Semester: V (Regular)
Subject Name:- DAA & Code:- CS2012
Branch (s): - CSE, IT, CSCE

AUTUMN MID SEMESTER EXAMINATION-2023

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Design and Analysis of Algorithms
[CS2012]

Time: 1 1/2 Hours **Full Mark: 40**

Answer any four Questions including Question No. 1 which is compulsory.
The figures in the margin indicate full marks. Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

1. Answer all the questions. [2 x 5 = 10]

a)

I. What is the time complexity of the following function, fun( )?

```
int fun(int n){
    int i, j, s=0;
    for(i= 2; i<n; i=i+1)
       for(j = 1; j < n; j= j * i)
               s=s+j;
    return s;
}
```

A. $\Theta(n)$ B. $\Theta(n*\log(n))$ C. $\Theta(n^2)$ D. $\Theta(n\sqrt{n})$ E. NONE

II. What exact value will the function fun() return if the value of ‘n’ is 5?

b) You are executing an algorithm with worst-case time complexity $O(n^2)$ on a CPU that can perform 10^6 operations per second. Calculate the time required to solve a worst-case input of size 100.

c)

I. Solve the recurrence using the master theorem.
$T(n) = 16T(n/4) + n$

II. In a binary max-heap containing ‘n’ number, the smallest element can be found in time

A. O(n) B. O(log n) C. O(log log n) D. O(1)

d) Match the following algorithms with their recurrences:

Bubble-sort	$T(n) = T(n/2)+ \Theta(1)$
Quick-sort	$T(n) = T(n-1)+ \Theta(n)$
Merge-sort	$T(n) = T(k) + T(n-k)+\Theta(n)$
Binary-search	$T(n) = 2T(n/2)+ \Theta(n)$

e) Find the time complexity of the following C function, assume n>0.

```
int recursive(int n){
     if(n==1)
          return 1;
     else
          return (recursive(n-1)+recursive(n-1));
}
```

2. Solve the following recurrences: [5 x 2 = 10]

a) $T(n) = 2\,T(\sqrt{n}) + \log n$ and $T(1) = 1$
b) $T(n) = T(n/10) + T(9n/10) + n$

3. [5 x 2 = 10]

a) Write a recursive algorithm to perform a linear search that searches for the given element in the first location and, if not found, it recursively calls the linear search. Derive the recurrence relation for recursive linear search and solve the recurrence using the iterative method.

b) Consider an alternative to binary search called ternary search, which divides the input array into three equal parts and recursively searches one of these three segments. Write the algorithm and find the key element 13 from the array A = {2, 3, 5, 6, 8, 9, 12, 13, 14} with indices from 1 to 9 using the above procedure.

4. [5 x 2 = 10]

a) Write down the PARTITION(A, p, r) procedure with the last element as the pivot, where p and r are lower & upper bounds of array A. Describe in a step-by-step process how to get the pass1 result of PARTITION() by taking the last element as the pivot on the following array elements. Derive the time complexity of the PARTITION() procedure.
A = {2, 5, 7, 5, 9, 3, 8, 6}

b) Discuss the trade-offs between the Quick-Sort and Merge-Sort algorithms. Compare and contrast their time complexity, space complexity, and stability.

5. [5 x 2 = 10]

a) Write the algorithm to build a MIN-HEAP. Describe your Algorithm to build a MIN-HEAP from the array A = {5, 7, 8, 2, 1, 0, 3, 9, 4, 5, 6} in a step-by-step process. Derive the time complexity of building a MIN-HEAP. Assume the root is at index 1.

b) Given a knapsack with a capacity of 15 kg and a set of items with their weights in kg and profits in rupees:
Items: A B C
Weights: 8 4 10
Profits: 30 15 45
Solve the fractional knapsack problem to maximize the total profits obtained in each of the following cases and find the case(s) with highest profit.

I. Greedy w.r.t. profits only
II. Greedy w.r.t. weights only
III. Greedy w.r.t. profit per unit weight

*** Best of Luck ***

SOLUTION & EVALUATION SCHEME

1. Answer all the questions. [2 x 5 = 10]

a)

I. What is the time complexity of the following function, fun()?

```
int fun(int n){
    int i, j, s=0;
    for(i= 2; i<n; i=i+1)
       for(j = 1; j < n; j= j * i)
              s=s+j;
    return s;
}
```

II. What exact value will the function fun() return if the value of ‘n’ is 5?

Scheme:

- I. Correct answer: 1 Mark, Wrong answer: 0 mark
- II. Correct answer: 1 Mark, Wrong answer but explanation approaches to answer: 0.5 mark

Answer:

- I. B - Θ (n log n) or E - None

Note:: The correct answer is O(n log n), which does not match any of the provided options. Nonetheless, option B bears a resemblance to the correct answer. Therefore, if students have selected option B, they will be awarded full marks.

- II. 16

b) You are executing an algorithm with worst-case time complexity $O(n^2)$ on a CPU that can perform 10^6 operations per second. Calculate the time required to solve a worst-case input of size 100.

Scheme:

- Correct answer: 2 Marks
- Wrong answer, but explanation approaches to answer: Step Marking

Answer: Time required <= C * 0.01 sec (<= C. 10 ms)

Note# if students have written time required = 0.01 sec or 10 ms by assuming C = 1, full marks could be given.

Explanation:

$T(n) <= C. n^2$

10^6 operations completed in 1 sec ⇒ 100^2 operations can be completed in <= c * 0.01 Sec.

c)

I. Solve the recurrence using the master theorem.
   $T(n) = 16T(n/4) + n$

II. In a binary max-heap containing 'n' number, the smallest element can be found in time
   A. O(n) B. O(log n) C. O(log log n) D. O(1)

Scheme:

- I. Correct answer: 1 Mark, Wrong answer, but explanation approaches to answer: 0.5 mark
- II. Correct answer: 1 Mark, Wrong answer 0 mark

Answer:

- I. $\Theta(n^2)$
- II. O(n)

Explanation:

a = 16, b=4, f(n) = n

$n^{\log_b a} = n^{\log_4 16} = n^2$
f(n) = n

Comparing $n^{\log_b a}$ and f(n), we get $n = O(n^2)$

Satisfies Case 1 of Master's Theorem, $T(n) = \Theta(n^2)$

d) Match the following algorithms with their recurrences:

Bubble-sort	$T(n) = T(n/2)+ \Theta(1)$
Quick-sort	$T(n) = T(n-1)+ \Theta(n)$
Merge-sort	$T(n) = T(k) + T(n-k)+\Theta(n)$
Binary-search	$T(n) = 2T(n/2)+ \Theta(n)$

Scheme:

- Correct answer: 2 Marks
- Wrong answer, but some options correct: Step Marking (0.5-1.5 Marks)

Answer:

Bubble-sort ------ $T(n) = T(n-1)+ \Theta(n)$
Quick-sort ------ $T(n) = T(k) + T(n-k)+\Theta(n)$
Merge-sort ------ $T(n) = 2T(n/2)+ \Theta(n)$
Binary-search -- $T(n) = T(n/2)+ \Theta(1)$

e) Find the time complexity of the following C function, assume n>0.

```
int recursive(int n){
        if(n==1)
                return 1;
        else
                return (recursive(n-1)+recursive(n-1));
}
```

Scheme:

- Correct answer: 2 Marks
- Wrong answer, but explanation approaches to answer: Step Marking (0.5 - 1.5 Marks)

Answer: $O(2^n)$

Here the recurrence relation is $T(n) = T(n-1) + T(n-1) + O(1) = 2T(n-1) + 1$
Solving the recurrence $T(n) = O(2^n)$

2. Solve the following recurrences: [5 x 2 = 10]

a) $T(n) = 2\,T(\sqrt{n}) + log\, n$ and $T(1) = 1$

Scheme:

Correct answer: 5 Marks

Wrong answer, but explanation approaches to answer: Step Marking

Answer

$$\begin{aligned}
&\text{let } \; n = 2^m \\
&\Rightarrow T(2^m) = 2T(\sqrt{2^m}) + \log(2^m) \\
&\Rightarrow T(2^m) = 2T(2^{m/2}) + m \\
&\text{let } \; S(m) = T(2^m) \\
&\Rightarrow S(m) = 2S(m/2) + m \\
&\Rightarrow S(m) = 2(2S(m/4) + m/2) + m \\
&\Rightarrow S(m) = 2^2 S(m/2^2) + m + m \\
&\text{By substituting further,} \\
&\Rightarrow S(m) = 2^k S(m/2^k) + m + m + \ldots + m + m \\
&\text{let } \; m = 2^k \Rightarrow S(m/2^k) = S(1) = T(2) = 2 \\
&\Rightarrow S(m) = 2 + m(k-1) + m
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow S(m) = 2 + mk \\
&\Rightarrow S(m) = 2 + m\log m \\
&\Rightarrow S(m) = O(m\log m) \\
&\Rightarrow S(m) = T(2^m) = T(n) = O(\log n \log\log n)
\end{aligned}$$

S(m) = 2 S(m/2) +m

Solving the above recurrence by master method, S(m) = Θ (m lg m) = Θ (lg n lg lg n)

Note# If any alternative method used to solved the recurrence, full marks could be given for correct answer.

b) $T(n) = T(n/10) + T(9n/10) + n$

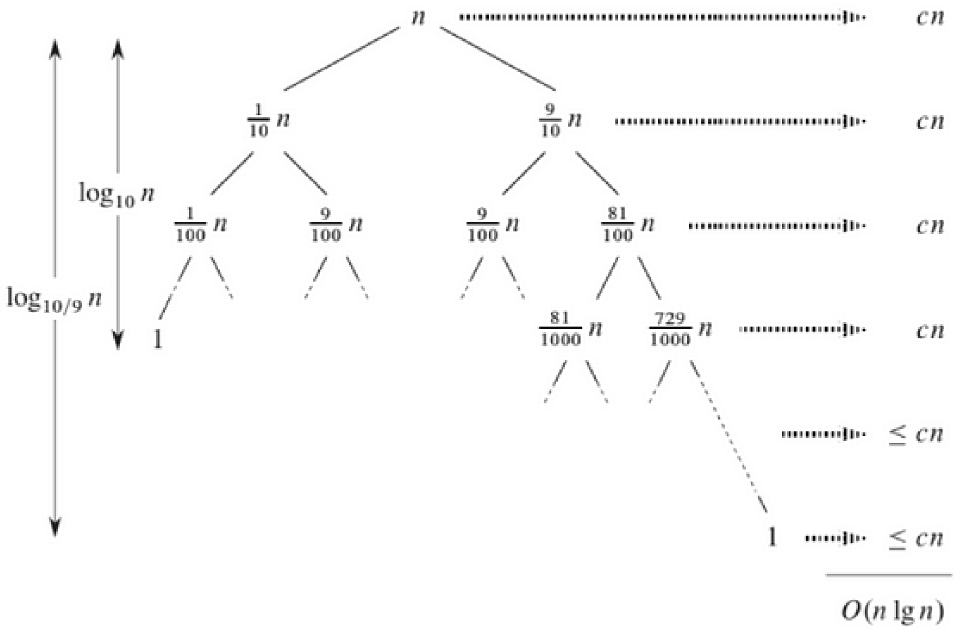
Scheme:

Correct answer: 5 Marks

Wrong answer, but explanation approaches to answer: Step Marking

Answer: T(n) = O (n $\log_{10/9}$n) or O (nlgn)

Explanation:



Choose the longest path from root node to leaf node

$(9/10)^0n \rightarrow (9/10)^1n \rightarrow (9/10)^2n \rightarrow \ldots\ldots\ldots \rightarrow (9/10)^kn$

Size of problem at last level = $(9/10)^kn$

At last level size of problem becomes 1

$(9/10)^kn = 1$

$(9/10)^k = 1/n$

k = $\log_{10/9}$(n)

Total no of levels in recursion tree = k +1 = $\log_{10/9}$(n) + 1

Then $T(n) \leq n \log_{10/9}(n)$

⇨ **T(n) = O (n $\log_{10/9}$n) which can also be expressed as O (nlgn)**

3. [5 x 2 = 10]

a) Write a recursive algorithm to perform a linear search that searches for the given element in the first location and, if not found, it recursively calls the linear search. Derive the recurrence relation for recursive linear search and solve the recurrence using the iterative method.

Scheme:

- Algorithm or program or function : 2 Marks
- Recurrence relation: 1 Mark
- Solving the recurrence: 2 Marks
- Wrong answer, but explanation approaches to answer : Step Marking

Answer

Algorithm LinearSearch(A, p, r, key)

```
if(p<=r)
{
        if(key == A[r])
                return r;
        else
                LinearSearch(A, p, r-1, key);
}
else
        return -1;
```

Let T(n) be the number of comparisons (time) required for linear search on an array of size n.

Note, when n = 1, T(1) = 1.

Then, T(n) = T(n - 1) +1 = and T(1) = 1

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

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

$= [[T(n-3)+1]+1]+1$

…

At k^{th} step,

$T(n) = T(n-k) + k$

Let $n-k = 1$,

⇨ $T(n) = T(1) + n$

⇨ $T(n) = 1 + n$

⇨ $T(n) = \Theta(n)$

b) Consider an alternative to binary search called ternary search, which divides the input array into three equal parts and recursively searches one of these three segments. Write the algorithm and find the key element 13 from the array A = {2, 3, 5, 6, 8, 9, 12, 13, 14} with indices from 1 to 9 using the above procedure.

Scheme:

- Algorithm or Program or function: 3 Marks
- Illustration: 2 Marks
- Wrong answer, but explanation approaches to answer: Step Marking

Answer:

Algorithm TernarySearch(A, p, r, key)

```
if(p<=r)
{
        onethird = p+(r-p)/3.0;
        twothird = r-(r-p)/3.0;
         if(key == A[onethird])
                return onethird;
         else if(key == A[twothird])
                return twothird;
        else if(key < A[onethird])
                TernarySearch(A, p, onethird-1, key);
        else if(key > A[onethird] && key < A[twothird])
                TernarySearch(A, onethird+1, twothird-1, key);
        else
                TernarySearch(A, twothird+1, r, key);
}
else
        return -1;
```

Illustration:

TernarySearch(A, p, r, key) = TernarySearch(A, 1, 9, 13)

p								r
1	2	3	4	5	6	7	8	9
2	3	5	6	8	9	12	13	14

onethird = p+(r-p)/3 = 1+(9-1) /3 = 1+ 2 = 3

twothird = r - (r-p)/3 = 9 – 2= 6

key =13

p		onethird			twothird			r
1	2	3	4	5	6	7	8	9
2	3	5	6	8	9	12	13	14

Key = 13 > a[twothird]

TernarySearch(a, twothird+1, r, key);= TernarySearch(a, 7, 9, 13)

onethird = p+(r-p)/3 = 7+(9-7) /3 = 7

twothird = r-(r-p)/3 = 9 – (9-7)/3 = 8

key =13

p						onethird	twothird	r
1	2	3	4	5	6	7	8	9
2	3	5	6	8	9	12	13	14

key == a[twothird], return 8

The key 13 is found at index position 8.

4. [5 x 2 = 10]

a) Write down the PARTITION(A, p, r) procedure with the last element as the pivot, where p and r are lower & upper bounds of array A. Describe in a step-by-step process how to get the pass1 result of PARTITION() by taking the last element as the pivot on the following array elements. Derive the time complexity of the PARTITION() procedure.

A = {2, 5, 7, 5, 9, 3, 8, 6}

Scheme:
PARTITION algorithm or program: 2 Marks
Illustration: 2 Marks
Time complexity of PARTITION: 1 Mark
Wrong answer, but explanation approaches to answer: Step Marking

<u>Algorithm PARTITION (A, p, r)</u>

1. x ← A[r] //last element as pivot
2. i← p – 1
3. **for** j ← p **to** r – 1 **do**
4. **if** A[j] ≤ x **then**
5. i ← i + 1
6. swap A[i] ↔ A[j]
7. swap A[i + 1] ↔ A[r]
8. **return** i + 1

p							r
1	2	3	4	5	6	7	8
2	5	7	5	9	3	8	**6**

x = A[8] = 6

i = 0, j = 1 2 5 7 5 9 3 8 **6**

i = 1, j = 2 2 5 7 5 9 3 8 **6**

i = 2, j = 3 2 5 7 5 9 3 8 **6**

i = 2, j = 4 2 5 5 7 9 3 8 **6**

i = 3, j = 5 2 5 5 7 9 3 8 **6**

i = 3, j = 6 2 5 5 3 9 7 8 **6**

i = 4, j = 7 2 5 5 3 9 7 8 **6**

2 5 5 3 **6** 7 8 9

Time Complexity of Partition: we are running a single loop and doing constant operation in each iteration. So, the time complexity is O (n).

b) Discuss the trade-offs between the Quick-Sort and Merge-Sort algorithms. Compare and contrast their time complexity, space complexity, and stability.

Scheme:

- Time complexity comparison: 1.5 Marks
- Space complexity comparison: 1.5 Marks
- Stability comparison: 1 Mark
- Overall trade-off: 1 mark
- Wrong answer, but explanation approaches to answer: Step Marking

Answer:

Sl. No.	Parameter	Merge Sort	Quick Sort
1	Time Complexity	Average Case: O(n log n) Worst Case: O(n log n) Best Case: O(n log n) Merge Sort's worst case is more predictable and consistent than Quick Sort's worst case.	Average Case: O(n log n) Worst Case: O(n^2) (un-optimized pivot selection) Best Case: O(n log n) Quick Sort's average case is typically faster than other O(n log n) algorithms due to smaller constant factors.
2	Space Complexity	Requires additional space for temporary storage of sub-arrays (usually an additional array or memory allocation). Space complexity is higher, O(n), due to the need for temporary storage.	In-place partitioning: O(log n) due to the recursion stack Additional space is needed only for recursion stack, making Quick Sort memory-efficient.
3	Stability	Stable: Equal elements maintain their original relative order after sorting. The merging step preserves stability.	Not stable: Equal elements may change their relative order after sorting. Swapping elements during partitioning can lead to instability.
4	Suitability for Different Inputs	More suited for larger datasets and external sorting (where data doesn't fit entirely in memory). Performs consistently well on all input scenarios, making it a reliable choice. Well-suited for parallelization due to its divide-and-conquer nature. More stable performance on already sorted or reverse-sorted arrays.	Generally faster than Merge Sort for small arrays due to reduced constant factors. Efficient on average when the pivot selection is balanced, and the data is uniformly distributed. Poor performance on nearly sorted or reverse-sorted arrays (unless pivot selection is modified). Less suited for large datasets with limited memory (high recursion depth).

5	Practical Considerations	Slower in practice due to higher memory usage and slower memory access patterns. Stable performance across different cases makes it preferable when predictability is crucial.	Generally has better cache performance due to sequential memory access. Faster in practice for many scenarios due to smaller constant factors. Performance can degrade with poor pivot selection or highly repetitive data.

In summary, Quick Sort and Merge Sort have different strengths and weaknesses. Quick Sort is faster in practice for small to moderately sized arrays and is memory-efficient due to its in-place nature. Merge Sort, on the other hand, offers stable performance across all input scenarios, making it a reliable choice. The choice between the two depends on the specific requirements of the task at hand, including memory constraints, input characteristics, and desired stability.

5. [5 x 2 = 10]

a) Write the algorithm to build a MIN-HEAP. Describe your Algorithm to build a MIN-HEAP from the array A = {5, 7, 8, 2, 1, 0, 3, 9, 4, 5, 6} in a step-by-step process. Derive the time complexity of building a MIN-HEAP. Assume the root is at index 1.

Scheme:

- Algorithm or Program or function : 2 Marks
- Building a Min Heap (step by step): 2 Marks
- Correct Time Complexity derivation: 1 Mark
- Wrong answer, but explanation approaches to answer: Step Marking

Answer:

<u>Algorithm MIN-HEAPIFY(A, i)</u>

```
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ Heap-Size(A) and A[l] < A[i]
   then smallest ←l
else smallest ←i
if r ≤ Heap-Size(A) and A[r] < A[smallest]
   then smallest ←r
if smallest ≠ i
   then exchange A[i] ↔ A[smallest]
         MIN-HEAPIFY(A, smallest)
```

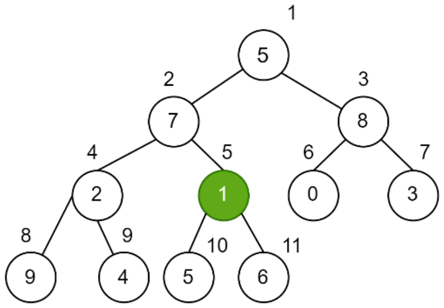
<u>Algorithm BUILD-MIN-HEAP(A)</u>

```
Heap-Size(A) = length[A]
for i ←  Floor (length[A] /2 ) down to 1
    do MIN-HEAPIFY(A, i)
```

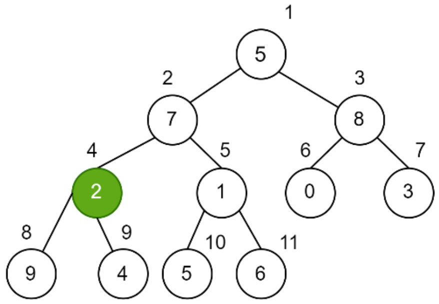
1	2	3	4	5	6	7	8	9	10	11
5	7	8	2	1	0	3	9	4	5	6

BUILD-MIN-HEAP(A)
Heap-size = 11

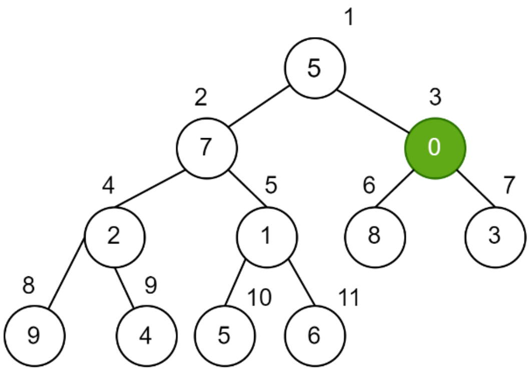
i = 5, MIN-HEAPIFY(A, 5)



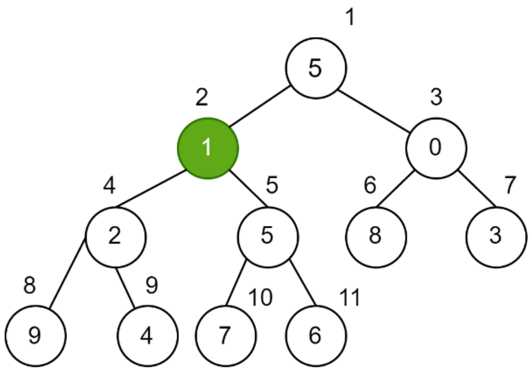
i = 4, MIN-HEAPIFY(A, 4)



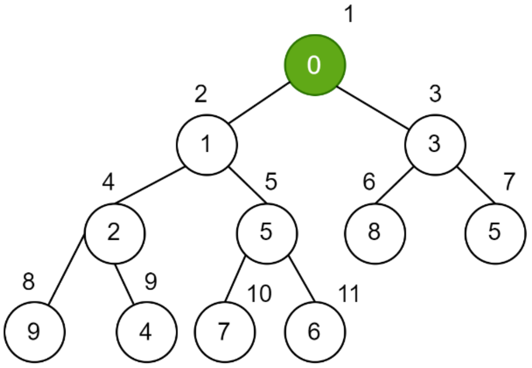
i = 3, MIN-HEAPIFY(A, 3)



i = 2, MIN-HEAPIFY(A, 2)



i = 2, MIN-HEAPIFY(A, 1)



For all nodes with varying height, the time complexity T(n)

$$= \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil * O(h)$$

$$= O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{h}{2^h} \right\rceil \right)$$

$$\leq \left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right)$$

$$= O(n)$$

$$But \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

b) Given a knapsack with a capacity of 15 kg and a set of items with their weights in kg and profits in rupees:
Items: A B C
Weights: 8 4 10
Profits: 30 15 45
Solve the fractional knapsack problem to maximize the total profits obtained in each of the following cases and find the case(s) with highest profit.

I. Greedy w.r.t. profits only
II. Greedy w.r.t. weights only
III. Greedy w.r.t. profit per unit weight

Scheme:

- Correctly written Case III: 5 marks
- Wrong calculation, but explanation approaches to answer: Step Marking

Answer:

I. Greedy w.r.t. profits only

First sort the objects/items details w.r.t. value of the profits in descending order.

Items (x)	Weight (w)	Profits (p)	Selection
C	10	45	1
A	8	30	5/8
B	4	15	0

Solution vector = (⅝, 0,1)
So, total profit of this approach is ∑ Pi * Xi = (45*1) + (30 * 5/8) + (15*0)
= 45 + 150/8 + 0
= 45 + 18.75
= **63.75**

II. Greedy w.r.t weights only

First sort the objects/items details w.r.t. value of the weights in ascending order.

Items (x)	Weight (w)	Profits (p)	Selection
B	4	15	1
A	8	30	1
C	10	45	3/10

Solution vector = (1, 1, 3/10)

So, total profit of this approach is ∑ Pi * Xi = (15*1) + (30 *1) + (45* 3/10)
= 15 + 30 + 135/10
= 45 + 13.5
= 58.5

III. Greedy w.r.t. profit per unit weight

First sort the objects/items details w.r.t. value of their calculated profit per unit weight in descending order.

Items (x)	Weight (w)	Profits (p)	Profit/weight	Selection
C	10	45	4.5	1
A	8	30	3.75	5/8
B	4	15	3.75	0

Solution vector = (⅝, 0, 1)

So, total profit of this approach is ∑ Pi * Xi = (45*1) + (30 * 5/8) + (15*0)
= 45 + 150/8 + 0
= 45 + 18.75
= 63.75

Another order of selection:

Items (x)	Weight (w)	Profits (p)	Profit/weight	Selection
C	10	45	4.5	1
B	4	15	3.75	1
A	8	30	3.75	1/8

Solution vector = (1/8, 1, 1)

So, total profit of this approach is ∑ Pi * Xi = (45*1) + (15 * 1) + (30*1/8)
= 45 + 15 + 3.75
= 63.75