Question 1.

$$4 \log \log n < 4 \lg n < n^{1/2} \lg^4(n) < 5^n$$
$$< n^4 < (\lg n)^{.5 \lg n} < n^{n/5} < 5^n < 5^{5n}$$
$$< (n/4)^{n/4} < n^{n/4} < 4^{n^4} < 4^{4^n} < 5^{5^n}$$

Question 2

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

$n^{\log_4 2} = n^{1/2} > f(n)$

Case 1 : $T(n) = \Theta(n^{1/2})$

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

$f(n) = n^{1/2} = n^{\log_b a}$

Case 2 : $T(n) = \Theta(n^{1/2} \log n)$

3. $T(n) = 7T(n/3) + n^2$

$n^{\log_b a} = n^{\log_3 7} < f(n)$

Case 3 : $T(n) = \Theta(n^2)$

4. $T(n) = 7T(n/2) + n^2$

$n^{\log_b a} = n^{\log_2 7} = n^{2.80} > n^2 = f(n)$

Case 1 : $T(n) = \Theta(n^{\log_2 7})$

5. $T(n) = T(n-3) + n^2$

Can't be solved by Master's Theorem

6. $T(n) = 2T(n/2) + n \log n$

$n^{\log_2 2} = n^1 \geq f(n)$

Case 1: $T(n) = \Theta(n)$

7. $T(n) = T(n-2) + n \log n$

Can't be solved by Master's Theorem

Question 3.

$T(n) = 2T(n/2) + O(n)$ = UP: O, LP: Ω

$\Rightarrow T(n) \leq 2T(n/2) + cn$

for some +ve constant c.

Guess: $T(n) \leq dn \log n$ (for some +ve const d)

$$T(n) \leq 2T(n/2) + cn$$
$$= 2\left(\frac{dn}{2}\log\frac{n}{2}\right) + cn$$
$$= dn\log\frac{n}{2} + cn \quad \text{—— 1}$$
$$= \underbrace{dn\log n}_{\text{desired}} - \underbrace{dn + cn}_{\text{residual}}$$

from 1

$T(n) \leq dn \log n$ — if $-dn + cn \leq 0$

$d \geq c$

$\therefore T(n) = O(n \log n)$ UP.

LOWER BOUND

$T(n) \geq 2T(n/2) + cn$ (for some const c)

Guess: $T(n) \geq dn \log n$ (for some const 'd')

Sub: $T(n) \geq 2T(n/2) + cn$

$$= 2\left(\frac{dn}{2}\log\frac{n}{2}\right) + cn$$

$= dn \log \frac{n}{2} + cn$ ——— 11

$= \underbrace{dn \log n}_{\text{desired}} - \underbrace{dn + cn}_{\text{residual}}$

from 11

$T(n) \geq dn \log n$

we proof : $-dn + cn \geq 0$

$\therefore \boxed{d \leq c}$

$\therefore T(n) = \Omega(n \log n)$ CP

$T(n) = \Theta(n \log n)$ $\therefore d = c$

Question 4

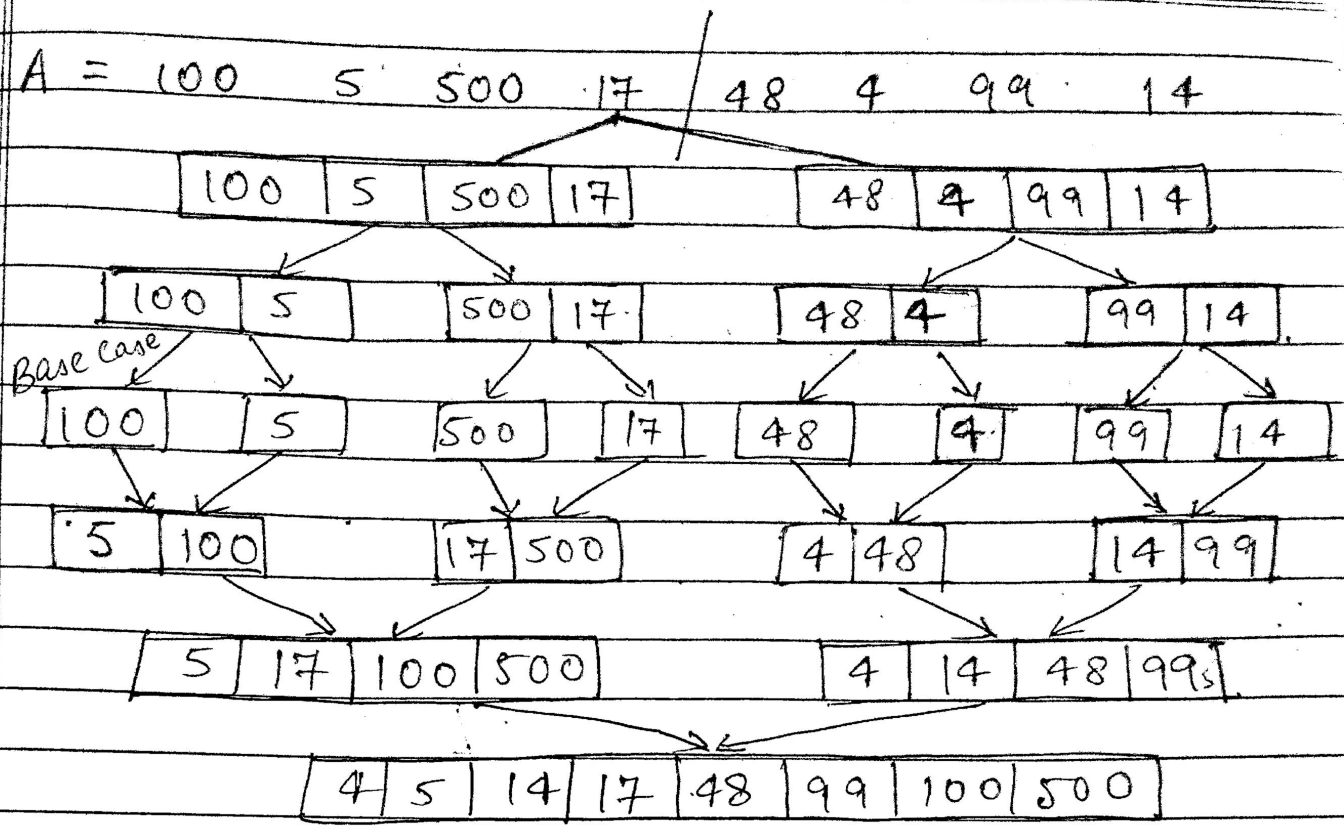
The merge sort algorithm closely follows the divide and conquer paradigm. Intuitively, it operated as follows.

Divide : Divide the n-element sequence to be sorted into two subsequences of n/2 elements each

Conquer : Sort the two subsequences recursively using merge sort

Combine : Merge the two sorted subsequences to produce the sorted answer.

The following example explains the working of merge sort :



Pseudo-code for recursive merge sort

```
Merge_Sort(A, p, r)

If  p < r:
    q = floor((p + r) / 2)
    Merge_Sort(A, p, q)
    Merge_Sort(A, q + 1, r)
    Merge(A, p, q, r)
```

```
Merge(A, p, q, r)

n1 = q - p + 1
n2 = r - q
```

Let $L[1 \ldots n_1+1]$ and $R[1 \ldots n_2+1]$ be new arrays

```
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Date / /
Page

Pseudocode : Iterative Mergesort

```
Item min( Item a, Item b )
{return isLess(a,b)? a:b;}
mergesort( Item a[], int p, int q)
{
    m = 1
    while ( m<q-p):
        i = p
        while( i < q)
            from = i
            mid = i+m-1;
            to = min( i+m+m-1, q)
            merge (a, from, mid, to)
            i = i+m+m
        m = m+m
}
```

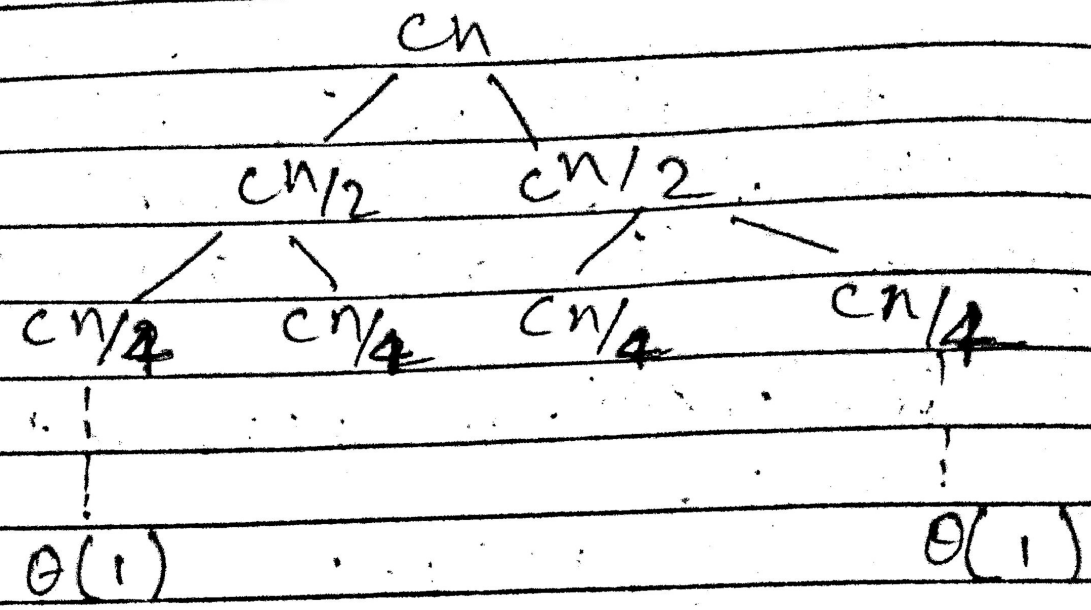
Time complexity of merge sort

$T(n) = 2T(n/2) + \Theta(n)$ if $n > 1$

Recurrence equation

$T(n) = 2T(n/2) + cn$ $c > 0$

Recurrence tree



height of tree = log n
no. of leaves = n
∴ $n \log n + cn$

Ans. Time complexity = $O(n \log n)$

Question 5

Date / /

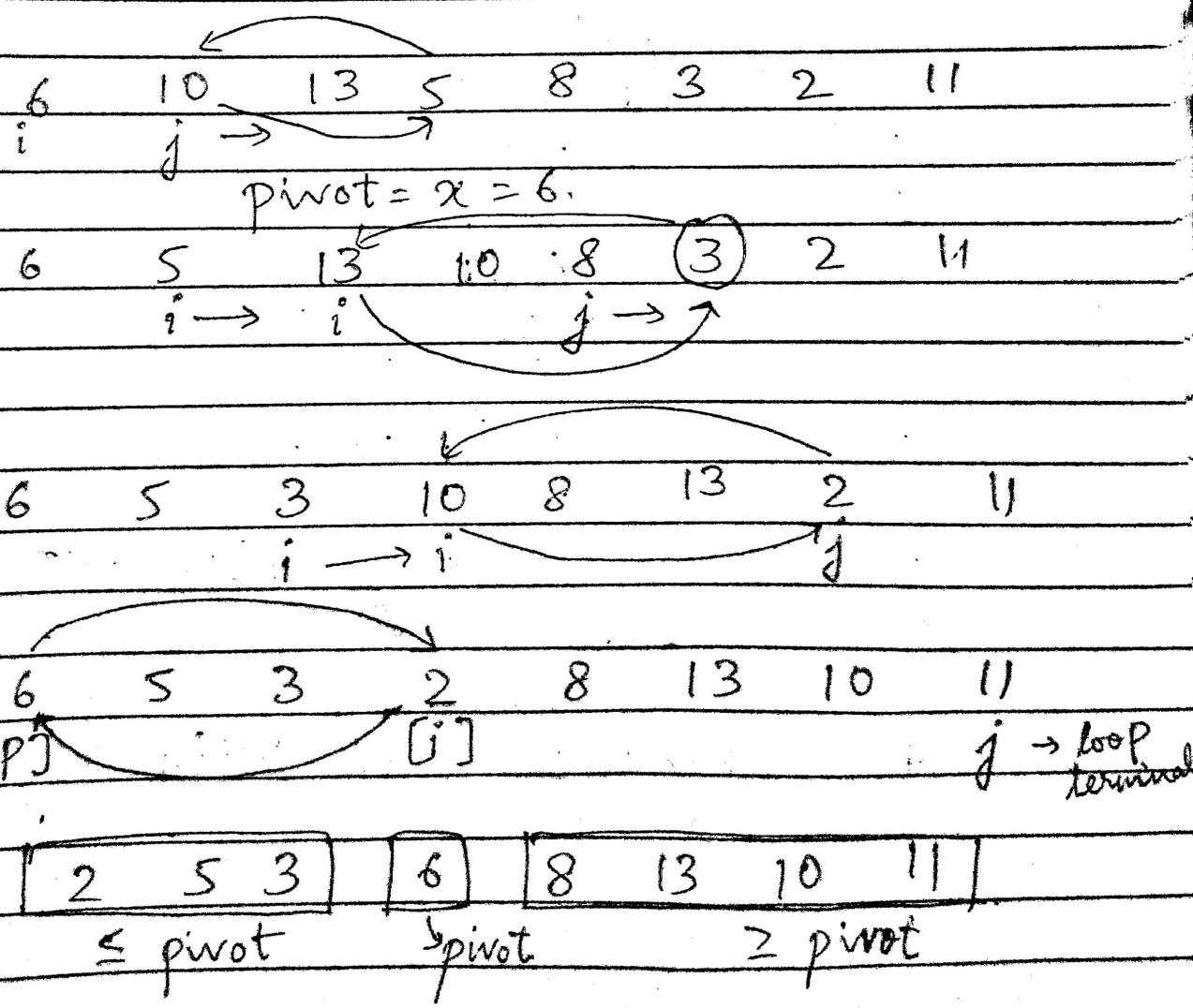
Page

bill 10/10

Pseudocode : Quick Sort

```
Partition (A, p, q)  // A[p ... q]
1.   x ← A[p]        // pivot A[p]
2.   i ← p[index]
3.   for j ← p+1 to q
4.       if A[j] ≤ x
5.           then i = i+1
6.           exchange A[i] ↔ A[j]
7.       exchange A[p] ↔ A[i]
8.   return i  // to know where the pivot element
```

E.g.

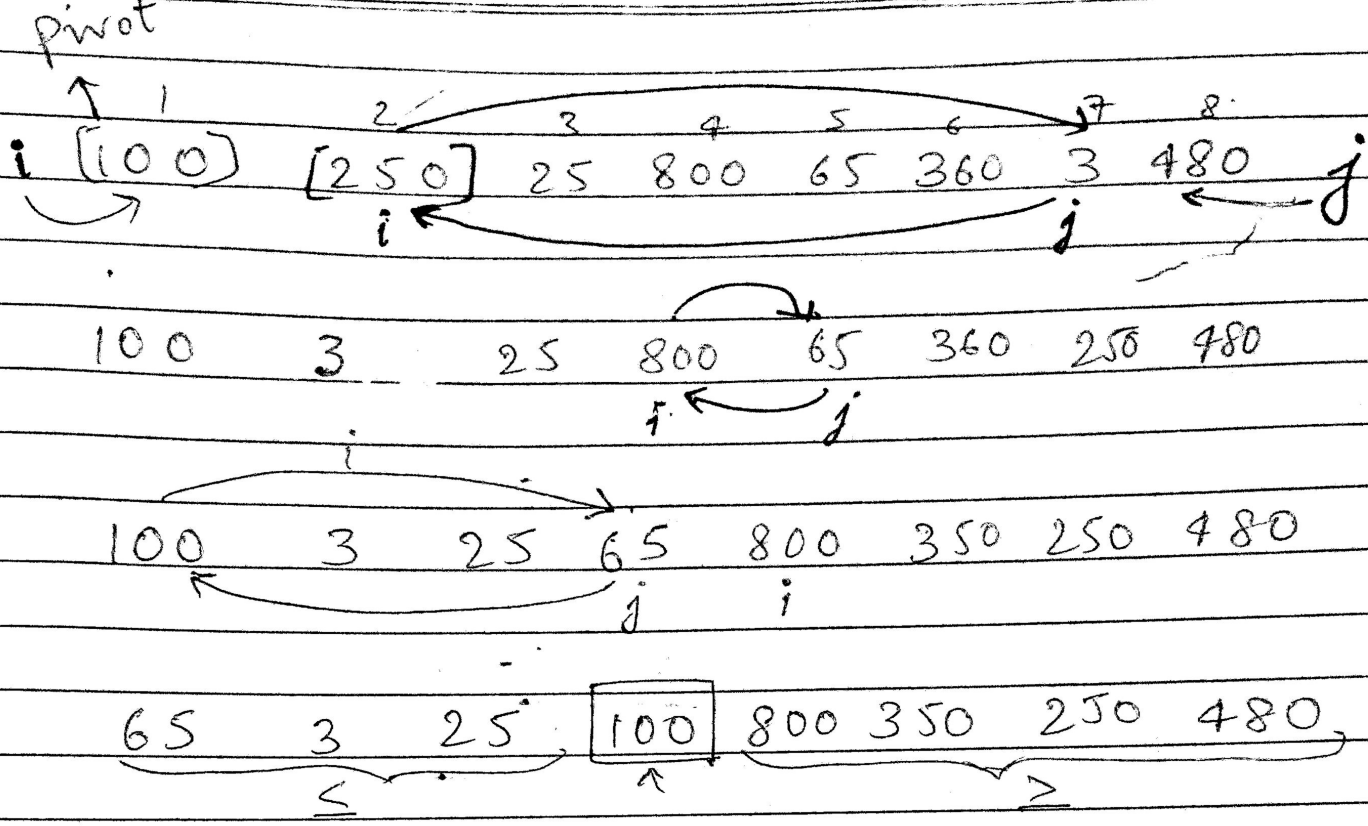


```
QuickSort(A, p, q)
    if p < q
    then q ← Partition(A, p, q)
         QuickSort(A, p, q-1)
         QuickSort(A, q+1, q)
```

Initial call for quicksort(A, 1, n).

Hoare Partition Algorithm

```
1.  x = A[p]
2.  i = p-1
3.  j = q+1
4.  while true
5.      repeat
6.          j = j-1
7.      until A[j] ≤ x
8.      repeat
9.          i = i+1
10.     until A[i] ≥ x
11.     if i < j
12.         exchange A[i] with A[j]
13.     else
            return j
14. exchange a[0] with a[j]
```



Analysis of Quick Sort

Best Case: each partition will be half $(n/2)$

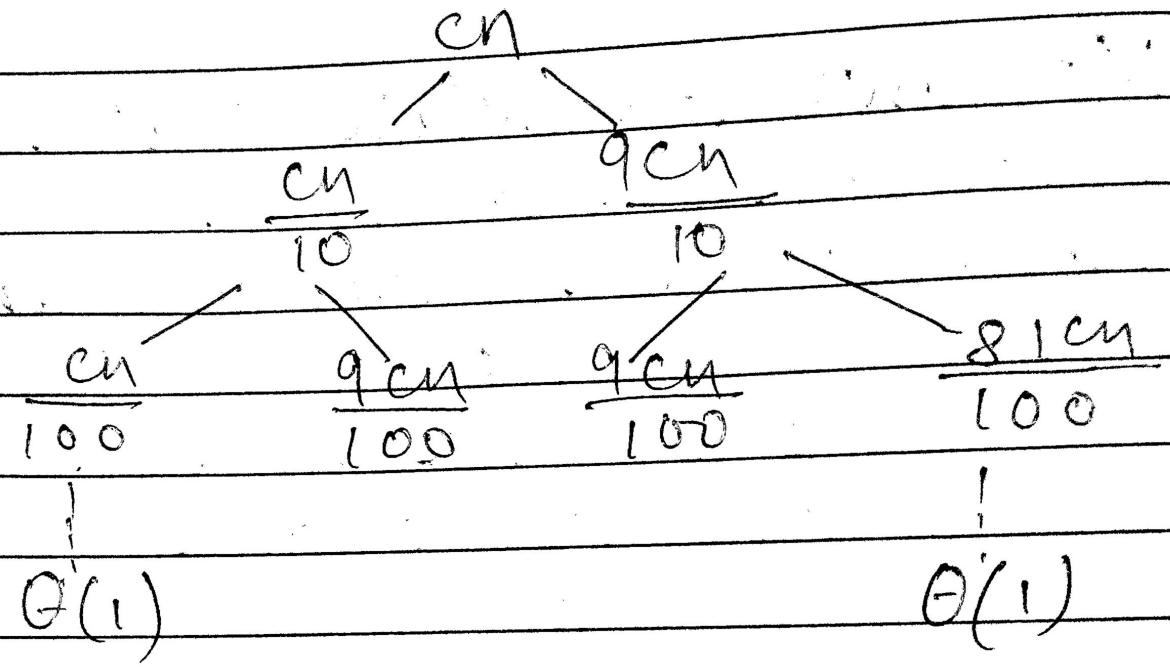
$$T(n) = 2T(n/2) + \Theta(n)$$
$$T(n) = n \log n$$

Average Case:
Partition happen with ratio of $\frac{1}{10}, \frac{9}{10}$

Recurrence

$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9n}{10}\right) + \Theta(n)$$

$T(n) =$



$$T(n) = cn \log_{10/9} n + \theta(n)$$

Time complexity $\Rightarrow$ $\boxed{T(n) = n \log n}$