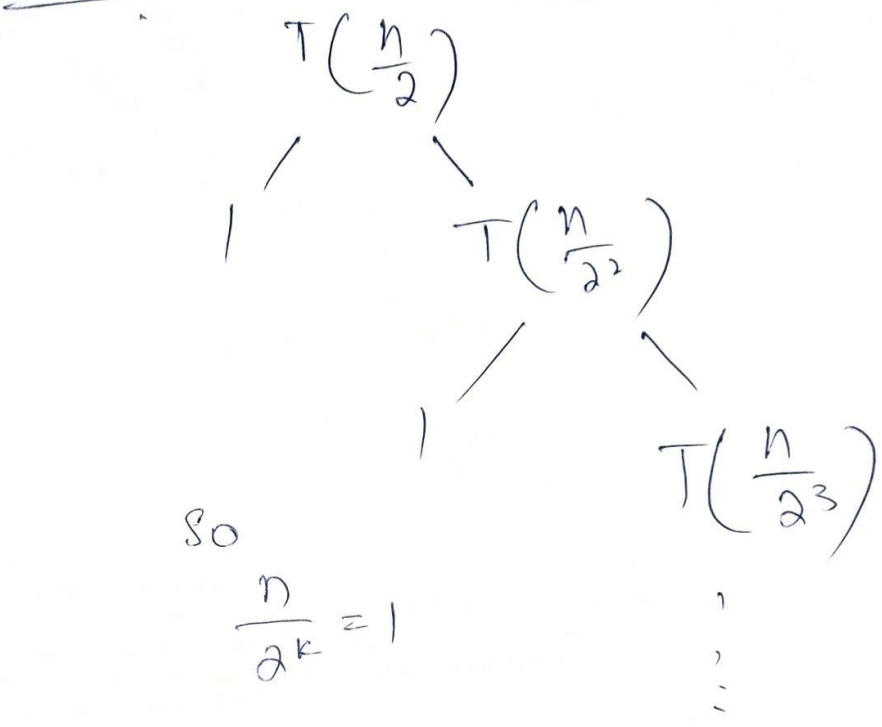
The following code shows how the given code works. For the following code, find the complexity T(n) using recursion tree method.

```
class MidExam {
    // Returns index of x if it is present in arr[l..r],
    // else return -1
    int midExam(int arr[], int l, int r, int x)
    {
        if (r >= l) {
            int mid = l + (r - l) / 2;
             // If the element is present at the
            // middle itself
            if (arr[mid] == x)
                return mid;
             // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
                return midExam(arr, l, mid - 1, x);

            // Else the element can only be present
            // in right subarray
            return midExam(arr, mid + 1, r, x);
        }

        // We reach here when element is not present
        // in array
        return -1;
    }
     // Driver method to test above
    public static void main(String args[])
    {
        MidExam ob = new MidExam();
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
        int result = ob.midExam(arr, 0, n - 1, x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element      found      at      index
"                               + result);
    }}
```

Q4

$$T\left(\frac{n}{2}+1\right)$$

So



So

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

$$\Rightarrow \log(n)$$

- Correct = full marks

else

zero.

Question 5 [10 Marks]

Show the merge sort trace for the following array [12,3,4,10,41,5,15,6]. Show all the steps taken to sort the above array along with the sub list created in each step. One example is as follows

Arr [4,1,3]

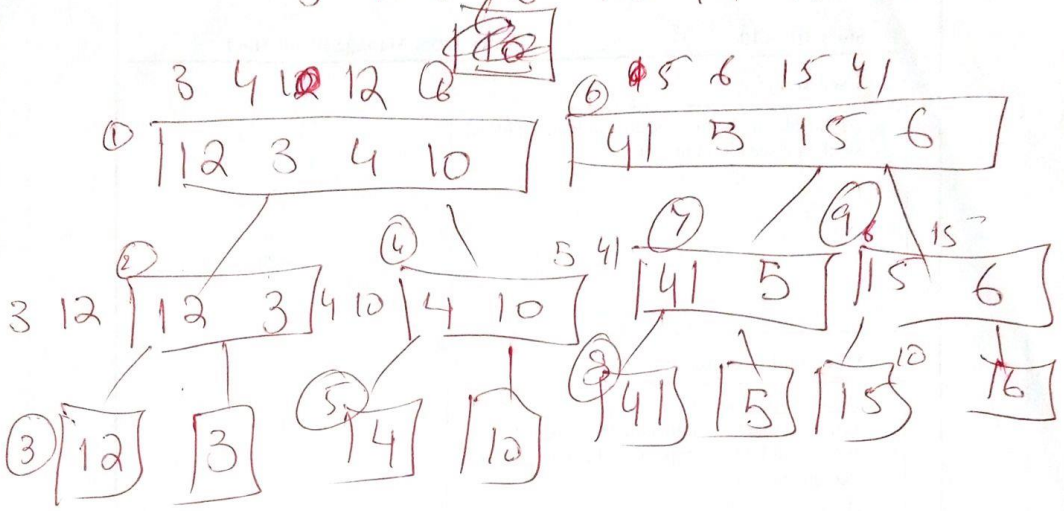
Step 1: Subarray1 {2} Subarray2 {1} array {1,4,3}

Step 2: Subarray1 {1,4} Subarray2 {3} array {1,3,4}

Solution ~~30~~ Q5

12 3 4 10 41 5 15 6

3 4 5 6 10 12 15 41



---

12 3 4 10 41 5 15 6

12 3 4 or 41 5

12 3

4 10 → full ok = 10 marks

→ ~~Part~~ only tree no steps 5 marks

→ zero.

Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1+1+\cdots+1}_{u-l+1 \text{ times}} = u-l+1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$
2. $\sum_{i=1}^{n} i = 1+2+\cdots+n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$
3. $\sum_{i=1}^{n} i^2 = 1^2+2^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$
4. $\sum_{i=1}^{n} i^k = 1^k+2^k+\cdots+n^k \approx \frac{1}{k+1}n^{k+1}$
5. $\sum_{i=0}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1}$ ($a \ne 1$); $\sum_{i=0}^{n} 2^i = 2^{n+1}-1$
6. $\sum_{i=1}^{n} i2^i = 1\cdot 2+2\cdot 2^2+\cdots+n2^n = (n-1)2^{n+1}+2$
7. $\sum_{i=1}^{n} \frac{1}{i} = 1+\frac{1}{2}+\cdots+\frac{1}{n} \approx \ln n+\gamma$, where $\gamma \approx 0.5772\ldots$ (Euler's constant)
8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

Sum Manipulation Rules

1. $\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$
2. $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$
3. $\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i$, where $l \le m < u$
4. $\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$

Properties of Logarithms

1. $\log_a 1 = 0$
2. $\log_a a = 1$
3. $\log_a x^y = y \log_a x$
4. $\log_a xy = \log_a x + \log_a y$
5. $\log_a \frac{x}{y} = \log_a x - \log_a y$
6. $a^{\log_b x} = x^{\log_b a}$
7. $\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$

Approximation of a Sum by a Definite Integral

$$\int_{l-1}^{u} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l}^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_{l}^{u+1} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l-1}^{u} f(x)dx \quad \text{for a nonincreasing } f(x)$$

Floor and Ceiling Formulas

The *floor* of a real number x, denoted $\lfloor x \rfloor$, is defined as the greatest integer not larger than x (e.g., $\lfloor 3.8 \rfloor = 3$, $\lfloor -3.8 \rfloor = -4$, $\lfloor 3 \rfloor = 3$). The *ceiling* of a real number x, denoted $\lceil x \rceil$, is defined as the smallest integer not smaller than x (e.g., $\lceil 3.8 \rceil = 4$, $\lceil -3.8 \rceil = -3$, $\lceil 3 \rceil = 3$).

1. $x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$

2. $\lfloor x + n \rfloor = \lfloor x \rfloor + n$ and $\lceil x + n \rceil = \lceil x \rceil + n$ for real x and integer n

3. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

4. $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$

Miscellaneous

1. $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ as $n \to \infty$ (Stirling's formula)

2. Modular arithmetic (n, m are integers, p is a positive integer)

$$(n+m) \bmod p = (n \bmod p + m \bmod p) \bmod p$$
$$(nm) \bmod p = ((n \bmod p)(m \bmod p)) \bmod p$$

Theorem 4.1 (Master theorem)
Let $a \ge 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Master Method: 2

Consider a recurrence of the form

$T(n) = a\,T(n/b) + f(n)$

with a>=1, b>1, and f(n) eventually positive and $f(n) = O(n^k \log^p n)$

if($\log_b a$) >k

$O(n^{\wedge} \log_b a)$

if($\log_b a$) = k

if p > -1 $O(n^k \log^{p+1} n)$

if p = -1 $O(n^k \log(\log n))$

if p < -1 $O(n^k)$

if($\log_b a$) < k

if p>=0 $O(n^k \log^p n)$

if p<0 $O(n^k)$