Jannatul Ferdous Umama

ID : 20-42616-1

Sub : Algorithms

Sec : P

Topic : Mid Assignment.

Jannatul Ferdous Umama
ID : 20-42616-1

Answer to the question no-1.

Counting sort : Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position to each object in the output sequence

Time Complexity : $O(n+k)$ where n is the number of elements in input array and k is the range of input.

Here,

6	3	1	2	1	1	2	3	4	4

Counting Sort [A] [Pseudo code]

```
for i = 0 to n do
c [i] = 0
for j = 0 to n do
c [A [j]] = c [A [j]] + 1
```

Jannatul Ferdous Umama
ID : 20-42616-1

```
for i = 1 to n do
c[i] = c[i] + c[i-1]
for j = m-1 down to '0' do
B[c[A(j)]] - 1] = A[j]
c[A(j)] = c[A[j]] - 1
end function
```

Time complexity

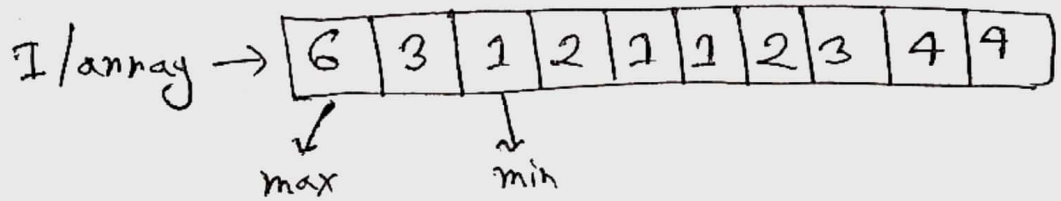
$n + r + n + n$

$= 3n + r$

$= O(n+r)$

Space complexity

$n + r$



Index	:	1	2	3	4	5	6
Count	:	3	2	2	2	0	1
Sum Count	:	3	5	7	9	9	10

position	:	0	1	2	3	4	5	6	7	8	9
S/array	:	1	1	1	2	2	3	3	4	4	6

[Sorted]

Jannatul Ferdous Umama
ID: 20-42626-1

Answer to the question no-2

Greedy Algorithm : A greedy Algorithm is a simple, intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. Greedy algorithms are quite successful in some problems, such as Huffman encoding which is used to compress data, or Dijkstra's algorithm, which is used to find the shortest path through a graph.

Example : Suppose one has to choose a number of coin. He can choose by weight to get mixinum. He can also choose by the value of coin to get maximum rate.

Jannatul Ferdous Umama
ID: 20-42616-1.

Jobs	Time
J1	4
J2	5
J3	8
J4	10
J5	13
J6	14
J7	15
J8	18
J9	20
J10	24
J11	30
J12	32

longest time

$J_{10} + J_{12} + J_{11} = 24 + 30 + 32$
$= 86$

Shortest time.

$J_1 + J_2 + J_3 = 4 + 5 + 8$
$= 17.$

The advantage to using a greedy algorithm is that solutions to smaller instances of the problem can be straightforward and easy to understand. Always taking the best available choice is usually easy.

Jannatul Ferdous Umama
ID : 20-42626-1

Answer to the question no-3.

Activity Selection : The Activity Selection Problem is an optimization problem which deals with the selection of non-conflicting activities that needs to be executed by a single person or machine in a given time frame.

⇒ ⊛ It might not be possible to complete all the activites, since their timings can collapse.

⇒ ⊛ Two activities, say i and j, are said to be non-conflicting if $s_i >= f_j$ or $s_j >= f_i$ where s_i and s_j denote the starting time of activities i and j respectively, and f_i and f_j refer to the finishing time of the activities i and j respectively.

⇒ ⊛ Greedy approach can be used to find the solution since we want to maximize the count of activities that can be executed. This approach will greedily choose an activity with earliest finish time at every step, thus yielding an optimal solution.

Jannatul Ferdaus Umama
ID: 20-42626-1

Activity (a)	1	2	3	4	5	6	7	8
s_i	4	0	1	1	3	2	5	4
f_i	6	2	3	6	4	7	6	8

Activity Selection, [Pseudo code]

$A \rightarrow \{a_1\}$
$n \rightarrow$ length
$i \rightarrow 1$
for $m \rightarrow 2$ to n
do if $S_m \geq f_i$
then $A \rightarrow A \cup \{a_m\}$
$i \rightarrow m$
return A.

Sorted Array according to f_i:

Activity	1	2	3	4	5	6	7	8
S_i	0	1	3	4	1	5	2	4
f_i	2	3	4	6	6	6	7	8

{1, 3, 4} which is a larger at (optimal Sol^n)

Jannatul Ferdous Umama
ID: 20-42616-1

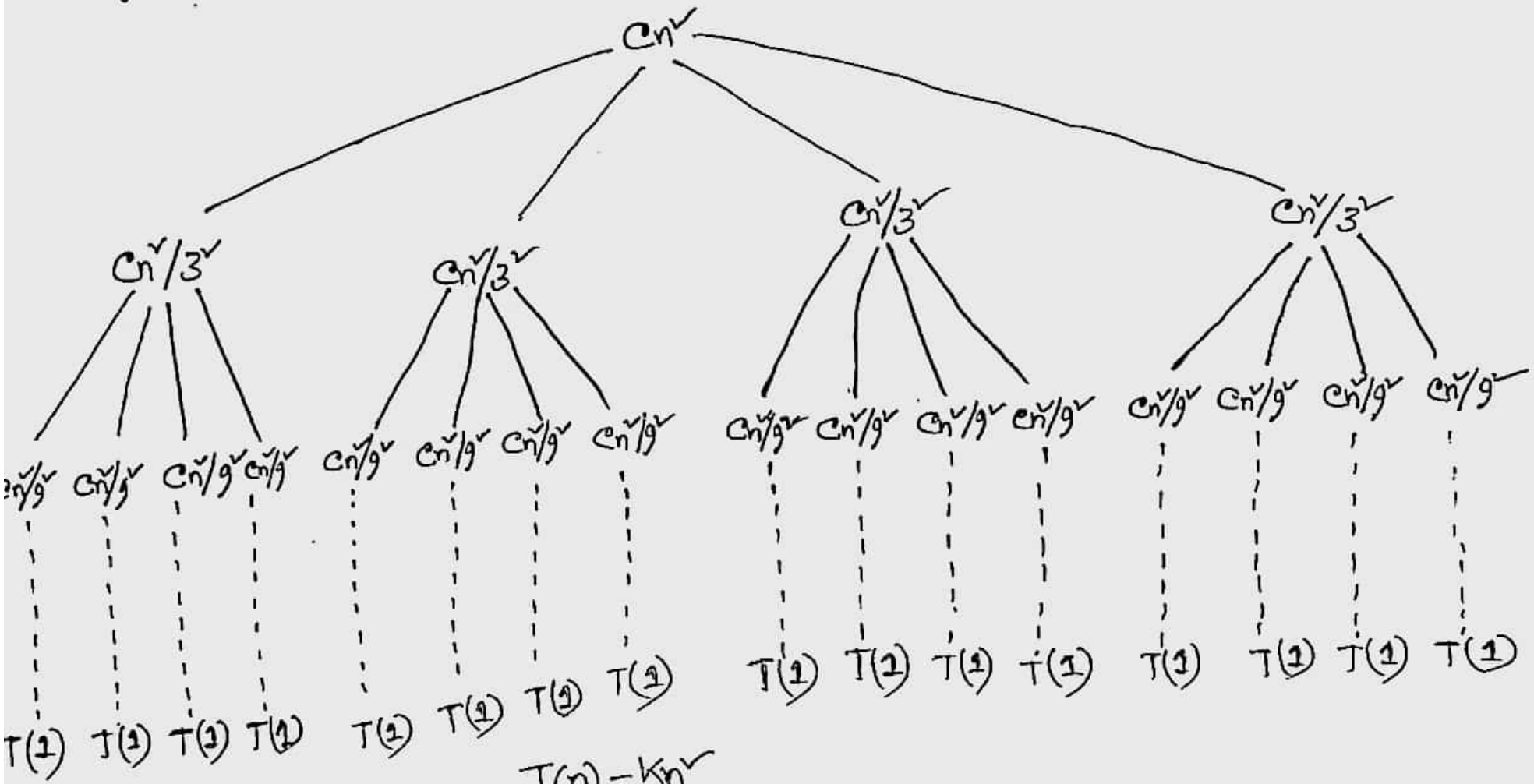
but his is not unique, consider {2, 3, 9} is another optimal solution.

Assume there exist n activities with each of them being represented by a start time S_i and finish time f_i. Two activities i and j are said to be non-conflicting if $S_i \geq f_j$ or $S_j \geq f_i$. The activity selection problem consists in finding the maximal solution set (S) of non-conflicting activities, or more precisely there must exist no solution set S, such that $|S'| > |S|$ in the case that multiple maximal solutions have equal sizes.

Answer to the question no- 4

Here,

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



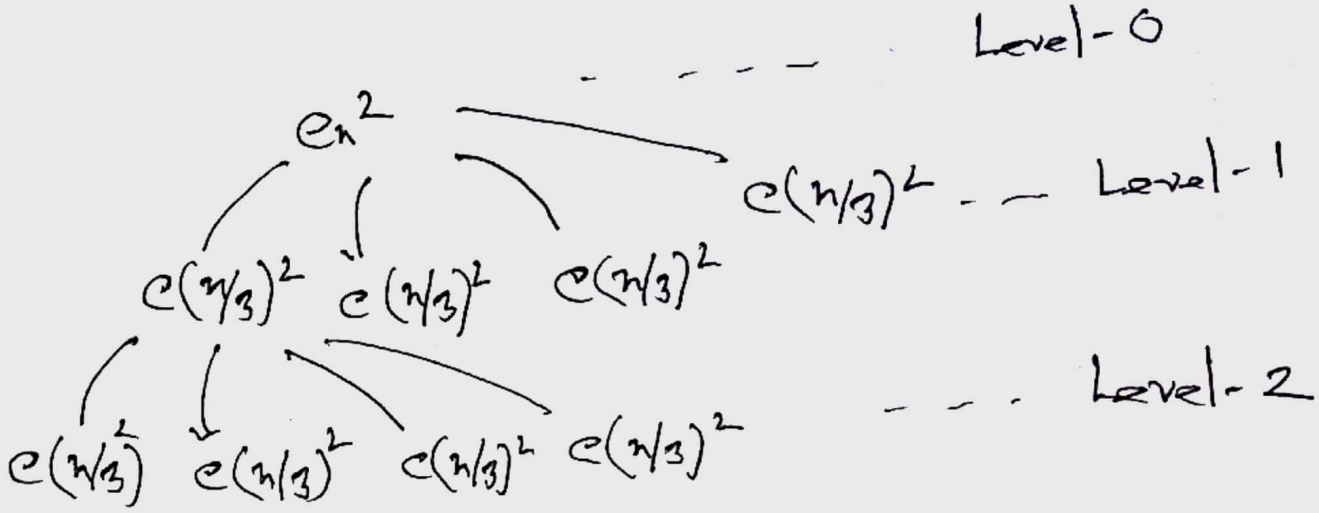
$T(n) = kn^2$

$= Tn = n \log_4 3$

Total $= \Theta(n^2)$

Jannatul Ferdous Unama
ID: 20-42616-1

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



Determine cost of each level:

Cost of level 0: cn^2

Cost of level 1:

$$c(n/3)^2 + c(n/3)^2 + c(n/3)^2 + c(n/3)^2 = (4/9)cn^2$$

Cost of level 2:

$$c(n/9)^2 * 16 = (9^2 \times 16)cn^2$$

Total number of levels in the recursion tree

Size of subproblem at level 0: $n/3^0$

Size of subproblem at level 1: $n/3^1$

Size of subproblem at level 2: $n/3^2$

Jannatul Ferdous Umama
ID: 20-42616-1

Continue in similar manner we have.

Size of Sub problem at level $i = n/3^i$

Suppose at level x (Last level) size of Sub-problem becomes 1. Then

$$n/3^x = 1$$

$$3^x = n$$

Taking log on both sides we get.

$$x \log 3 = \log n$$

$$x \log 3n$$

∴ Total number of levels in the recursion tree. $\log_3 n + 1$

Determine number of nodes in the last level

Level 0 has 4^0 nodes ie. 1 node

Level 1 has 4^1 nodes ie. 4 node

Level 2 has 4^2 nodes ie. 16 nodes.

Jannatul Ferdaus Umama
ID : 20-42616-1

Continuing in similar manner, we have.

Level - $\log_3 n$ has $4^{\log_3 n}$ nodes i.e. $n^{\log_3 4}$ nodes.

Determine cost of last level

Cost of last level :

$$n^{\log_3 4} \times T(1) = \theta(n^{\log_3 4})$$

Add costs of all the levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation -

$$T(n) = \left\{cn^2 + \frac{4}{9}cn^2 + \frac{16}{(9)^2}cn^2 + \dots\right\} + \theta(n^{\log_3 4})$$

$$T(n) = \left\{cn^2 + \frac{4}{9}cn^2 + \left(\frac{4}{9}\right)^2 cn^2 + \dots + \left(\frac{4}{9}\right)^{\log_3 n - 1} cn^2\right\} + \theta(n^{\log_3 4})$$

$$= \sum_{i=0}^{\log_3 n - 1} \left(\frac{4}{9}\right)^i cn^2 + \theta(n^{\log_3 4})$$

The left term is just the sum of a geometric series So, $T(n)$ evaluates to

$$\frac{(4/9)^{\log_3 n} - 1}{(4/9) - 1} cn^2 + \theta(n^{\log_3 4})$$

Jannatul Ferdous Umama
ID: 20-42636-1

This looks complicated but we can bound it (from above) by the sum of infinite series.

$$\sum_{i=0}^{\infty} \left(\frac{4}{9}\right)^i cn^2 + \Theta\left(n^{\log_3 4}\right) = \frac{1}{1-(4/9)} cn^2 + \Theta\left(n^{\log_3 4}\right)$$

Since function in $\Theta(n^{\log_3 4})$ are also in $O(n^2)$, this whole expression is $O(n^2)$

Therefore, we can guess that $T(n) = O(n^2)$.

(Ans)

Jannatul Ferdous Umama
ID: 20-42636-1

Answer to the question no- 5

i) $T(n) = 4T(n/2) + n$

here, $a = 4$, $b = 2$, $k = 1$, $p = 0$

Comparing a with b^k

$a = 4$, $b^k = 2^1 = 2$

So, $a > b^k$

$T(n) = \Theta(n^{\log_b a})$

$= \Theta(n^{\log_2 4})$

$= \Theta(n^2)$

ii) $T(n) = 9T(n/3) + n^2$

Here, $a = 9$, $b = 3$, $k = 2$, $p = 0$

Comparing a with b^k

$a = 9$ and $b^k = 3^2 = 9$

So, $a = b^k$ and $p > -1$

$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

$= \Theta(n^{\log_3 9} \log^{0+1} n)$

$= \Theta(n^2 \log n)$

Jannatul Ferdous Umama
ID: 20-42616-1

iii) $T(n) = 2T(n/2) + n$

here, $a=2, b=2, k=1, p=0$

Comparing a with b^k

$\therefore a=2 \quad b^k = 2^1 = 2$

$\therefore a = b^k$ and $p > -1$

$T(n) = \theta(n^{\log_b^a} \log^{p+1} n)$

$= \theta(n^{\log_2^2} \log^{0+1} n)$

$= \theta(n \log n)$

iv) $T(n) = 4T(n/2) + n^3$

here, $a=4, b=2, k=3, p=0$

Comparing a with b^k

$a = 4 \quad b^k = 2^3 = 8$

$a < b^k$ and $p=0$

$T(n) = \theta(n^k \log^p n)$

$= \theta(n^3 \log^0 n)$

$= \theta(n^3)$

Jannatul Ferdaus Umama
ID: 20-92616-1

Answer to the question no-6

Using Repeated Substitution method : The substitution method is a condensed way of proving an asymptotic bound on a recurrence by induction. In the substitution method, instead of trying to find an exact closed-form solution, we only try to find a closed-form bound on the recurrence. This is often much easier than finding a full closed-form solution, as there is much greater leeway in dealing with constants. The substitution method is a powerful approach that is able to prove upper bounds for almost all recurrences. However, its power is not always needed; for certain types of recurrences, the master method can be used to derive a tight bound with less work.

Jannatul Ferdaus Umama
ID : 20-42616-1

In those cases, it is better to simply use the master method, and to save the substitution method for recurrences that actually need its full power. Note that the substitution method still requires the use of induction. The induction will always be of the same basic form, but it is still important to state the property you are trying to prove, split into one or more base cases and the inductive case, and note when the inductive hypothesis is being used.

$$T(n) = \begin{cases} 1 & n=1 \\ 3T(n/3)+n & n>1 \end{cases}$$

[Guess $T(n) \leq cn \log_3 n$ for some c, for any $n \geq n_0$]

$n=1$, $T(1) = 1 \not< c \cdot \log_3 1$ not true

So, $n=3$, $T(3) = 3T(1) + 3$

$= 6 \not< c\, 3 \log_3 3$

$= 3c$

true for $c \geq 2$

Jannatul Ferdous Umama
ID: 20-42616-1

$$T(n) = 3T(n/3) + n$$
$$\leq 3(c\, n/3 \log_3 n/3) + n \quad \text{(by inductive hyp)}$$
$$= cn \log_3 n/3 + n$$
$$= cn \log_3 n - cn \log_3 3 + n$$
$$= cn \log_3 n - cn + n$$

We need to prove $T(n) \leq cn \log_3 n$ true when $c \geq 1$

Putting 2 conditions on c, we get $c \geq 2$

So, $T(n) \leq 2n \log_3 n$ true.

Jannatul Ferdous Umama
ID: 20-42616-1

Answer to the question no- 7

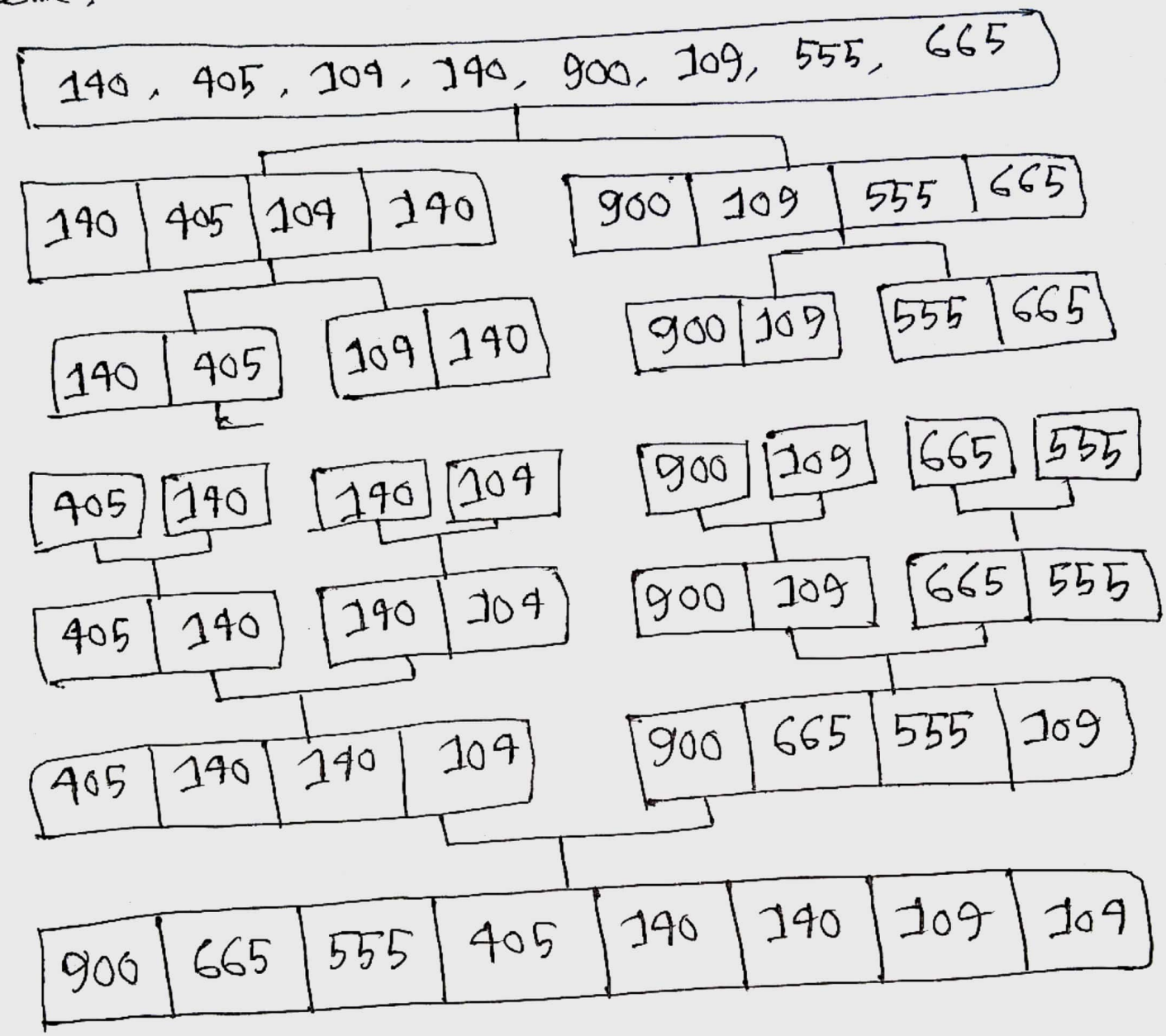
Divide and conquer algorithms :

The two sorting algorithms we've seen so far, selection sort and insertion sort, have worst-case running times of $\theta(n^2)$. When the size of the input array is large, these algorithms can take a long time to run. In algorithms method, the design is to take an input, break the input into small pieces, decide the problem on each small pieces and merge the solution is called devide and conquer.

Merge Sort : In particular, merge sort runs in $\theta(n \lg n)$ time in all cases, and quick sort runs in $\theta(n \lg n)$ time in the best case and on average, though its worst-case running time is $\theta(n^2)$. Three steps →
(i) Devide (ii) Sort. (iii) Merge.

Jannatul Ferdous Umama
ID : 20-42616-1

Here,



⊛ Worst case time Complexcity $O(n \log(n))$

Jannatul Ferdous Umama
ID: 20-42616-1

Answer to the question no - 8

Knapsack problem states that : Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Knapsack problem is real-world decision making processes in a wide veriety of fields such as finding the least wasteful way to cut raw materials, selection of investments and portfolio etc.

Given the knapsack $W = 60$

Item	A	B	C	D
Profit	280	100	120	220
Weight	30	20	20	24
Ratio P/W	9.33	5	6	9.166

Jannatul Ferdous Umama
ID: 20-42626-1

Item	A	B	C	
Profit	280	220	36	Total Profit 536
Weight	30	24	6	
Remaining weight	30	6	0	

Here, total profit is 536, using fractional knapsack.

Knapsack problem can be found real-world scenarios like resource allocation in financial constraints or even in selecting investments and portfolios. It also can be found in fields such as applied mathematics, complexity theory, cryptography, combinatorics and computer science.

Jannatul Ferdous Umama
ID : 20-42616-1

Answer to the question no-9

Asymptotic Notations : Asymptic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value. For example : In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear the best case. But, when the input array is in reverse condition, the algorithm takes the maximum time to sort the elements the worst case. When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

Jannatul Ferdous Umama
ID: 20-42616-1

There are mainly three asymptotic notations:

(i) Big-O notation
(ii) Omega notation
(iii) Theta notation.

(i) Big-O Notation: Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

$$f(n) = 3n^3 + 20n^2 + 5$$

$$3n^3 + 20n^2 + 5n \leq 20n^3$$

$f(n)$ $\quad$ c $\quad$ $g(n)$

$$f(n) \leq cg(n)$$

So, we can say $f(n) = O(g(n))$.

Jannatul Ferdous Umama
ID: 20-42636-1

(II) Big-Omega Notation : Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

$$f(n) = 3n^3 + 20n^2 + 5$$

$$3n^3 + 20n^2 + 5 \geq n^3 \times 1$$

$3n^3 + 20n^2 + 5$ → $f(n)$

n^3 → $g(n)$

1 → c

$$f(n) > cg(n)$$

So, we can say $f(n) = \Omega(g(n))$

(III) Big-Theta-Notation : Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm. it is used for analyzing the average-case complexity of an algorithm.

Jannatul Ferdous Umama
ID: 20-42676-1

$$f(n) = 3n^3 + 20n^2 + 5$$

$$1 \times n^3 \leq 3n^3 + 20n^2 + 5 \leq 20n^3$$

c_1 $g(n)$ $f(n)$ c_2 $g(n)$

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

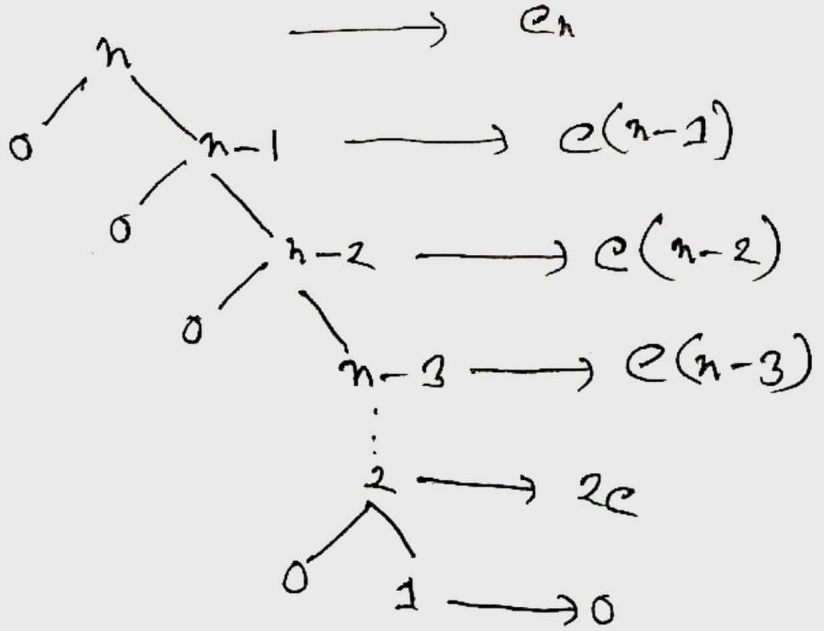
So, we can say. $f(n) = \Theta(g(n))$

Jannatul Ferdous Unona
ID: 20-42616-1

Answer to the question no- 10

Algorithm	Time complexity			Space complexity
	Best	average	Worst	Worst
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Marge Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n\log(n))$	$O(n)$
Quick Sort	$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n^2)$	$O(\log(n))$
Counting Sort	$\Omega(n+k)$	$\Theta(n+k)$	$\Theta(n+k)$	$O(k)$

Worst case of Quick Sort :



Jannatul Ferdous Umama
ID : 20-42516-1

Total up the partitioning time of each level.

$$c_n + c(n-1) + c(n-2) + \ldots + 2(c)$$

$$\Rightarrow c(n + (n-1) + (n-2) + \ldots$$

$$\Rightarrow c((n+1)(n/2) - 1)$$

The last line is because $1+2+3+\ldots+n$ is the arithmetic series. Some low order terms and constant co-efficients.

So, quick sorts worst case runtime is

$$O(n^2).$$