Introduction to Algorithm and Asymptotic Notations

Unit: 1

Design & Analysis of Algorithms

B.Tech 4th Sem

Renuka Sharma
Assistant Professor
CSE

Brief Introduction About Me

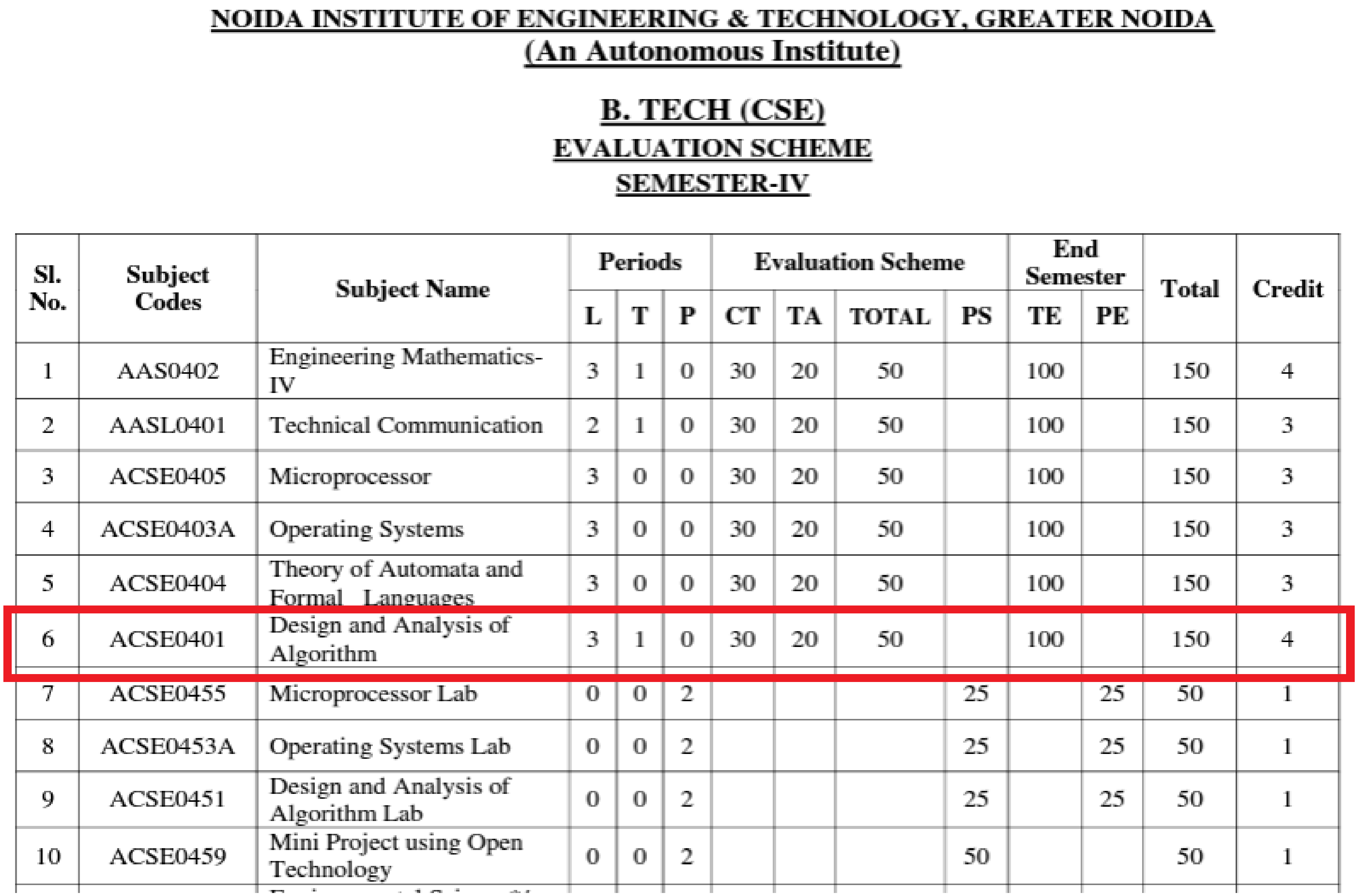
Name : Mini Jain

Designation : Assistant Professor IT & M.Tech Integrated
NIET Greater Noida

Qualification:

- B.Tech R.B.S Engg. College Agra in 2018.
- M.Tech R.B.S Engg. College Agra in 2020.

Evaluation Scheme

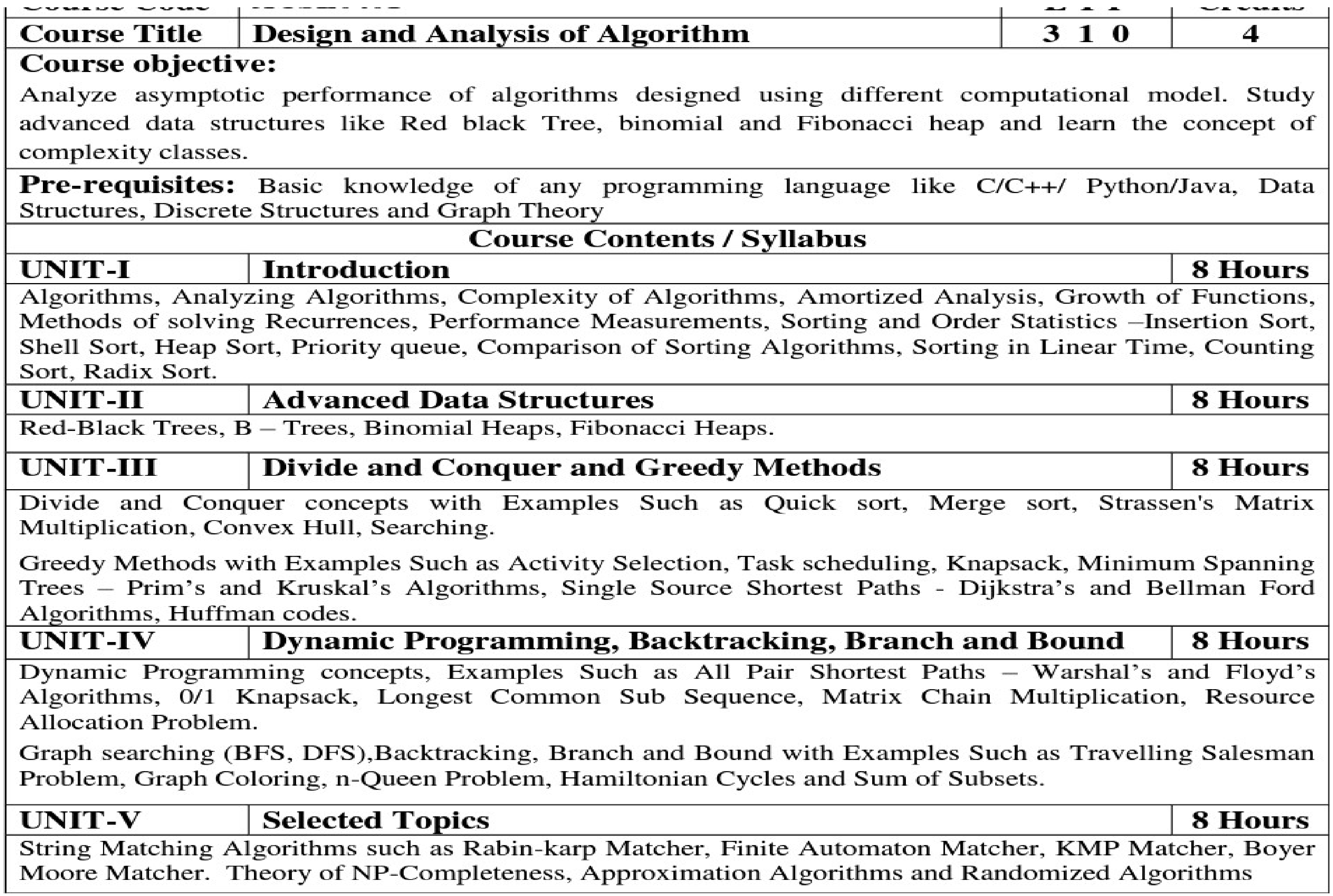


NOIDA INSTITUTE OF ENGINEERING & TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)

B. TECH (CSE)
EVALUATION SCHEME
SEMESTER-IV

Sl. No.	Subject Codes	Subject Name	Periods			Evaluation Scheme				End Semester		Total	Credit
			L	T	P	CT	TA	TOTAL	PS	TE	PE		
1	AAS0402	Engineering Mathematics-IV	3	1	0	30	20	50		100		150	4
2	AASL0401	Technical Communication	2	1	0	30	20	50		100		150	3
3	ACSE0405	Microprocessor	3	0	0	30	20	50		100		150	3
4	ACSE0403A	Operating Systems	3	0	0	30	20	50		100		150	3
5	ACSE0404	Theory of Automata and Formal Languages	3	0	0	30	20	50		100		150	3
6	ACSE0401	Design and Analysis of Algorithm	3	1	0	30	20	50		100		150	4
7	ACSE0455	Microprocessor Lab	0	0	2				25		25	50	1
8	ACSE0453A	Operating Systems Lab	0	0	2				25		25	50	1
9	ACSE0451	Design and Analysis of Algorithm Lab	0	0	2				25		25	50	1
10	ACSE0459	Mini Project using Open Technology	0	0	2				50			50	1

Syllabus



Course Title	**Design and Analysis of Algorithm**	**3 1 0**	**4**
Course objective: Analyze asymptotic performance of algorithms designed using different computational model. Study advanced data structures like Red black Tree, binomial and Fibonacci heap and learn the concept of complexity classes.			
Pre-requisites: Basic knowledge of any programming language like C/C++/ Python/Java, Data Structures, Discrete Structures and Graph Theory			
Course Contents / Syllabus			
UNIT-I	**Introduction**		**8 Hours**
Algorithms, Analyzing Algorithms, Complexity of Algorithms, Amortized Analysis, Growth of Functions, Methods of solving Recurrences, Performance Measurements, Sorting and Order Statistics –Insertion Sort, Shell Sort, Heap Sort, Priority queue, Comparison of Sorting Algorithms, Sorting in Linear Time, Counting Sort, Radix Sort.			
UNIT-II	**Advanced Data Structures**		**8 Hours**
Red-Black Trees, B – Trees, Binomial Heaps, Fibonacci Heaps.			
UNIT-III	**Divide and Conquer and Greedy Methods**		**8 Hours**
Divide and Conquer concepts with Examples Such as Quick sort, Merge sort, Strassen's Matrix Multiplication, Convex Hull, Searching. Greedy Methods with Examples Such as Activity Selection, Task scheduling, Knapsack, Minimum Spanning Trees – Prim's and Kruskal's Algorithms, Single Source Shortest Paths - Dijkstra's and Bellman Ford Algorithms, Huffman codes.			
UNIT-IV	**Dynamic Programming, Backtracking, Branch and Bound**		**8 Hours**
Dynamic Programming concepts, Examples Such as All Pair Shortest Paths – Warshal's and Floyd's Algorithms, 0/1 Knapsack, Longest Common Sub Sequence, Matrix Chain Multiplication, Resource Allocation Problem. Graph searching (BFS, DFS),Backtracking, Branch and Bound with Examples Such as Travelling Salesman Problem, Graph Coloring, n-Queen Problem, Hamiltonian Cycles and Sum of Subsets.			
UNIT-V	**Selected Topics**		**8 Hours**
String Matching Algorithms such as Rabin-karp Matcher, Finite Automaton Matcher, KMP Matcher, Boyer Moore Matcher. Theory of NP-Completeness, Approximation Algorithms and Randomized Algorithms			

Branch wise Application

- First, we will start with the internet which is very much important for our daily life and we cannot even imagine our life without the internet and it is the outcome of clever and creative algorithms. Numerous sites on the internet can operate and falsify this huge number of data only with the help of these algorithms.

- The everyday electronic commerce activities are massively subject to our data, for example, credit or debit card numbers, passwords, OTPs, and many more. The centre technologies used incorporate public-key cryptocurrency and digital signatures which depend on mathematical algorithms.

- Even an application that doesn't need algorithm content at the application level depends vigorously on the algorithm as the application relies upon hardware, GUI, networking, or object direction and all of these create a substantial use of algorithms.

- There are some other vital use cases where the algorithm has been used such as if we watch any video on YouTube then next time we will get related-type advice as recommended videos for us.

Course Objective

- Upon completion of this course, students will be able to do the following:
- Analyze the asymptotic performance of algorithms.
- Write rigorous correctness proofs for algorithms.
- Demonstrate a familiarity with major algorithms and data structures.
- Apply important algorithmic design paradigms and methods of analysis.
- Synthesize efficient algorithms in common engineering design situations.

Course Outcome

At the end of the semester, the student will be able:

	Description	Bloom's Taxonomy
CO 1	To have knowledge of basic principles of algorithm design and Analysis, asymptotic notations and growth of functions for time and space complexity analysis and applying the same in different sorting algorithms	Knowledge, analysis And design
CO 2	To apply different problem-solving approaches for advanced data structures	Knowledge, analysis And apply
CO 3	To apply divide and conquer method for solving merge sort, quick sort, matrix multiplication and Greedy Algorithm for solving different Graph Problem.	Knowledge, analysis and Apply
CO 4	To analyze and apply different optimization techniques like dynamic programming, backtracking and Branch & Bound to solve the complex problems	Knowledge, Analysis And Apply
CO 5	To understand the advanced concepts like NP Completeness and Fast Fourier Transform, to analyze and apply String Matching, Approximation and Randomized Algorithms to solve the complex problems	Knowledge, Analysis and Apply

Program Outcome

At the end of the semester, the student will be able:

POs	Engineering Graduates will be able to
PO1	Engineering Knowledge
PO2	Problem Analysis
PO3	Design & Development of solutions
PO4	Conduct Investigation of complex problems
PO5	Modern Tool Usage
PO6	The Engineer and Society
PO7	Environment and sustainability
PO8	Ethics
PO9	Individual & Team work
PO10	Communication
PO11	Project management and Finance
PO12	Life Long Learning

CO-PO and PSO Mapping

Design and Analysis of Algorithm (kCS-502)												
CO.K	PO 1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO1 1	PO12
ACSE0401.1	3	3	3	3	2	-	-	-	2	2	-	3
ACSE0401.2	3	3	3	3	2	2	-	1	1	1	-	3
ACSE0401.3	3	3	2	3	3	2	-	2	1	1	2	3
ACSE0401.4	3	3	3	3	2	2	-	2	2	1	3	3
ACSE0401.5	2	2	2	2	2	2	-	2	1	1	1	2
Average	2.8	2.8	2.6	2.8	2.2	1.6	-	1.8	1.4	1.2	1.2	2.8

Program Educational Objectives(PEOs)

PEO1: To have an excellent scientific and engineering breadth so as to comprehend, analyze, design and provide sustainable solutions for real-life problems using state-of-the-art technologies.

PEO2: To have a successful career in industries, to pursue higher studies or to support enterpreneurial endeavors and to face global challenges.

PEO3: To have an effective communication skills, professional attitude, ethical values and a desire to learn specific knowledge in emerging trends, technologies for research, innovation and product development and contribution to society.

PEO4: To have life-long learning for up-skilling and re-skilling for successful professional career as engineer, scientist, enterpreneur and bureaucrat for betterment of society

Result Analysis

End Semester Question Paper Template

B TECH

(SEM-V) THEORY EXAMINATION 20__-20__

DESIGN AND ANALYSIS OF ALGORITHMS

Time: 3 Hours **Total Marks: 100**

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.

SECTION A

1

Q.No.	**Question**	**Marks**	**CO**
1		2	
2		2	
.		.	
10		2	

End Semester Question Paper Templates

SECTION B

2. Attempt any three of the following: 3 x 10 = 30

Q.No.	Question	Marks	CO
1		10	
2		10	
.		.	
5		10	

3. Attempt any one part of the following: 1 x 10 = 10

Q.No.	Question	Marks	CO
1		10	
2		10	

NIET Greater Noida GET FUTURE READY Estd 2001 AN AUTONOMOUS INSTITUTE

End Semester Question Paper Templates

4. Attempt any one part of the following: 1 x 10 = 10

Q.No.	Question	Marks	CO
1		10	
2		10	

5. Attempt any one part of the following: 1 x 10 = 10

Q.No.	Question	Marks	CO
1		10	
2		10	

6. Attempt any one part of the following: 1 x 10 = 10

Q.No.	Question	Marks	CO
1		10	
2		10	

Prerequisite and Recap

- Prerequisite
- Basic concept of c programming language.
- Concept of stack, queue and link list.

- **Recap**
- Flow Chart
- Algorithm

Introduction About Subject With Videos

- The word Algorithm means “a process or set of rules to be followed in calculations or other problem-solving operations”. Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.

- https://youtu.be/6hfOvs8pY1k

Unit Content

- Introduction to Algorithm
- Characteristics of Algorithm
- Analysis of Algorithm
- Asymptotic Notations
- Recurrence Relation
- Sorting and order Statistics
- Shell sort,
- Heap sort,
- Sorting in linear time

Unit Objective

.

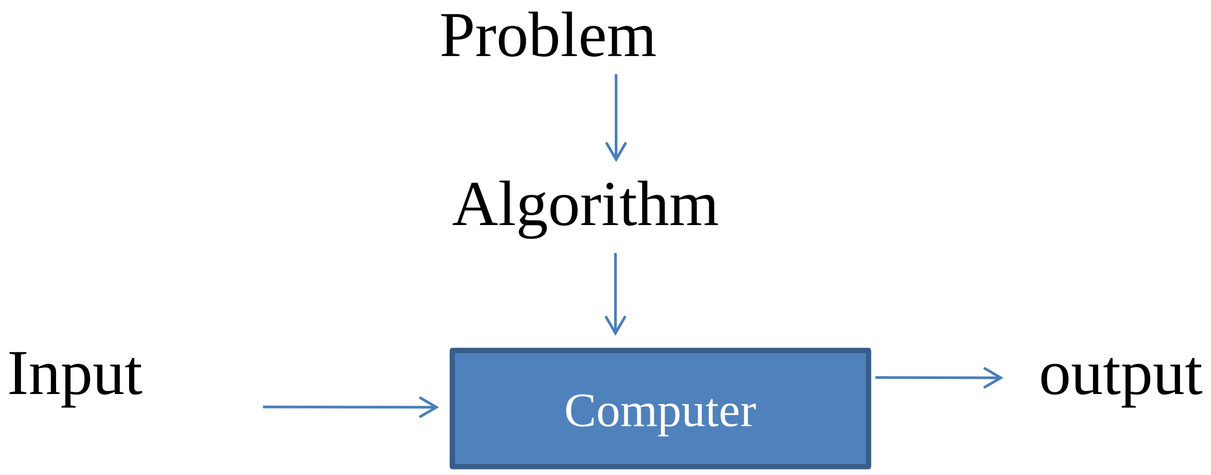
- This is an introductory chapter of Design & Analysis of Algorithm covering the concept, importance and characteristics of algorithms. The complexity and its calculation has been explained. Further, recursion and different methodologies to solve them have also been provided.
- In other part of unit different sorting algorithm – Insertion, Heap sort, Quick sort, bucket sort, Radix Sort and Counting sort have been discussed along with their complexities

Topic Objective

- The objective of this topic is to make students understand about
- Need of algorithm
- Problem statement
- Why study algorithm
- Characteristics of Algorithm

Introduction to the Algorithm(CO1)

An algorithm provides a step-by-step method for solving a computational problem, algorithms are not dependent on a particular programming language, machine, system, or compiler.



Algorithm

The Study of algorithm is necessary from the following point of view.

- Algorithm help us to understand scalability.
- Efficient algorithm leads to efficient program.
- Efficient program clear and unabiguous
- Efficient program make better use of hardware.

Characteristics of Algorithm

- Correctness
- Efficiency
- Simplicity
- Generality
- Non Ambiguity

Algorithm

All algorithms must satisfy following criteria-:

- Input
- Output
- Definiteness
- Finiteness
- Effectiveness

Algorithm

- An algorithm is a set of steps of operations to solve a problem.
- An algorithm is an efficient method that can be expressed within finite amount of time and space.
- Algorithm is independent from any programming languages.
- Algorithm design include creating an efficient algorithm to solve a problem in an efficient way using minimum time and space.
- Time is directly proportional to the number of operations on a program.

Analysis of Algorithm(CO1)- Objective

The objective of this topic is to make students understand about

- Space Complexity
- Time Complexity
- Best Case
- Worst Case
- Average Case

Prerequisite and Recap

- Prerequisite
- Definition of algorithm

- **Recap**
- Characteristics of algorithm

Analysis of Algorithm(CO1)

- The term **"analysis of algorithms"** was coined by Donald Knuth.
- To estimate the complexity function for arbitrarily large input.
- Analysis of algorithms is the determination of the amount of time and space resources required to execute it.
- The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as **time complexity**, or volume of memory, known as **space complexity**.
- The main concern of analysis of algorithms is the required time or performance

Analysis of Algorithm(CO1)

How to measure time

Count total number of fundamental operation in program and this total will give the rough estimate of the running time in terms of input size.

- **Best Case**

  The minimum number of steps taken on any instance of size n.

- **Average Case**

  An average number of steps taken on any instance of size n.

- **Worst Case**

  The maximum number of steps taken on any instance of size n.

Asymptotic Notations(CO1)- Objective

The objective of this topic is to make students understand about

- Five Asymptotic notations
  - Big Oh Notation
  - Big Omega Notation
  - Big Theta Notation
  - Small Oh Notation
  - Small Omega Notation

Prerequisite and Recap

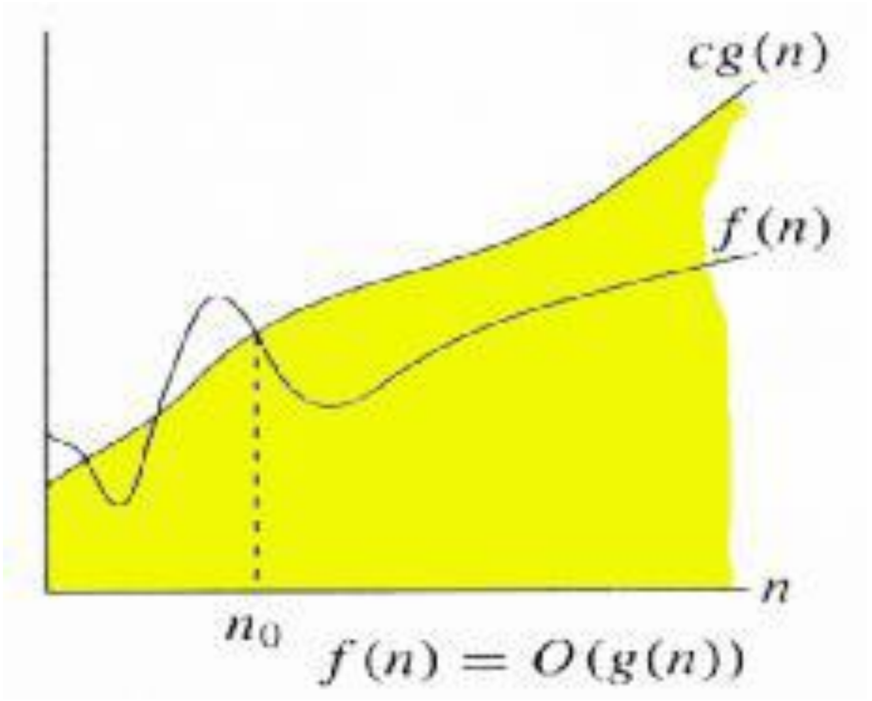
- Prerequisite
- Definition of algorithm

- **Recap**
- Cases of analysis of algorithm

Big Oh Notation

The function f(n) = O(g(n)) (read as " f of n is big oh of g of n") if there exist positive constants c and n0 such that

$f(n) \le cg(n)$ for all $n \ge n0$



Big Oh Notation

Example

Let us consider a given function,

$$f(n)=4.n^3+10.n^2+5.n+1$$

Considering $g(n)=n^3$,

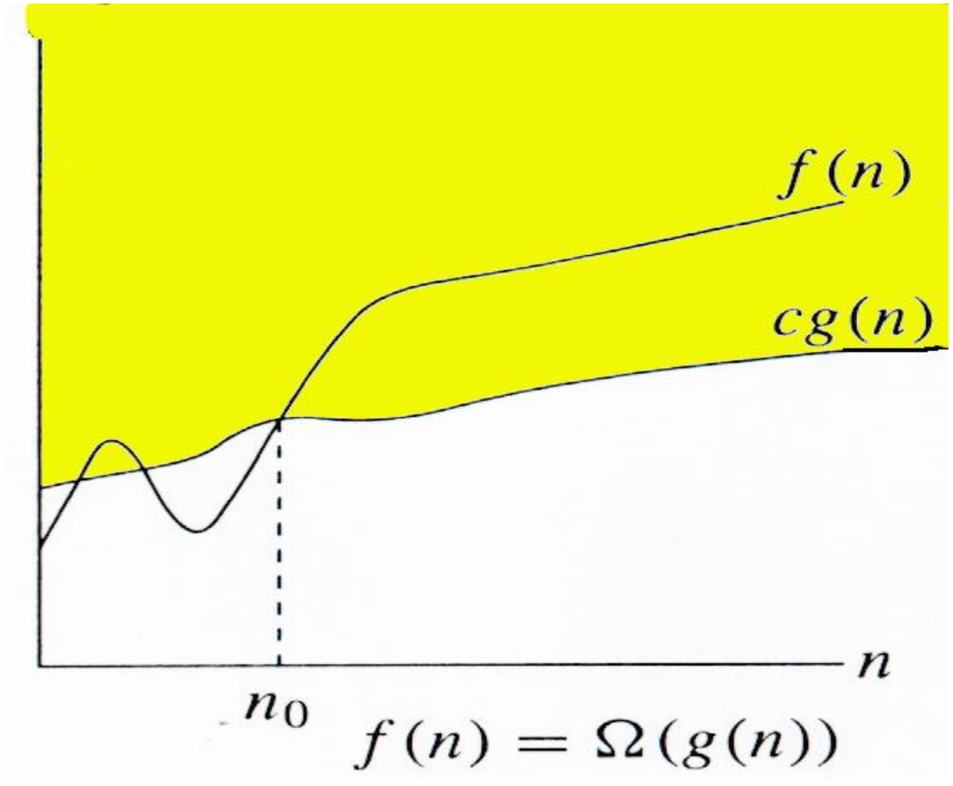
$f(n) \leqslant 5.g(n)$ for all the values of $n>2$.

Hence, the complexity of ***f(n)*** can be represented as $O(g(n))$, i.e. $O(n^3)$

Big Omega Notation

The function f(n) = Ω(g(n)) (read as “ f of n is Ω of g of n”) if there exist positive constants c and n0 such that

cg(n)≤ f(n) for all n≥ n0



Big Omega Notation

Example

Let us consider a given function,

$$f(n)=4.n^3+10.n^2+5.n+1$$

Considering $g(n)=n^3$,

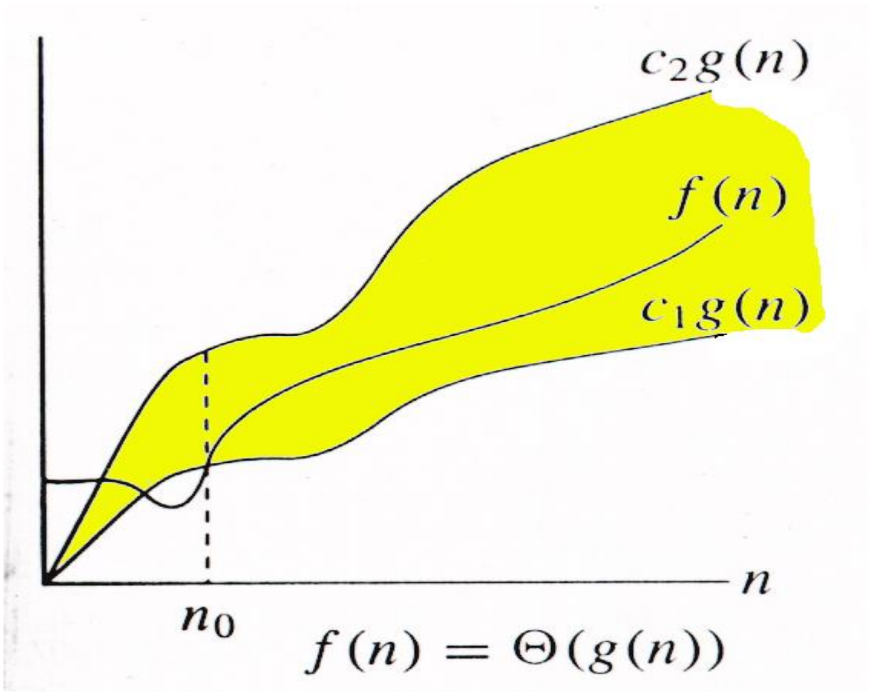
$f(n) \geq 4.g(n)$ for all the values of $n>0$.

Hence, the complexity of ***f(n)*** can be represented as $\Omega(g(n))$, i.e. $\Omega(n^3)$

Big Theta Notation

The function f(n) = Θ(g(n)) (read as " f of n is theta of g of n") if there exist positive constants c1 and c2 and n0 such that

c1g(n)≤f(n)≤c2g(n) for all n≥ n0



Big Theta Notation

Example

Let us consider a given function,

$$f(n)=4.n^3+10.n^2+5.n+1$$

Considering $g(n)=n^3$,

$4.g(n) \leqslant f(n) \leqslant 5.g(n)$ for all the large values of **n**.

Hence, the complexity of ***f(n)*** can be represented as $\theta(g(n))$, i.e. $\theta(n^3)$

Small Oh Notation

- Upper bound that is not asymptotically tight

  $f(n) = o(g(n))$ if $f(n) < c(g(n))$ for all $n \geq n0$

- $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = 0$$

- $g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Small Oh Notation

Example

Let us consider a given function,

$$f(n)=4.n^3+10.n^2+5.n+1$$

Considering $g(n)=n^4$,

$$\lim_{n\to\infty} (4.n^3+10.n^2+5.n+1/ n^4)=0$$

Hence, the complexity of ***f(n)*** can be represented as $o(g(n))$, i.e. $o(n^4)$

Small Omega Notation

- lower bound that is not asymptotically tight

  f(n) = ω(g(n)) if c(g(n))< f(n) for all n≥ n0

- *f*(*n*) becomes arbitrarily large relative to *g*(*n*) as *n* approaches infinity:

$$\lim_{n\to\infty} [f(n) / g(n)] = \infty.$$

- *g*(*n*) is a ***lower bound*** for *f*(*n*) that is not asymptotically tight.

Small Omega Notation

Example

Let us consider a given function,

$$f(n)=4.n^3+10.n^2+5.n+1$$

Considering $g(n)=n^2$,

$$\lim_{n\to\infty} (4.n^3+10.n^2+5.n+1/ n^2)=0$$

Hence, the complexity of ***f(n)*** can be represented as

ω (g(n)), i.e. $\omega(n^4)$

Recursion (CO1)- Objective

The objective of this topic is to make students understand about

- Different methods of solving recursion
- Substitution/ Iteration method
- Recursion Tree method
- Master method

Prerequisite and Recap

- Prerequisite
- Algorithm
- Mathematical concepts of limits and summation

- **Recap**
- Asymptotic notations

Recursion (CO1)

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- Using recursive algorithm, certain problems can be solved quite easily.
- Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.
- The running time of recursive algorithms is represented by an equation that describes a function in terms of its value on smaller functions. For eg.

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

Recursion

There are three methods to solve recurrence

- Substitution/ Iteration method
- Recursion Tree method
- Master method

Recursion

Substitution/ Iteration method

Guess the form of the answer by experience and creativity or by some heuristics guess the form of the solution. And then use induction to find the constants and show that solution works.

Example Solve $T(n)=T(n-1) + 1$ and $T(1) = \Theta(1)$

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

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

$T(n) = T(n-4) +1+1+1+1$

And so on $T(n) = T(n-k) + k$

Where $n-k =1 \Rightarrow n-k=1$

Thus $T(n)= T(1) + n-1 \Rightarrow \Theta(n)$

Recursion

Recursion Tree Method

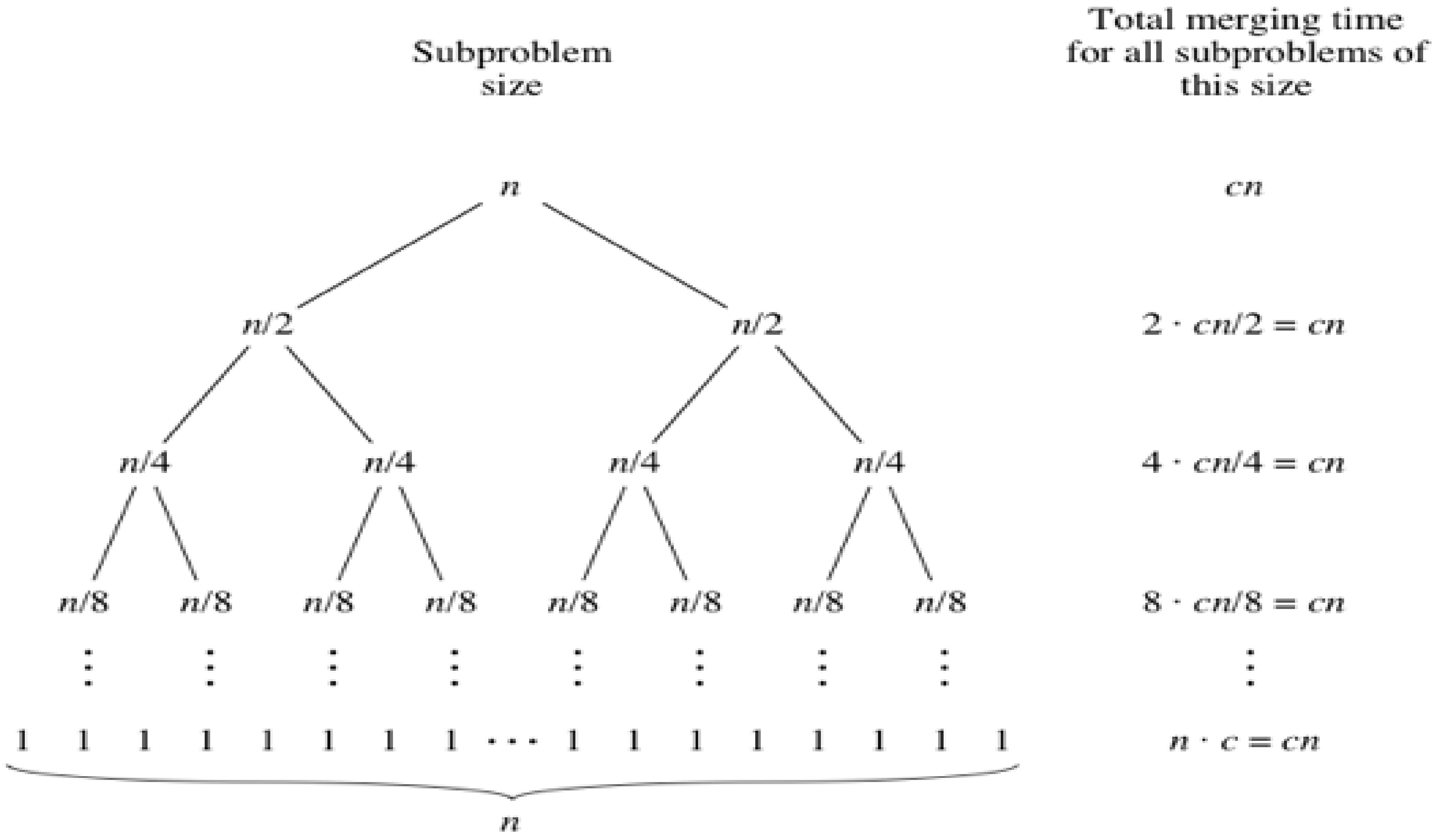
Each node represents the cost of a single sub problem.

- Sum up the costs with each level to get level cost.
- Sum up all the level costs to get total cost.
- Particularly suitable for divide-and-conquer recurrence.
- Best used to generate a good guess.
- If trying carefully to draw the recursion-tree and compute cost, then used as direct proof.

Example Recursion Tree of $T(n)=2T(n/2)+ C$

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

Recursion


Subproblem size
Total merging time for all subproblems of this size
n
cn
n/2
n/2
2 · cn/2 = cn
n/4
n/4
n/4
n/4
4 · cn/4 = cn
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
8 · cn/8 = cn
1 1 1 1 1 1 1 1 · · · 1 1 1 1 1 1 1 1
n · c = cn
n

Recursion

Master method

- Master Method is a direct way to get the solution.
- The master method applies to recurrences of the form

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

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Recursion

- There are three cases:
- Case I $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

  $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

Example

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

  Here $a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n$.

  CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

  $\therefore\ T(n) = \Theta(n^2)$.

Recursion

- Case II $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.

  $f(n)$ and $n^{\log_b a}$ grow at similar rates.

  Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

  Example

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

  Here $a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2.$

  CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

  $\therefore\ T(n) = \Theta(n^2 \lg n)$.

Recursion

Case III $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.

$f(n)$ grows polynomials faster than $n^{\log_b a}$ (by an n^{ε} factor),

and $f(n)$ satisfies the ***regularity condition*** that $a\, f(n/b) \le c\, f(n)$ for some constant $c < 1$.

Solution: $T(n) = \Theta(f(n))$.

Example

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

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$ a***nd*** $4(cn/2)^3 \le cn^3$ for $c =$ 1/2.

$\therefore\ T(n) = \Theta(n^3).$

Recursion

Examples of some standard algorithms whose time complexity can be evaluated using Master Method

Merge Sort: $T(n) = 2T(n/2) + \Theta(n)$. It falls in case 2 as c is 1 and $\mathrm{Log}_b a]$ is also 1. So the solution is $\Theta(n \mathrm{Log} n)$

Binary Search: $T(n) = T(n/2) + \Theta(1)$. It also falls in case 2 as c is 0 and $\mathrm{Log}_b a$ is also 0. So the solution is $\Theta(\mathrm{Log} n)$

Sorting(CO1)- Objective

The objective of this topic is to make students understand about

- Different Sorting Tehniques
- Insertion sorting
- Heap Sort
- Counting Sort
- Bucket Sort
- Radix Sort

Prerequisite and Recap

- Prerequisite
- Algorithm
- loops- for while
- C programming

- **Recap**
- Recursion

Sorting(CO1)

Insertion Sort

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.

- The array is virtually split into a sorted and an unsorted part.

- Values from the unsorted part are picked and placed at the correct position in the sorted part.

- Characteristics of Insertion Sort:
  - This algorithm is one of the simplest algorithm with simple implementation
  - Basically, Insertion sort is efficient for small data values

Sorting(CO1)

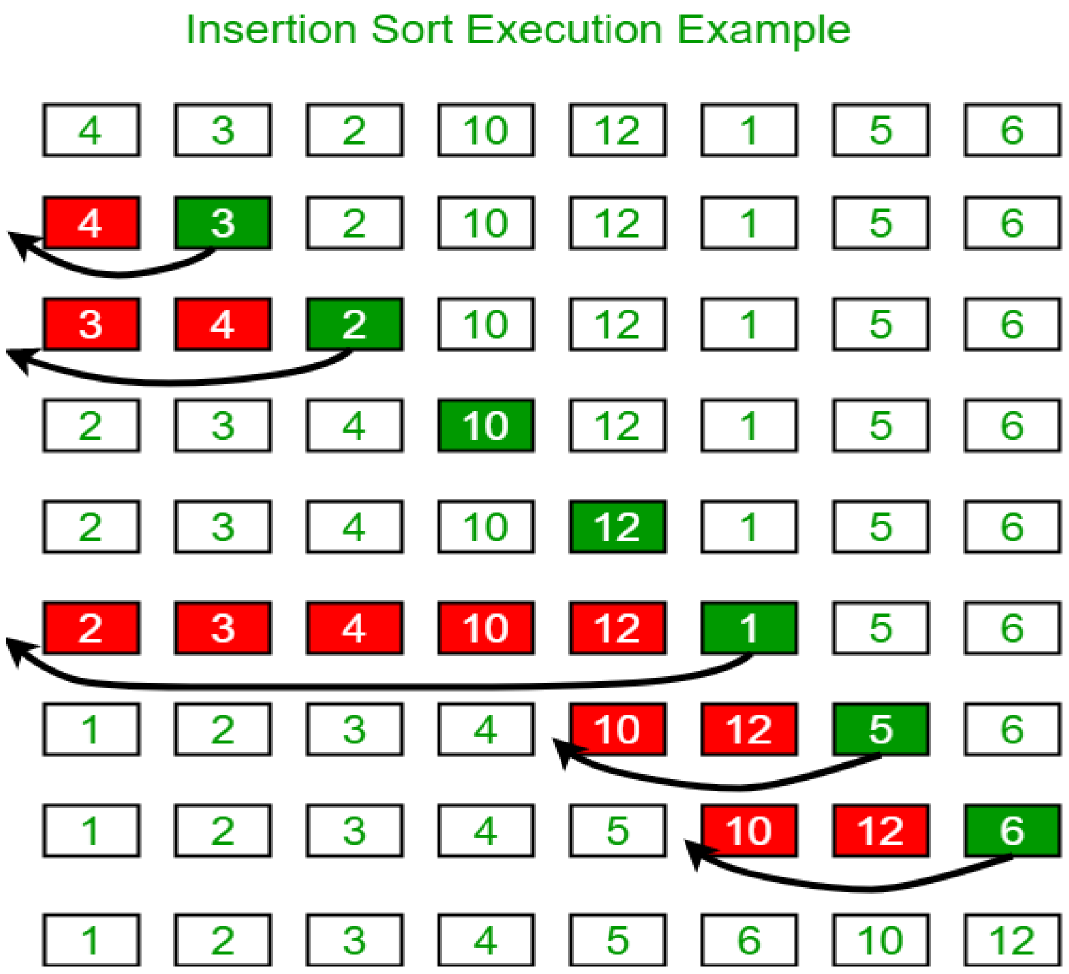
Insertion Sort Algorithm

Algorithm: We use a procedure INSERTION_SORT. It takes as parameters an array A[1.. n] and the length n of the array.

INSERTION_SORT (A)

```
1. FOR j ← 2 TO length[A]
2.   DO key ← A[j]
3. //{Put A[j] into the sorted sequence A[1 . . j − 1]}
4.    i ← j − 1
5. WHILE i > 0 and A[i] > key
6.    DO A[i +1] ← A[i]
7.        i ← i − 1
8. A[i + 1] ← key
```

Sorting(CO1)


Insertion Sort Execution Example
4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Sorting(CO1)

Insertion Sort

Three Cases:

Best-Case The best case occurs if the array is already sorted. the while-loop in line 5 executed only once for each *j. This happens if given array A is already* sorted. T(*n*) $= an + b = O(n)$ It is a linear function of *n.*

Worst case The worst-case occurs if the array is sorted in reverse order i.e., in decreasing order. the worst-case occurs, when line 5 executed *j times for each j. This can happens if array A starts* out in reverse order T(*n*) $= an^2 + bn + c = O(n^2)$

Average case: When half of the array is sorted and half of the array is unsorted it produces O(n^2)

Sorting(CO1)

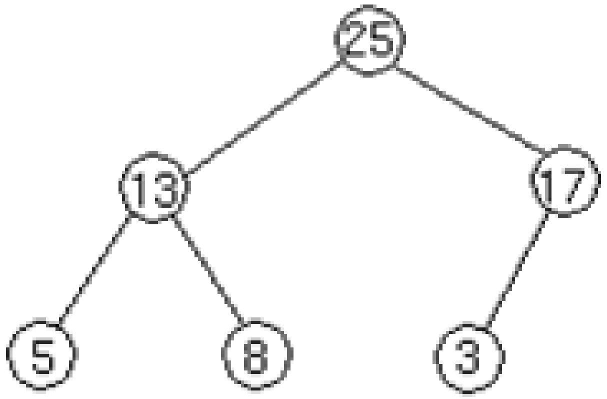
Heap Sort

- Heap sort is a comparison based sorting technique based on Binary Heap data structure.
- The binary heap data structure is an array that can be viewed as almost complete binary tree.
- Each node of the binary tree corresponds to an element of the array.
- It is similar to selection sort where we first find the maximum element and place the maximum element at the end.
- We repeat the same process for remaining element.

Sorting(CO1)

Heap Sort

- **For Example**
- We represent heaps in level order, going from left to right.



- The array corresponding to the heap above is [25, 13, 17, 5, 8, 3].

Sorting(CO1)

Heap Sort

The root of the tree A[1] and given index *i of a node, the indices of its parent, left child and right* child can be computed as

PARENT (i) = return floor($i/2$)

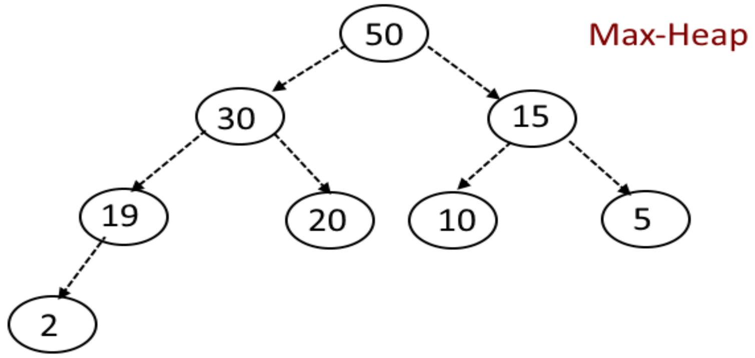
LEFT (i) = return $2i$

RIGHT (i) = return $2i + 1$

Sorting(CO1)

Heap Property

1. **Max-heap property:** A max-**heap** is a **binary** tree such that. - the data contained in each node is greater than (or equal to) the data in that node's children. ie. for every node i other than the root A[PARENT(i)] >= A[i]



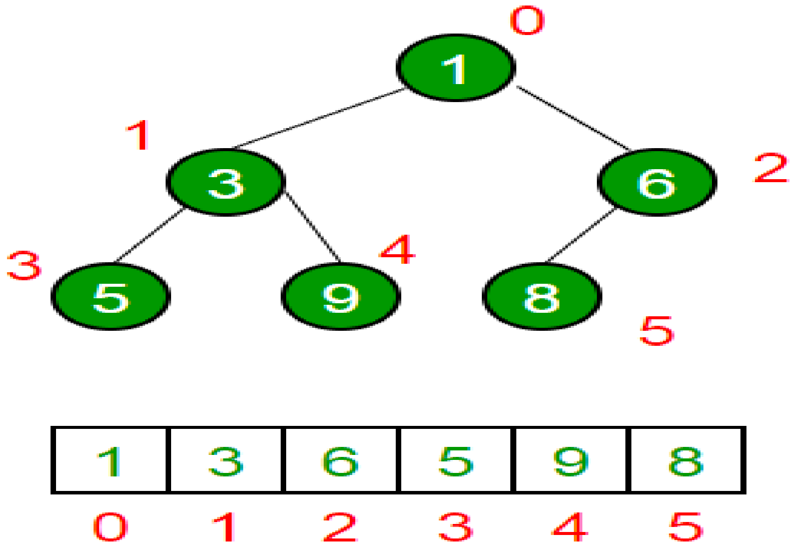
0	1	2	3	4	5	6	7	8
	50	30	15	19	20	10	5	2

for Node at i : Left child will be 2i and right child will be at 2i+1 and parent node will be at [i/2].

Sorting(CO1)

2. Min-heap property: A **min-heap** is a **binary** tree such that. - the data contained in each node is less than (or equal to) the data in that node's children. - the **binary** tree is complete
for every node i other than the root

A[PARENT(i)] <= A[i]



Sorting(CO1)

Four basic procedures on heap are

1. Heapify, which runs in O(lg *n) time.*

2. Build-Heap, which runs in linear time.

3. Heap Sort, which runs in O(*n lg n) time*

Sorting(CO1)

Heapify (*A, i)*

```
1. $l \leftarrow$ left [i]
2. $r \leftarrow$ right [i]
3. if $l \le$ heap-size [A] OR A[l] > A[i]
4.     then largest $\leftarrow l$
5.     else largest $\leftarrow i$
6. if $r \le$ heap-size [A] OR A[r] > A[largest]
7.     then largest $\leftarrow r$
8. if largest $\ne i$
9.     then exchange A[i] $\leftrightarrow$ A[largest]
10. Heapify (A, largest)
```

Sorting(CO1)

Build Max Heap (*A)*

```
1. heap-size [A] ← length [A]
2. For i ← lower bound (length [A])/2 down to 1
3.        Do Max Heapify [A, i]
```

Sorting(CO1)

Characteristics of Heap Sort Algorithm

- The heap sort combines the best of both merge sort and insertion sort. Like merge sort, the worst case time of heap sort is O(*n log n) and like insertion sort, heap sort sorts in-place.*

- *The* heap sort algorithm starts by using procedure BUILD-HEAP to build a heap on the input array A[1 . . *n].*

- *Since the maximum element of the array stored at the root A[1], it can be put into its* correct final position by exchanging it with *A[n] (the last element in A).*

- *If we now discard node* n from the heap than the remaining elements can be made into heap. Note that the new element at the root may violate the heap property.

Sorting(CO1)

Heap Sort Algorithm

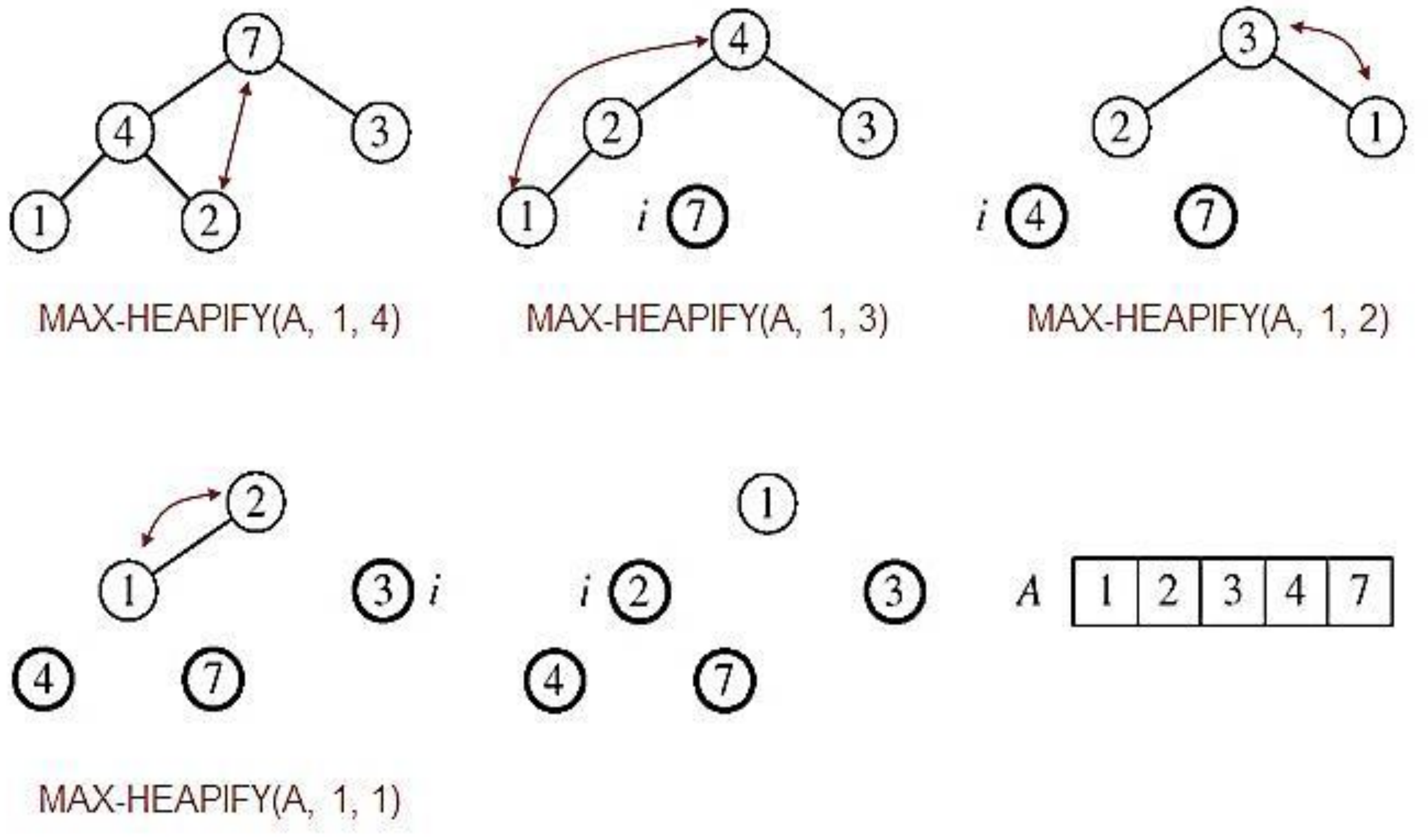
HEAPSORT (A)

```
1. BUILD_HEAP (A)
2. for i ← length (A) down to 2 do
exchange A[1] ↔ A[i]
heap-size [A] ← heap-size [A] - 1
Heapify (A, 1)
```

The HEAPSORT procedure takes time O(*n lg n), since the call to BUILD_HEAP takes* time *O(n) and each of the n -1 calls to Heapify takes time O(lg n).*

Sorting(CO1)

Example: A=[7, 4, 3, 1, 2]



Sorting(CO1)

Counting Sort

- **Counting sort** is a stable **sorting** technique, which is used to **sort** objects according to the keys that are small numbers.

- It counts the number of keys whose key values are same.

- This **sorting** technique is effective when the difference between different keys are not so big, otherwise, it can increase the space complexity

- It works by counting the number of objects having distinct key values (kind of hashing).

- Then there will be some arithmetic to calculate the position of each object in the output sequence.

Sorting(CO1)

Counting Sort

Depends on a *key assumption: numbers to be sorted are integers in {0, 1, . . . , k}.*

- Input: *A[1 . . n], where A[j] Î {0, 1, . . . , k} for j = 1, 2, ..., n.* Array *A and values n and k are given as parameters.*

- Output: *B[1 . . n], sorted. B is assumed to be already allocated* and is given as a parameter.

- Auxiliary storage: *C[0 . . k]*

NIET Greater Noida GET FUTURE READY

Sorting(CO1)

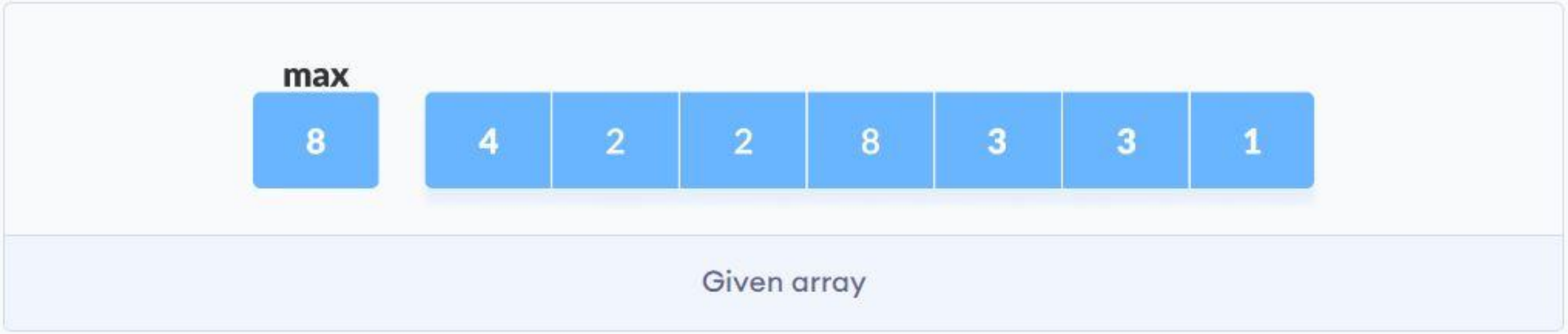
COUNTING_SORT (A, B, k)

```
1. for i ← 0 to k do
2.       c[i] ← 0
3. for j ← 1 to length[A] do
4.       c[A[j]] ← c[A[j]] + 1
5. //c[i] now contains the number of elements equal to i
6. for i ← 1 to k do
7.       c[i] ← c[i] + c[i-1]
8. // c[i] now contains the number of elements ≤ i
9. for j ← length[A] downto 1 do
10.      B[c[A[j]]] ← A[j]
11.      c[A[j]] ← c[A[j]] – 1
```

Analysis-The average time **complexity** for **Bucket Sort** is O(n + k). The worst time **complexity** is $O(n^2)$. The space **complexity** for **Bucket Sort** is O(n+k).

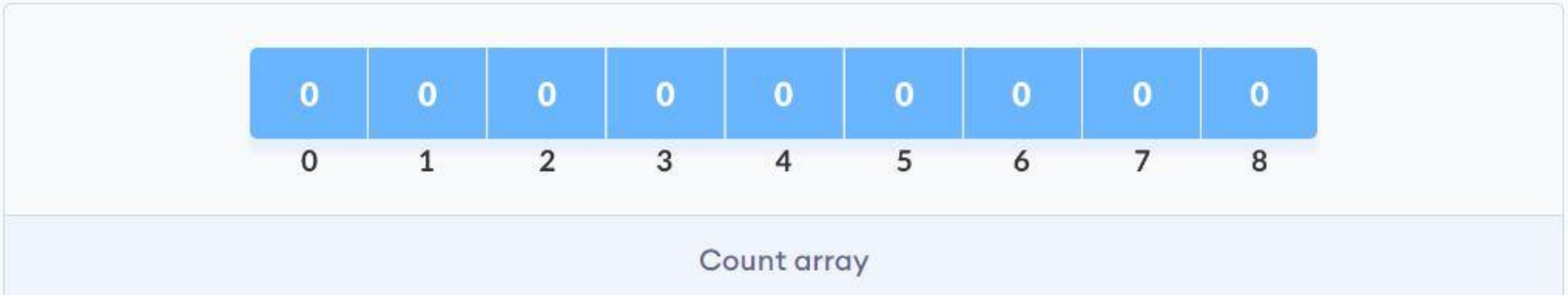
Sorting(CO1)

1. Find out the maximum element (let it be `max`) from the given array.



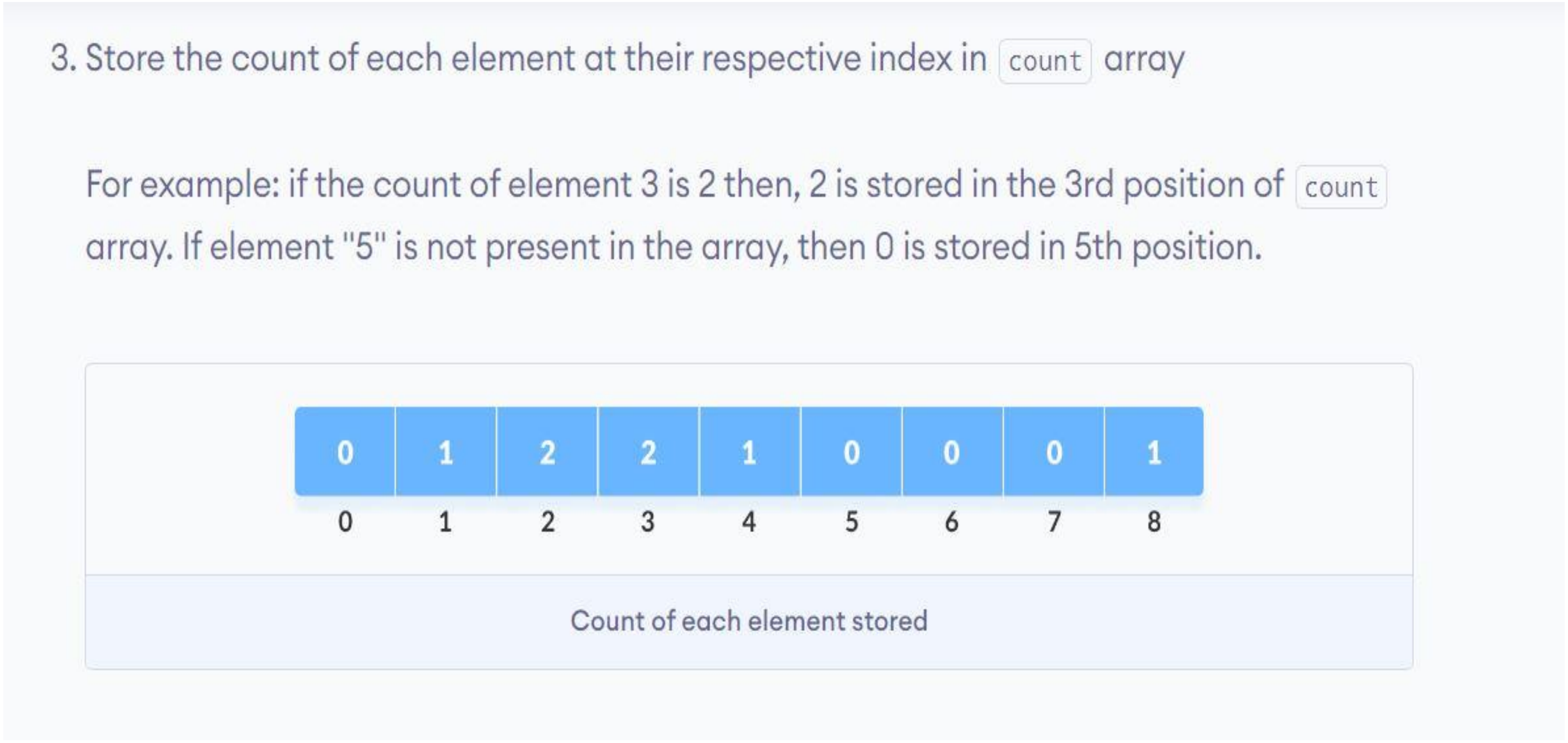
Given array

2. Initialize an array of length `max+1` with all elements 0. This array is used for storing the count of the elements in the array.



Count array

Counting Sort



3. Store the count of each element at their respective index in `count` array

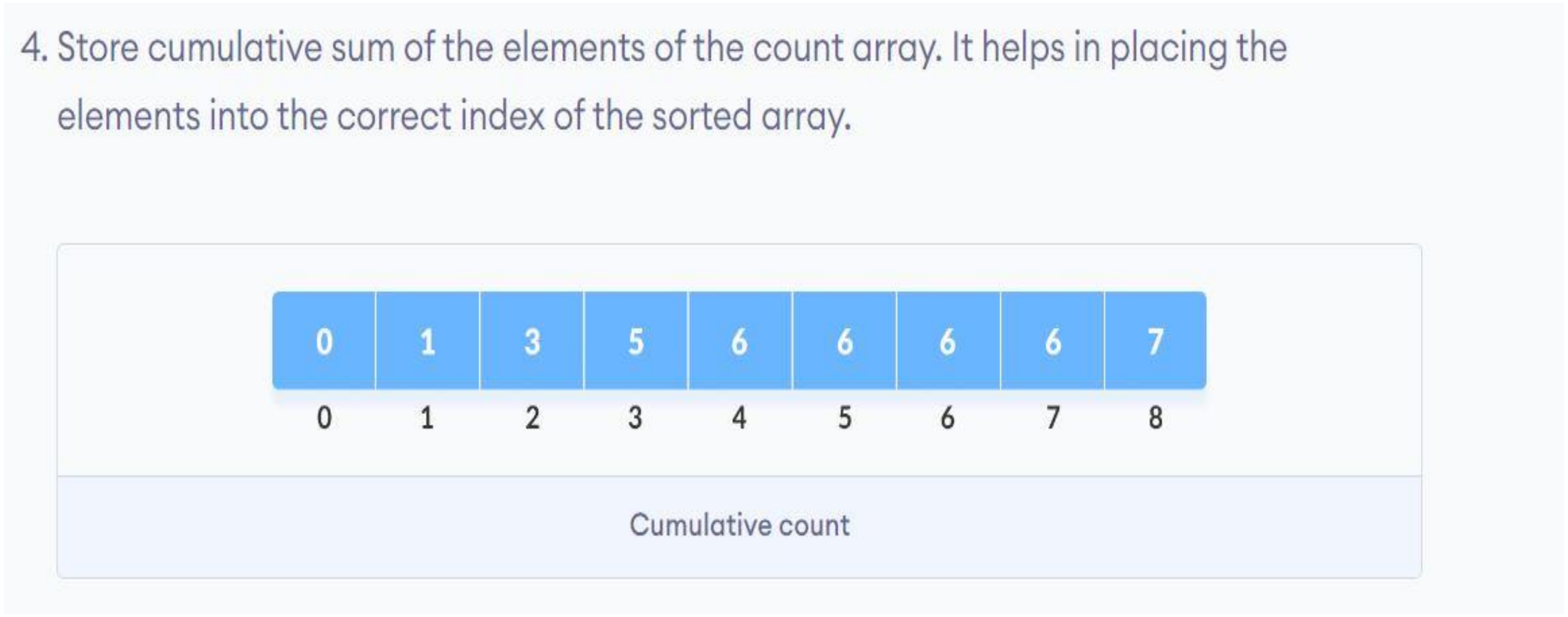
For example: if the count of element 3 is 2 then, 2 is stored in the 3rd position of `count` array. If element "5" is not present in the array, then 0 is stored in 5th position.

0	1	2	3	4	5	6	7	8
0	1	2	2	1	0	0	0	1

Count of each element stored

Sorting(CO1)

Counting Sort



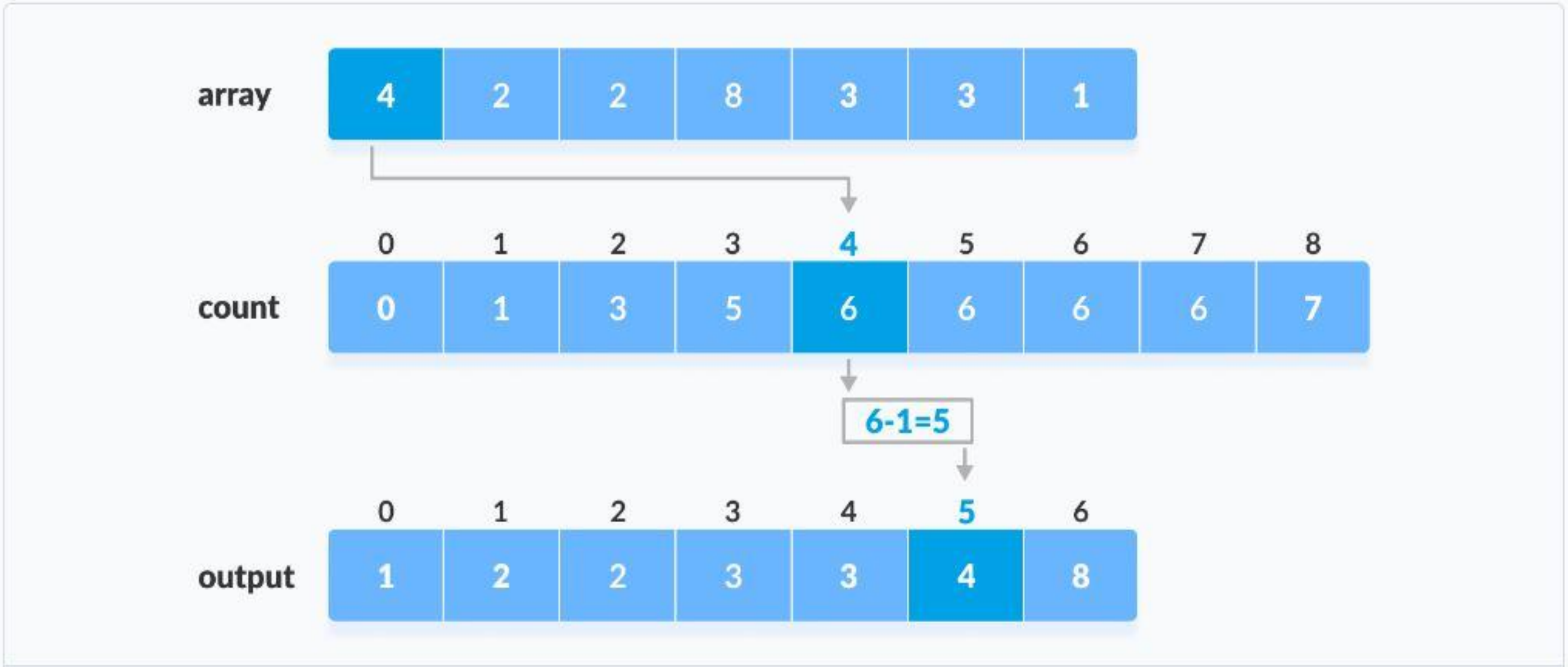
4. Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.

0	1	3	5	6	6	6	6	7
0	1	2	3	4	5	6	7	8

Cumulative count

Sorting(CO1)

5. Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated as shown in figure below.



Counting sort

Radix Sort

- The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit.
- Radix sort uses counting sort as a subroutine to sort.
- Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.

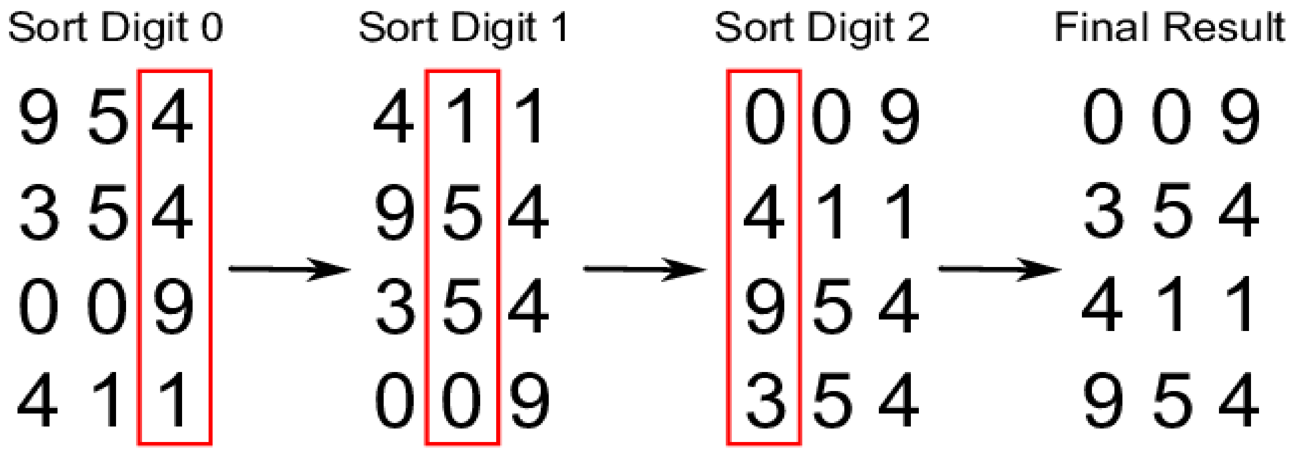
- **To sort *d digits:***

```
RADIX_SORT (A, d)
    for i ← 1 to d do
        use a stable sort to sort A on digit i
  // counting sort will do the job
```

Sorting(CO1)

Radix Sort:

Following example shows how Radix sort operates on four 3-digits number.



Radix sort complexity is O(kn) for n keys which are integers of word size k. For all there cases time i.e best , worst and average **time complexity** is O(kn).

Sorting(CO1)

Bucket Sort

- Bucket sort, is a sorting algorithm that works by partitioning an array into a number of buckets.
- Each bucket is then sorted individually, by using a different sorting algorithm.
- Divide [0, 1) into n equal-sized buckets.
- Distribute the n input values into the buckets.
- Sort each bucket.
- Then go through buckets in order, listing elements in each one

Sorting(CO1)

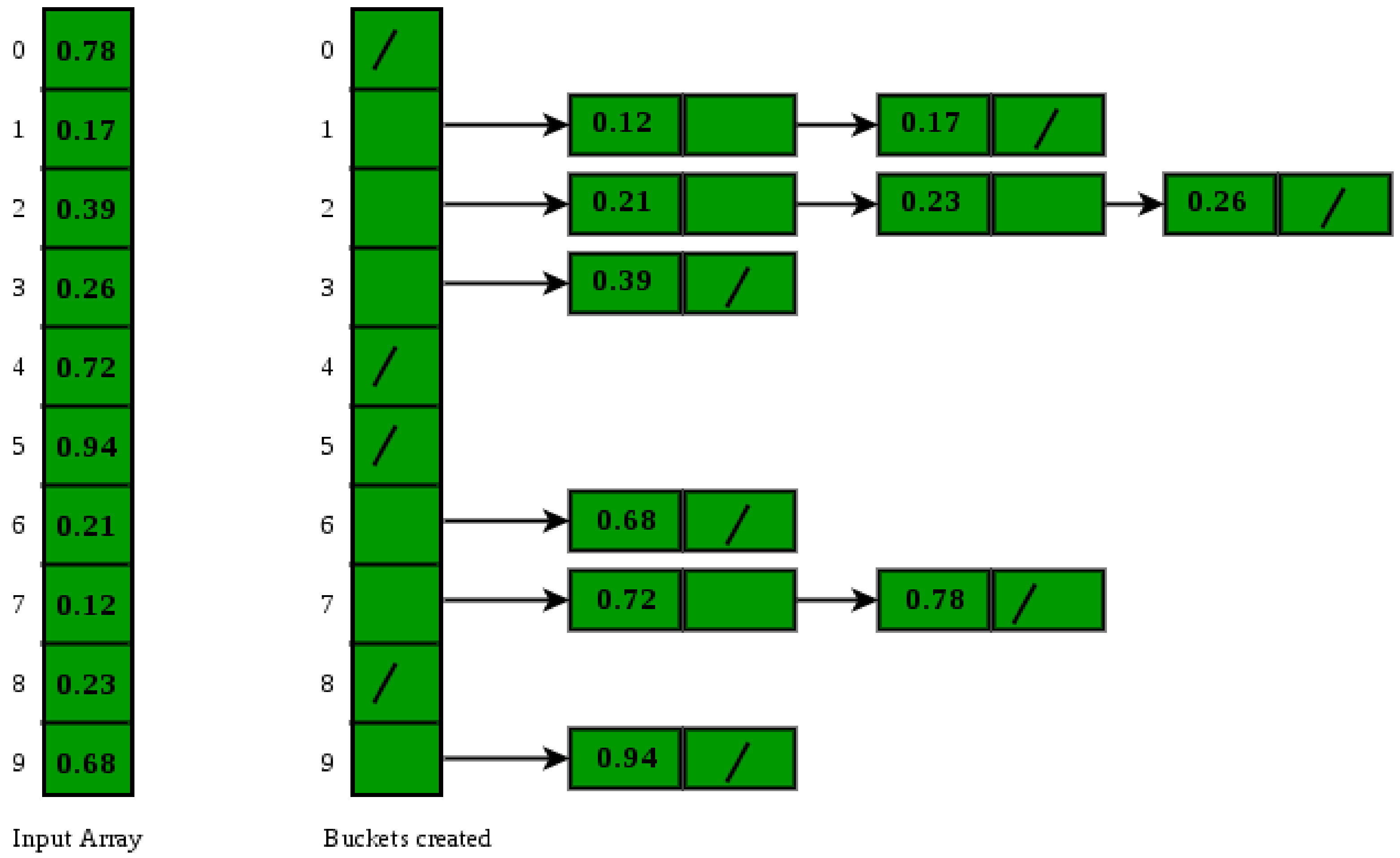
BUCKET_SORT (A)

```
1. n ← length [A]
2. For i = 1 to n do
3. Insert A[i] into list B[nA[i]]
4. For i = 0 to n-1 do
5. Sort list B with Insertion sort
6. Concatenate the lists B[0], B[1], . . B[n-1] together in order.
```

- Time Complexity: O(n + k) for best case and average case and O(n^2) for the worst case.
- Space Complexity: O(nk) for worst case

Sorting(CO1)

Bucket Sort



Daily Quiz

Q1) Process of inserting an element in stack is called _____________

Q2) Process of removing an element from stack is called ___________

Q3) Master's theorem is used for?

Q4) How many cases are there under Master's theorem?

Q5) In recursion, the condition for which the function will stop calling itself is _____________

Q6) What is the average case running time of an insertion sort algorithm?

Daily Quiz

Q7) What is the running time of an insertion sort algorithm if the input is pre-sorted?

Q8) In C, what are the basic loops required to perform an insertion sort?

Q9)Which of the following sorting algorithm is best suited if the elements are already sorted?

Q10) What is recurrence for worst case of QuickSort and what is the time complexity in Worst case?

Weekly Assignment

Q1 Solve the recurrence relation by iteration $T(n) = T(n-1) + n^4$ [CO1]

Q2 Rank the following by growth rate $2^{\lg n}$, $(\sqrt{2})^{\lg n}$, log n!, log (logn), $\log^2 n$, $(1/3)^n$, $n^{1/\lg n}$, $(3/2)^n$ [CO1]

Q3 Solve the recurrence: $T(n) = 50\,T(n/49) + \log n!$ [CO1]

Q4 Solve the following recurrence: $T(n) = \sqrt{n}\,T(\sqrt{n}) + n$ [CO1]

Q5 Use the master method to give tight asymptotic bounds for the following recurrence. $T(n) = 4T(n/2) + n$. [CO1]

Q6 Solve the recurrence $T(n) = 2\,T(\sqrt{n}) + 1$ by making a change of variables. Your solution should be asymptotically tight. Do not worry about whether values are integral [CO1]

Weekly Assignment

Q8) How will you sort following array of 5 elements using heap sort
5, 9, 1, 17 and 6. [CO1]

Q9) llustrate the operation of INSERTION-SORT on the array
A = 31, 41, 59, 26, 41, 58 [CO1]

Q10) What do you mean by ‘Stable sorting algorithms’? Quick sort is unstable where as merge is an stable sorting algorithm. Do you agree with the above statement? Justify your answer. [CO1]

Q11) Analyze the running time of quick sort in the average case.

Q12) What is time complexity of counting sort? Sort 1, 9, 3, 3, 4, 5, 6, 7, 7, 8 by counting sort. [CO1]

Q13) Find out the worst case running time of merge sort. [CO1]

Faculty Video Links, You tube & NPTEL Video Links and Online Courses Details

- **You tube/other Video Links**
- https://www.youtube.com/watch?v=yRM3sc57q0c&list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V
- https://www.youtube.com/watch?v=A03oI0znAoc
- https://www.youtube.com/watch?v=Nd0XDY-jVHs
- https://www.youtube.com/watch?v=4V30R3I1vLI
- https://www.youtube.com/watch?v=IawM82BQ4II
- https://www.youtube.com/watch?v=1K9ebQJosvo
- https://www.youtube.com/watch?v=OynWkEj0S-s
- https://www.youtube.com/watch?v=CyknhZbfMqc

Faculty Video Links, You tube & NPTEL Video Links and Online Courses Details

- You tube/other Video Links

- https://www.youtube.com/watch?v=BO145HIUHRg
- https://www.youtube.com/watch?v=mB5HXBb_HY8
- https://www.youtube.com/watch?v=6pV2IF0fgKY
- https://www.youtube.com/watch?v=7h1s2SojIRw
- https://www.youtube.com/watch?v=HqPJF2L5h9U
- https://www.youtube.com/watch?v=JMlYkE8hGJM
- https://www.youtube.com/watch?v=pEJiGC-ObQE

MCQ s

Q1. Two main measures for the efficiency of an algorithm are

a. Processor and memory b. Complexity and capacity
c. Time and space d. Data and space

Q2. The Worst case occur in linear search algorithm when

a. Item is somewhere in the middle of the array
b. Item is not in the array at all
c. Item is the last element in the array
d. Item is the last element in the array or is not there at all

Q3. The complexity of the average case of an algorithm is

a. Much more complicated to analyze than that of worst case
b. Much simpler to analyze than that of worst case
c. Sometimes more complicated and some other times simpler than that of worst case
d. None or above

MCQ s

Q4. Which of the following sorting algorithms is the fastest?

(a) Merge sort (b) Quick sort (c) Insertion sort (d) Shell sort

Q5. What is the worst case time complexity of a quick sort algorithm?

(a) O(N) (b) O(N log N) (c) $O(N^2)$ (d) O(log N)

Q6. Quick sort is a ___________

(a) greedy algorithm

(b) divide and conquer algorithm

(c) dynamic programming algorithm

(d) backtracking algorithm

Q7. What is the worst case time complexity of merge sort?

(a) O(n log n) (b) O(n2) (c) O(n2 log n) (d) O(n log n2)

Glossary Question

Q.1 The complexity of searching an element from a set of n elements using Binary search algorithm is_______

a. O(n log n)

b. O(log n)

c. O(n2) Incorrect

d. O(n)

Q.2______ is a condition that is always true at a particular point in an algorithm.

a. assertion

b. constant

c. exception

d. invariant Correct

Q.3 The running time of quick sort depends on the _________ .

a. Selection of pivot elements

b. Number of input

c. Number of passes

d. Arrangements of the elements

Glossary Question

Q.7 __________ is the worst case running time of shell sort, using Shell's increments

a) O(N)
b) O(N log N)
c) O(log N)
d) O(N^2)

Q.8 Heap sort is an implementation of ____________ using a descending priority queue.

a) insertion sort
b) selection sort
c) bubble sort
d) merge sort

Q.9 The descending heap property is ___________

a) A[Parent(i)] = A[i]
b) A[Parent(i)] <= A[i]
c) A[Parent(i)] >= A[i]
d) A[Parent(i)] > 2 * A[i]

Glossary Question

Q.10 ______is wort case time complexity of Heap sort?

a) O(nlogn)

b) O(n^2logn)

c) O(n^2)

d) O(n^3)

Q.11 In insertion sort, the average number of comparisons required to place the 7th element into its correct position is ______.

a) 9

b) 4

c) 7

d) 14

Q.12 _______ is the average case complexity of selection sort?

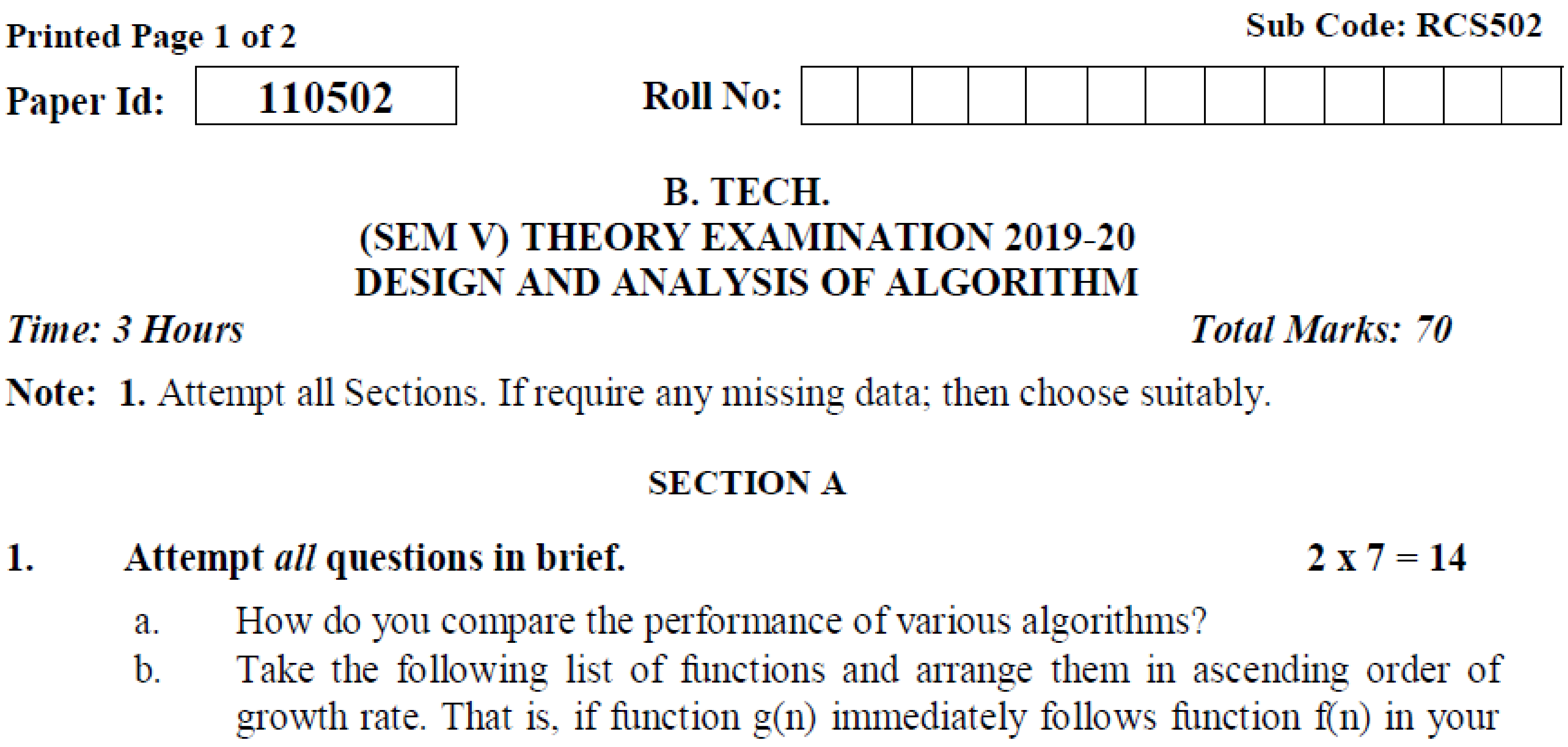
a) O(nlogn)

b) O(logn)

c) O(n)

d) O(n^2)

Old Question Papers(2019-2020)



Printed Page 1 of 2 **Sub Code: RCS502**

Paper Id:	**110502**	**Roll No:**	

B. TECH.
(SEM V) THEORY EXAMINATION 2019-20
DESIGN AND ANALYSIS OF ALGORITHM

Time: 3 Hours ***Total Marks: 70***

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.

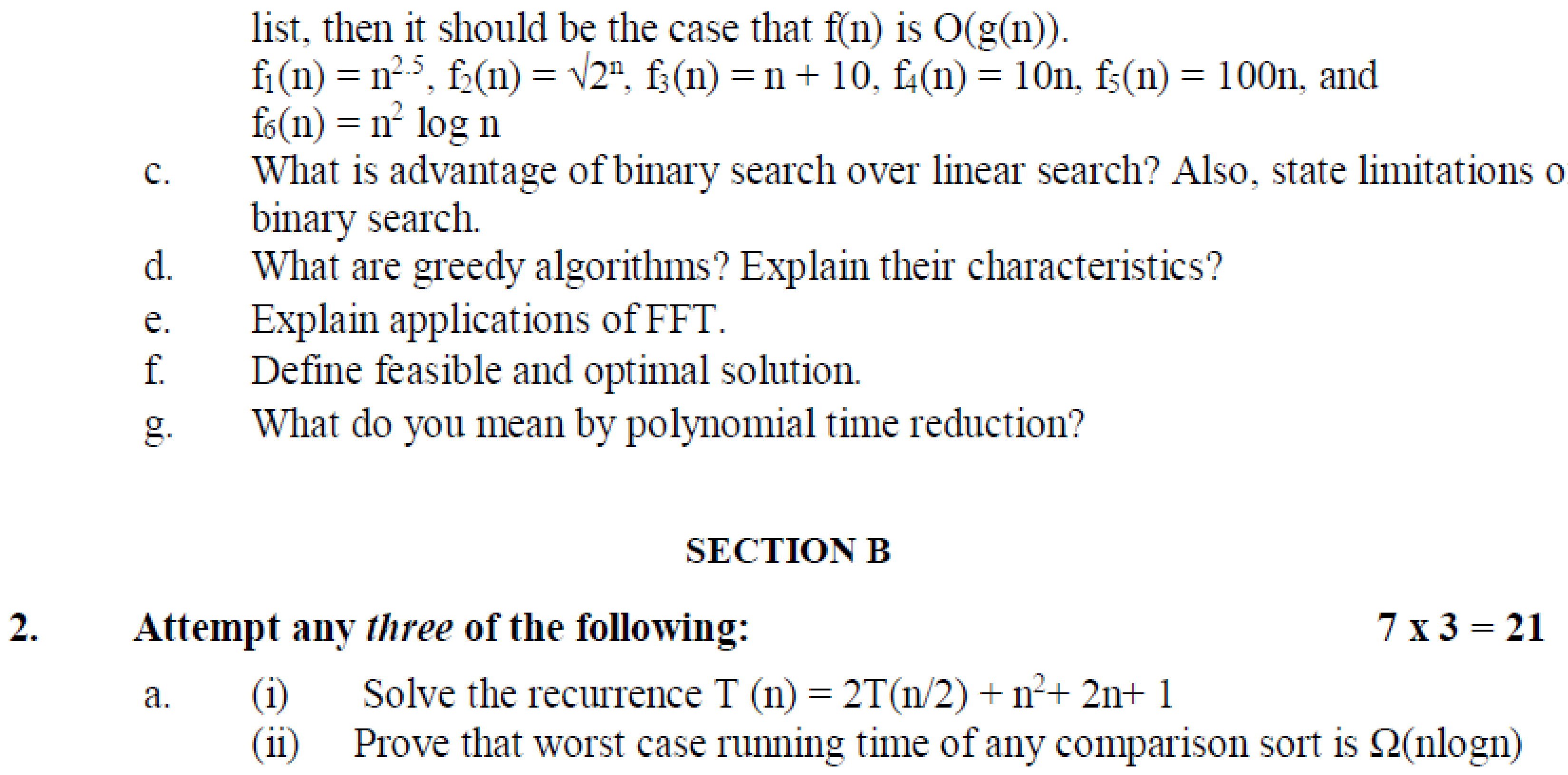
SECTION A

1. Attempt *all* questions in brief. **2 x 7 = 14**

a. How do you compare the performance of various algorithms?

b. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n) in your

Old Question Papers(2019-2020)



list, then it should be the case that f(n) is O(g(n)).
$f_1(n) = n^{2.5}$, $f_2(n) = \sqrt{2^n}$, $f_3(n) = n + 10$, $f_4(n) = 10n$, $f_5(n) = 100n$, and
$f_6(n) = n^2 \log n$

c. What is advantage of binary search over linear search? Also, state limitations of binary search.

d. What are greedy algorithms? Explain their characteristics?

e. Explain applications of FFT.

f. Define feasible and optimal solution.

g. What do you mean by polynomial time reduction?

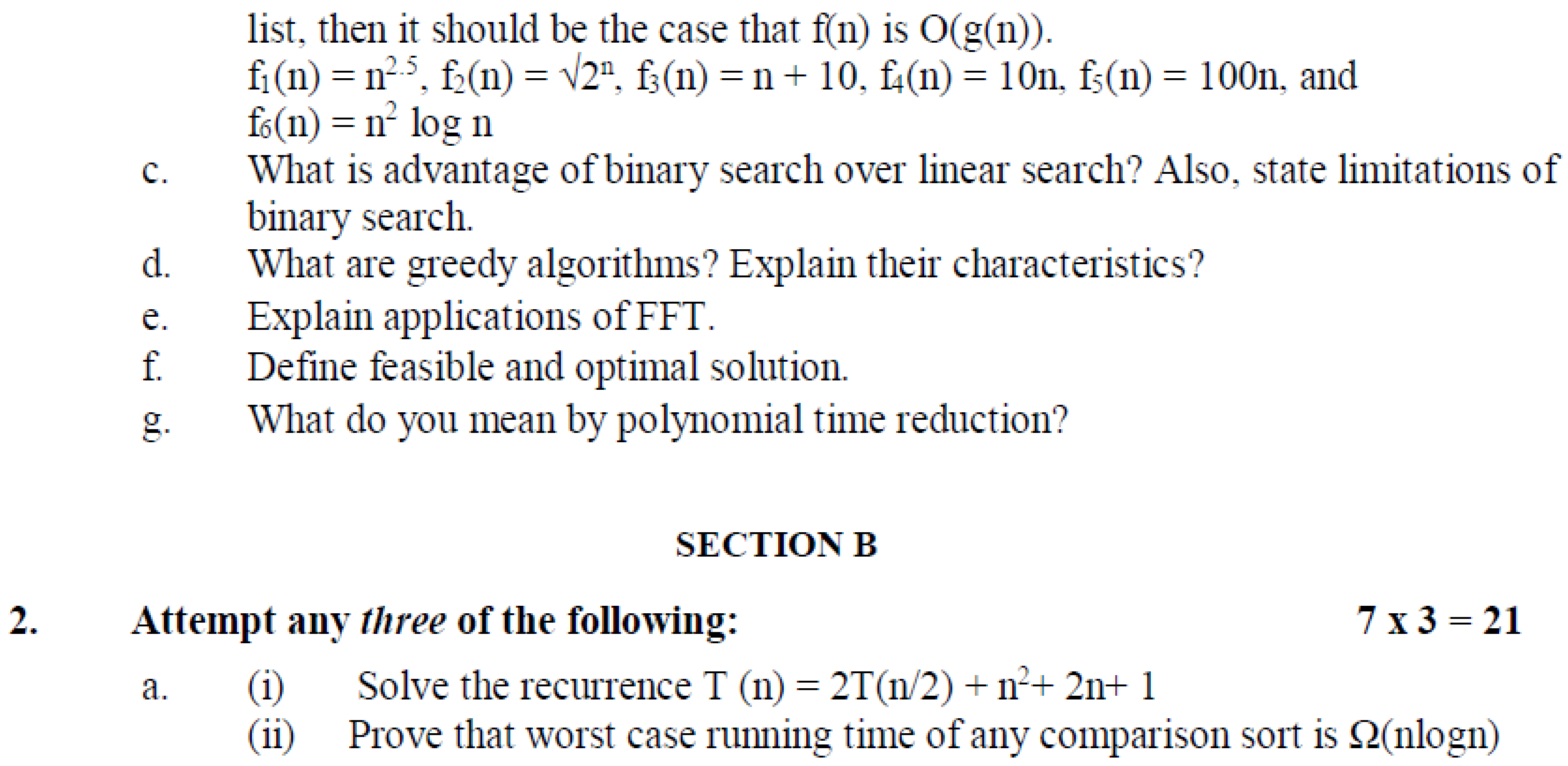
SECTION B

2. Attempt any *three* of the following: 7 x 3 = 21

a. (i) Solve the recurrence $T(n) = 2T(n/2) + n^2 + 2n + 1$

(ii) Prove that worst case running time of any comparison sort is $\Omega(n\log n)$

Old Question Papers(2019-2020)



list, then it should be the case that f(n) is O(g(n)).
$f_1(n) = n^{2.5}$, $f_2(n) = \sqrt{2^n}$, $f_3(n) = n + 10$, $f_4(n) = 10n$, $f_5(n) = 100n$, and
$f_6(n) = n^2 \log n$

c. What is advantage of binary search over linear search? Also, state limitations of binary search.

d. What are greedy algorithms? Explain their characteristics?

e. Explain applications of FFT.

f. Define feasible and optimal solution.

g. What do you mean by polynomial time reduction?

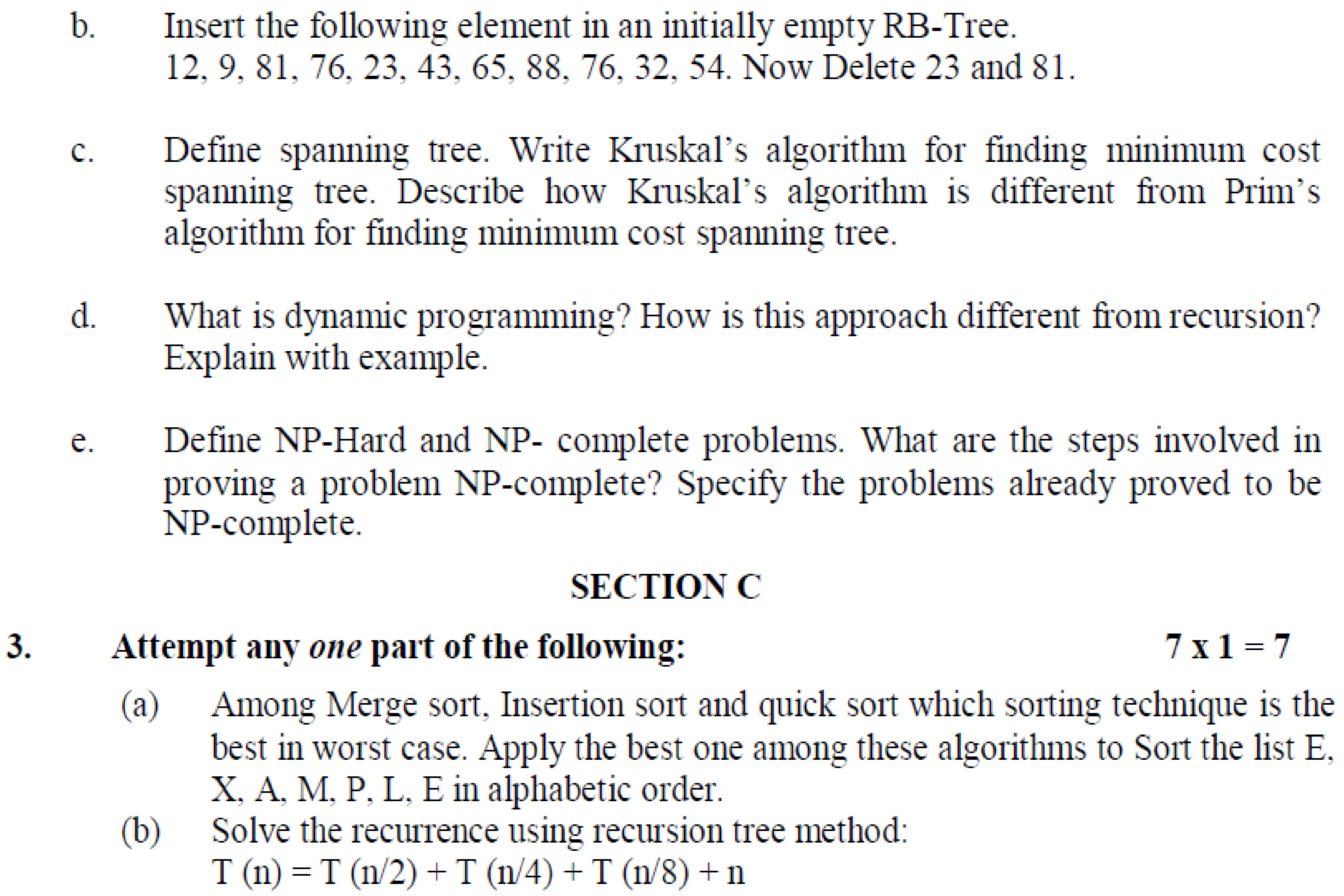
SECTION B

2. Attempt any *three* of the following: 7 x 3 = 21

a. (i) Solve the recurrence $T(n) = 2T(n/2) + n^2 + 2n + 1$

(ii) Prove that worst case running time of any comparison sort is $\Omega(n\log n)$

Old Question Papers(2019-2020)



b. Insert the following element in an initially empty RB-Tree.
12, 9, 81, 76, 23, 43, 65, 88, 76, 32, 54. Now Delete 23 and 81.

c. Define spanning tree. Write Kruskal's algorithm for finding minimum cost spanning tree. Describe how Kruskal's algorithm is different from Prim's algorithm for finding minimum cost spanning tree.

d. What is dynamic programming? How is this approach different from recursion? Explain with example.

e. Define NP-Hard and NP- complete problems. What are the steps involved in proving a problem NP-complete? Specify the problems already proved to be NP-complete.

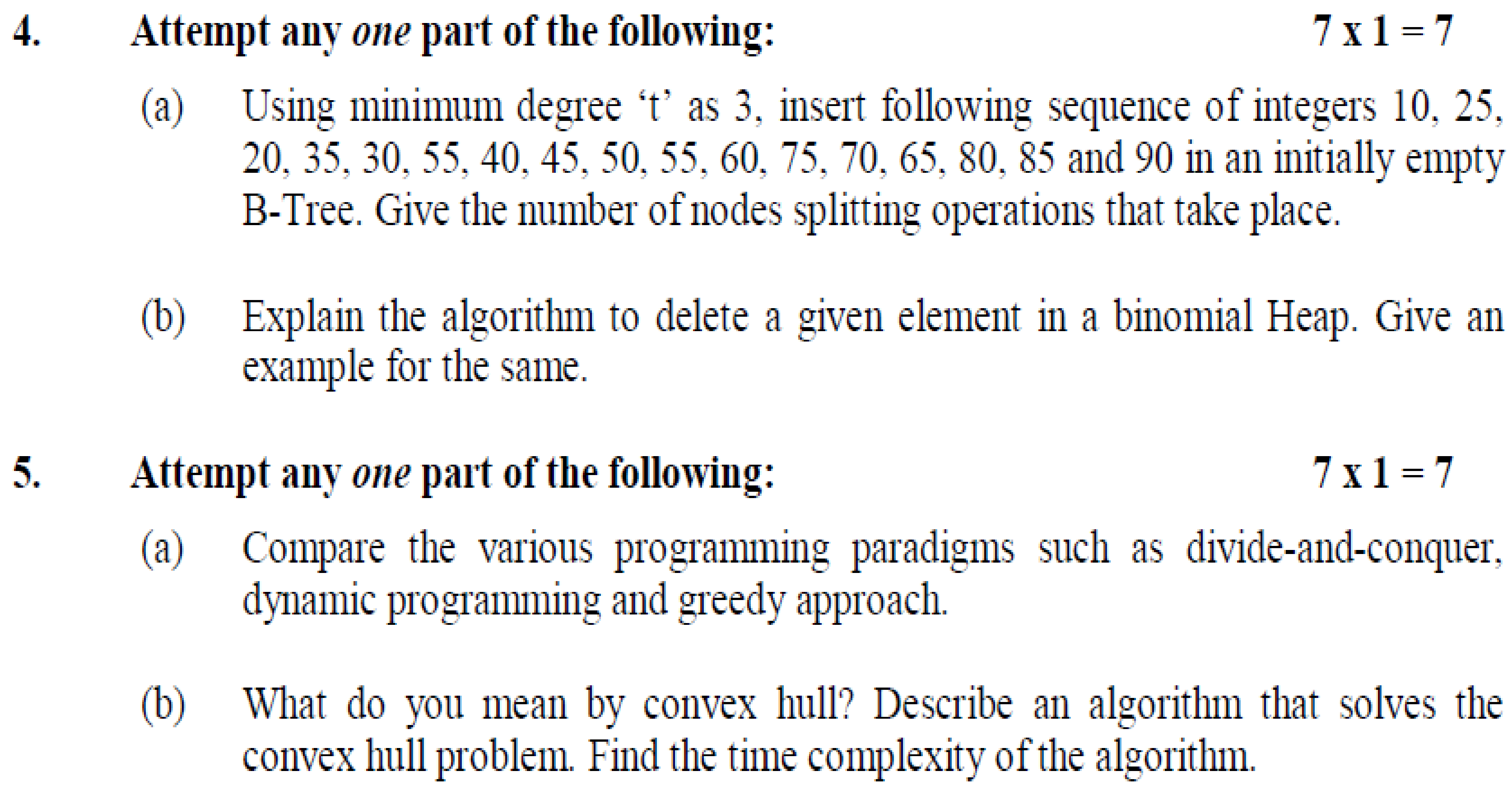
SECTION C

3. Attempt any *one* part of the following: 7 x 1 = 7

(a) Among Merge sort, Insertion sort and quick sort which sorting technique is the best in worst case. Apply the best one among these algorithms to Sort the list E, X, A, M, P, L, E in alphabetic order.

(b) Solve the recurrence using recursion tree method:
$T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Old Question Papers(2019-2020)



4. **Attempt any *one* part of the following:** **7 x 1 = 7**

(a) Using minimum degree 't' as 3, insert following sequence of integers 10, 25, 20, 35, 30, 55, 40, 45, 50, 55, 60, 75, 70, 65, 80, 85 and 90 in an initially empty B-Tree. Give the number of nodes splitting operations that take place.

(b) Explain the algorithm to delete a given element in a binomial Heap. Give an example for the same.

5. **Attempt any *one* part of the following:** **7 x 1 = 7**

(a) Compare the various programming paradigms such as divide-and-conquer, dynamic programming and greedy approach.

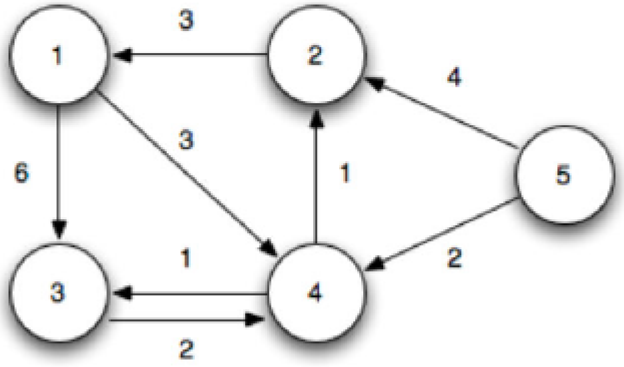
(b) What do you mean by convex hull? Describe an algorithm that solves the convex hull problem. Find the time complexity of the algorithm.

Old Question Papers(2019-2020)

6. Attempt any *one* part of the following: 7 x 1 = 7

(a) Solve the following 0/1 knapsack problem using dynamic programming. P=[11,21,31,33] w=[2,11,22,15] c=40, n=4.

(b) Define Floyd Warshall Algorithm for all pair shortest path and apply the same on following graph:

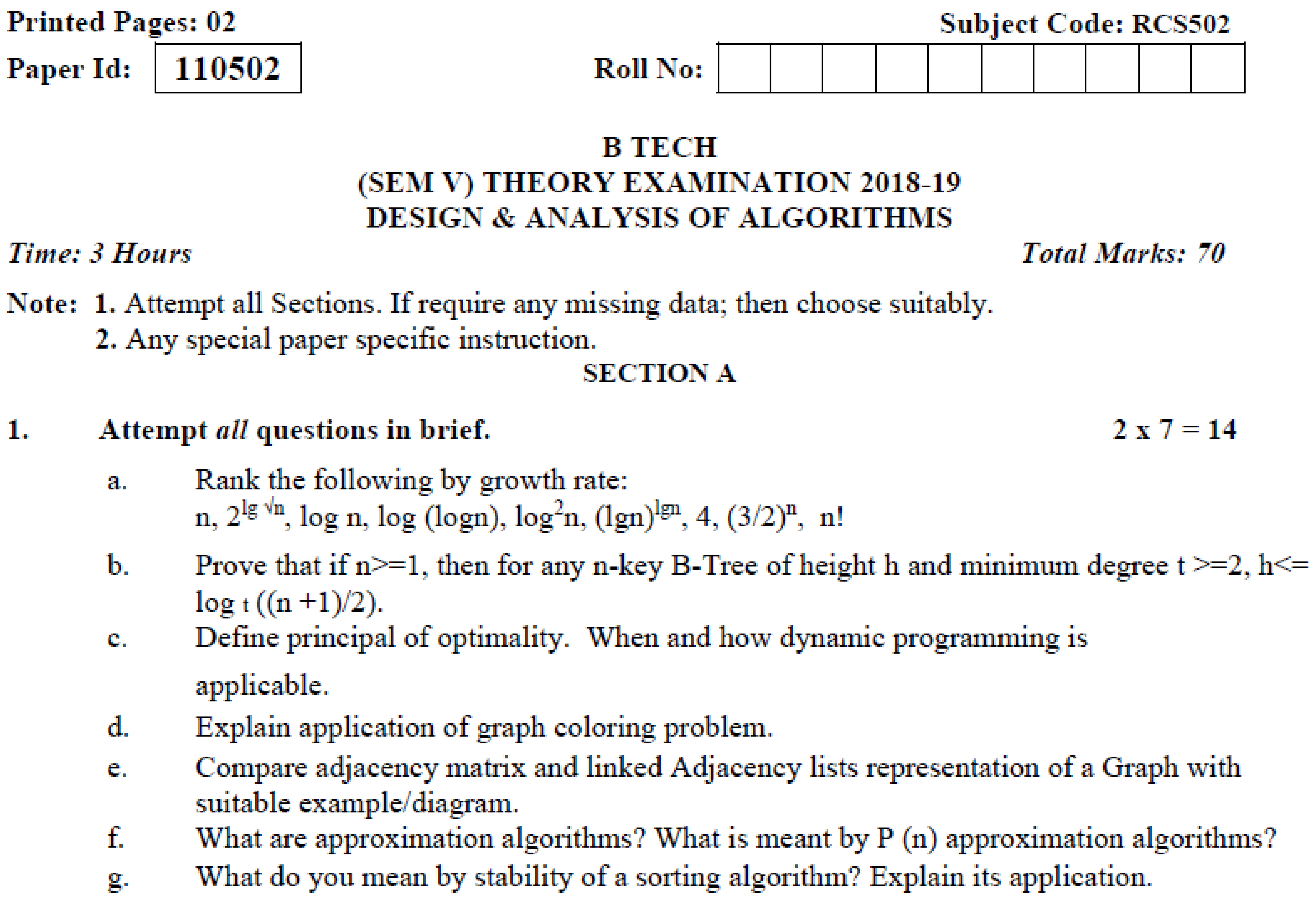


7. Attempt any *one* part of the following: 7 x 1 = 7

(a) Describe in detail Knuth-Morris-Pratt string matching algorithm. Compute the prefix function π for the pattern ababbabbabbababbabb when the alphabet is $\Sigma = \{a,b\}$.

(b) What is an approximation algorithm? What is meant by P (n) approximation algorithms? Discuss approximation algorithm for Travelling Salesman Problem.

Old Question Papers(2018-2019)



Printed Pages: 02 **Subject Code: RCS502**

Paper Id: 110502 **Roll No:**

B TECH
(SEM V) THEORY EXAMINATION 2018-19
DESIGN & ANALYSIS OF ALGORITHMS

Time: 3 Hours ***Total Marks: 70***

Note: **1.** Attempt all Sections. If require any missing data; then choose suitably.
2. Any special paper specific instruction.

SECTION A

1. **Attempt *all* questions in brief.** **2 x 7 = 14**

a. Rank the following by growth rate:
n, $2^{\lg \sqrt{n}}$, $\log n$, $\log(\log n)$, $\log^2 n$, $(\lg n)^{\lg n}$, 4, $(3/2)^n$, $n!$

b. Prove that if n>=1, then for any n-key B-Tree of height h and minimum degree t >=2, h<= $\log_t((n+1)/2)$.

c. Define principal of optimality. When and how dynamic programming is applicable.

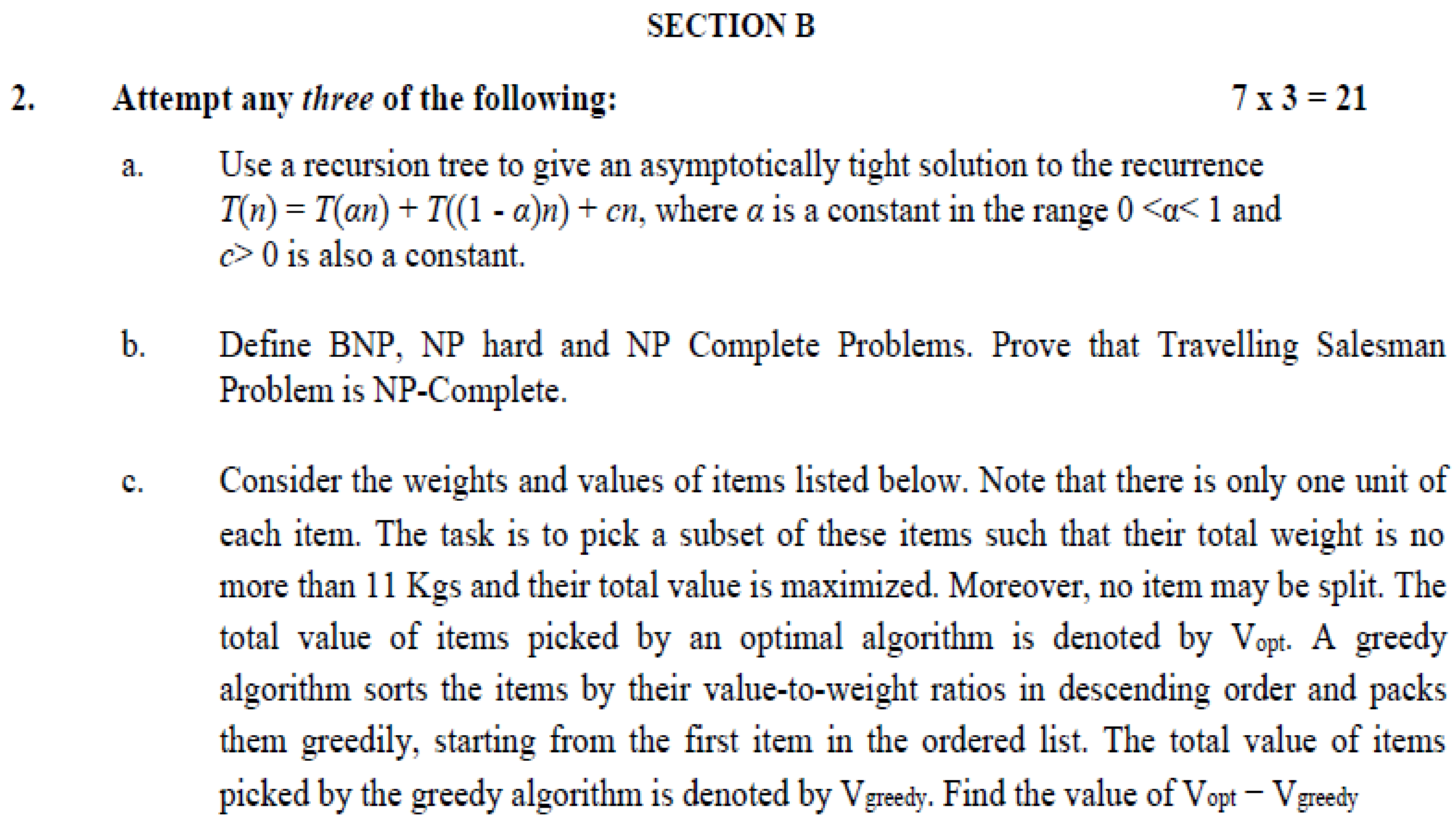
d. Explain application of graph coloring problem.

e. Compare adjacency matrix and linked Adjacency lists representation of a Graph with suitable example/diagram.

f. What are approximation algorithms? What is meant by P (n) approximation algorithms?

g. What do you mean by stability of a sorting algorithm? Explain its application.

Old Question Papers(2018-2019)



SECTION B

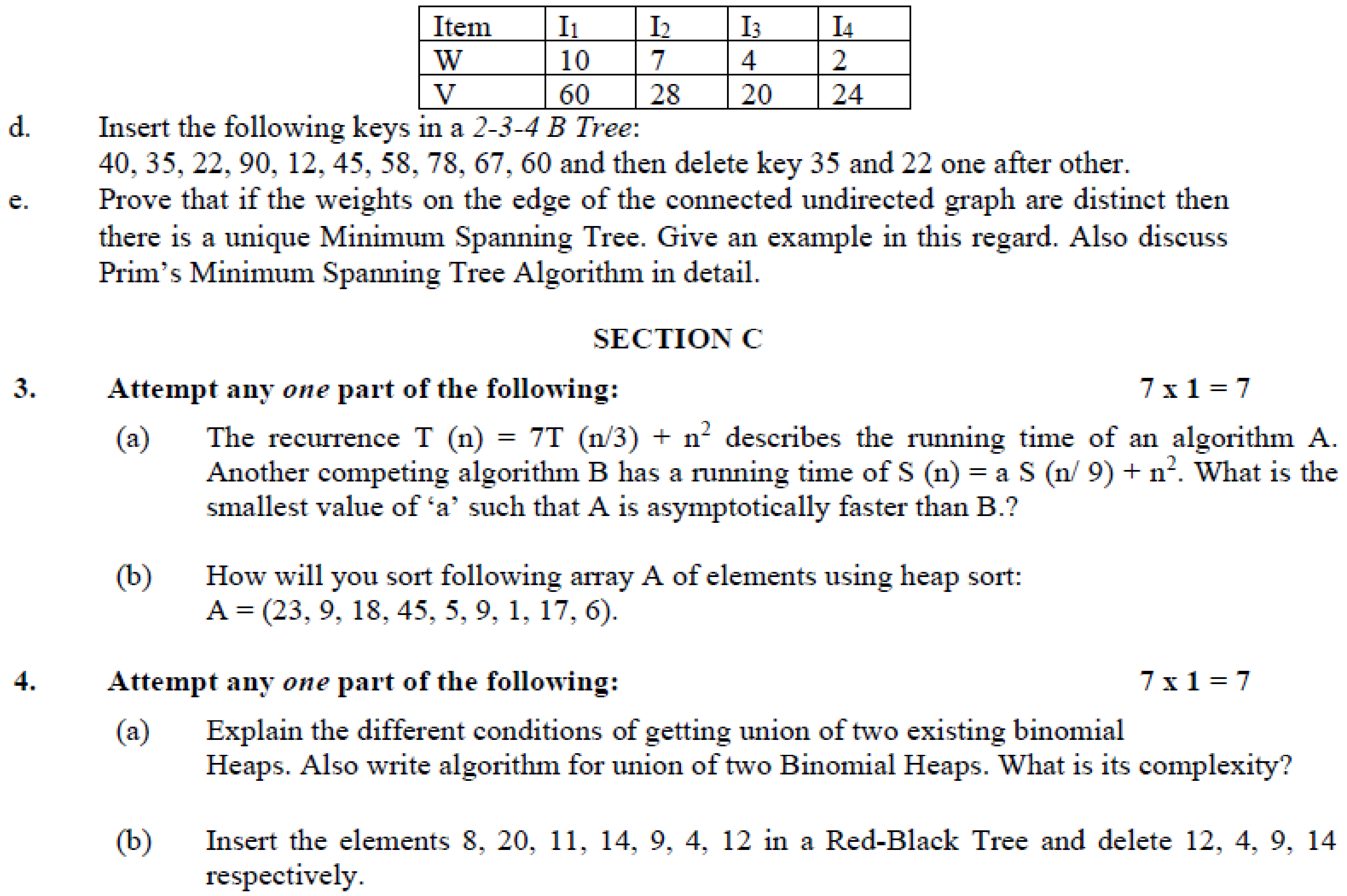
2. Attempt any *three* of the following: 7 x 3 = 21

a. Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$, where α is a constant in the range $0 <\alpha< 1$ and $c> 0$ is also a constant.

b. Define BNP, NP hard and NP Complete Problems. Prove that Travelling Salesman Problem is NP-Complete.

c. Consider the weights and values of items listed below. Note that there is only one unit of each item. The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}. Find the value of $V_{opt} - V_{greedy}$

Old Question Papers(2018-2019)



Item	I_1	I_2	I_3	I_4
W	10	7	4	2
V	60	28	20	24

d. Insert the following keys in a *2-3-4 B Tree*:
40, 35, 22, 90, 12, 45, 58, 78, 67, 60 and then delete key 35 and 22 one after other.

e. Prove that if the weights on the edge of the connected undirected graph are distinct then there is a unique Minimum Spanning Tree. Give an example in this regard. Also discuss Prim's Minimum Spanning Tree Algorithm in detail.

SECTION C

3. Attempt any *one* part of the following: 7 x 1 = 7

(a) The recurrence T (n) = 7T (n/3) + n^2 describes the running time of an algorithm A. Another competing algorithm B has a running time of S (n) = a S (n/ 9) + n^2. What is the smallest value of 'a' such that A is asymptotically faster than B.?

(b) How will you sort following array A of elements using heap sort:
A = (23, 9, 18, 45, 5, 9, 1, 17, 6).

4. Attempt any *one* part of the following: 7 x 1 = 7

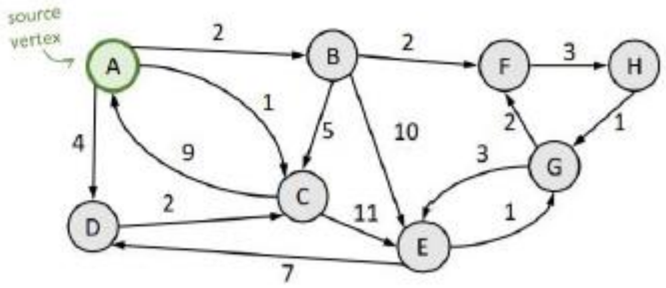
(a) Explain the different conditions of getting union of two existing binomial Heaps. Also write algorithm for union of two Binomial Heaps. What is its complexity?

(b) Insert the elements 8, 20, 11, 14, 9, 4, 12 in a Red-Black Tree and delete 12, 4, 9, 14 respectively.

Old Question Papers(2018-2019)

5. **Attempt any *one* part of the following:** **7 x 1 = 7**

(a) When do Dijkstra and the Bellman-Ford algorithm both fail to find a shortest path?Can Bellman ford detect all negative weight cycles in a graph?Apply Bellman Ford Algorithm on the following graph:



(b) Given an integer x and a positive number n, use divide & conquer approach to write a function that computes $x^n$with time complexity O (logn).

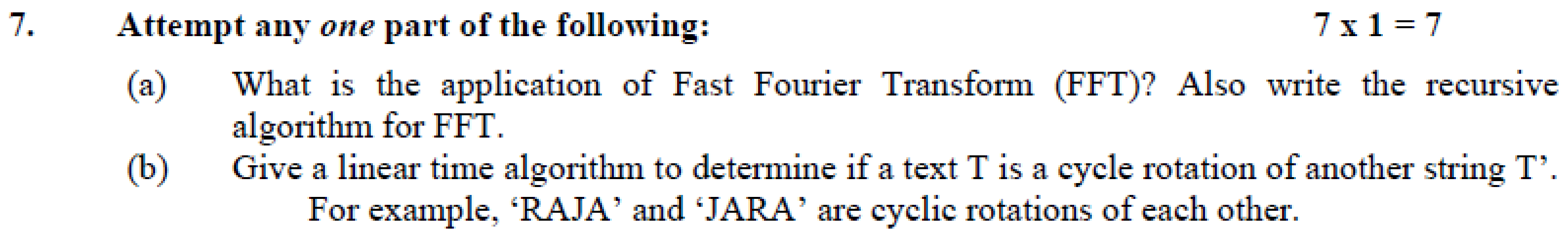
6. **Attempt any *one* part of the following:** **7 x 1 = 7**

(a) Solve the Subset sum problem using Backtracking, where n=4, m=18, w [4] = {5, 10, 8, 13}

(b) Give Floyd War shall algorithm to find the shortest path for all pairs of vertices in a graph. Give the complexity of the algorithm. Explain with example.

NIET Greater Noida
GET FUTURE READY

Old Question Papers(2018-2019)

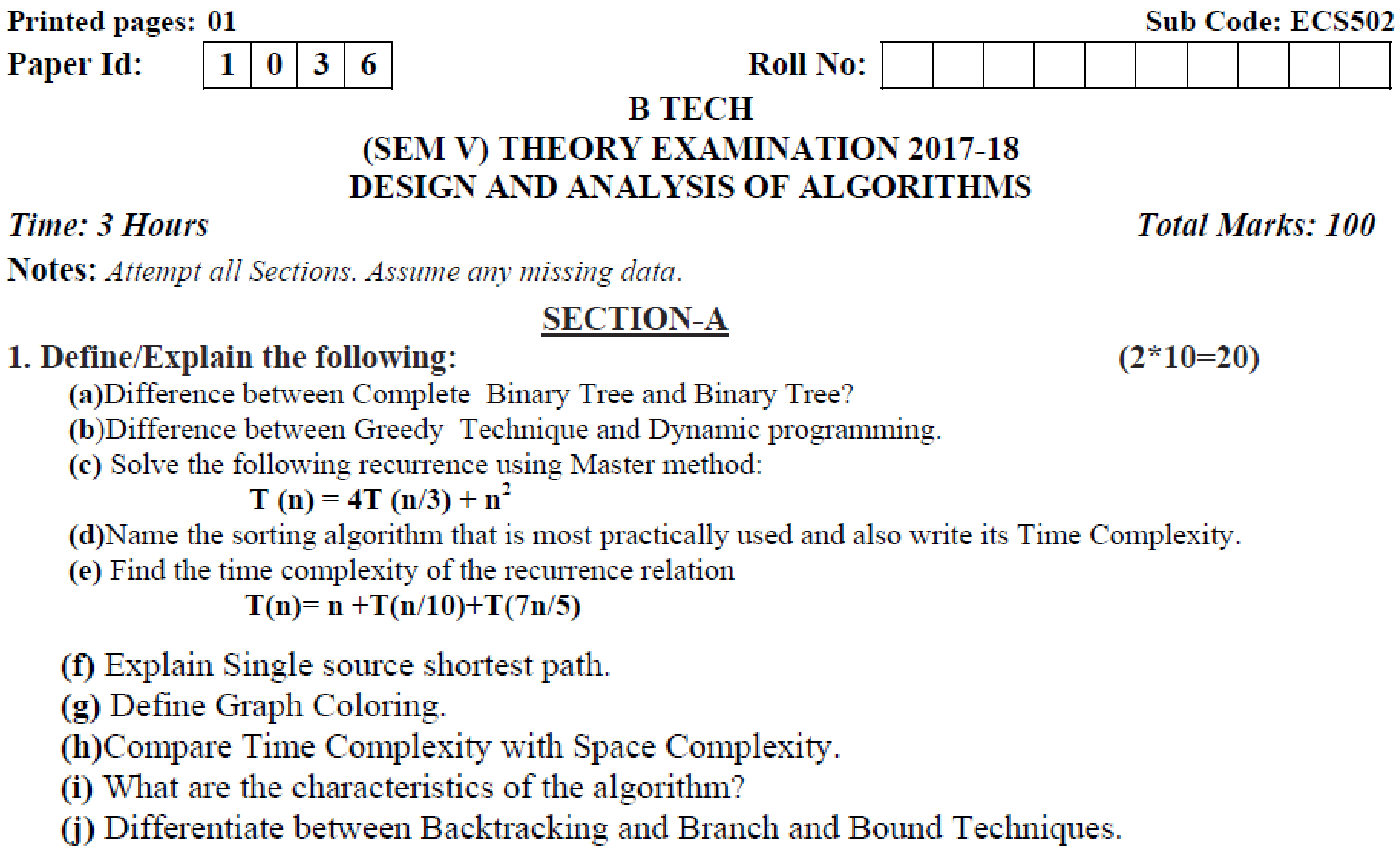


7. Attempt any *one* part of the following: 7 x 1 = 7

(a) What is the application of Fast Fourier Transform (FFT)? Also write the recursive algorithm for FFT.

(b) Give a linear time algorithm to determine if a text T is a cycle rotation of another string T'. For example, 'RAJA' and 'JARA' are cyclic rotations of each other.

Old Question Papers(2017-2018)



Printed pages: 01 **Sub Code: ECS502**

Paper Id: | 1 | 0 | 3 | 6 | **Roll No:**

B TECH

(SEM V) THEORY EXAMINATION 2017-18

DESIGN AND ANALYSIS OF ALGORITHMS

Time: 3 Hours ***Total Marks: 100***

Notes: *Attempt all Sections. Assume any missing data.*

SECTION-A

1. Define/Explain the following: **(2*10=20)**

(a) Difference between Complete Binary Tree and Binary Tree?

(b) Difference between Greedy Technique and Dynamic programming.

(c) Solve the following recurrence using Master method:

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

(d) Name the sorting algorithm that is most practically used and also write its Time Complexity.

(e) Find the time complexity of the recurrence relation

$$T(n)= n + T(n/10)+T(7n/5)$$

(f) Explain Single source shortest path.

(g) Define Graph Coloring.

(h) Compare Time Complexity with Space Complexity.

(i) What are the characteristics of the algorithm?

(j) Differentiate between Backtracking and Branch and Bound Techniques.

Old Question Papers(2017-2018)

SECTION-B

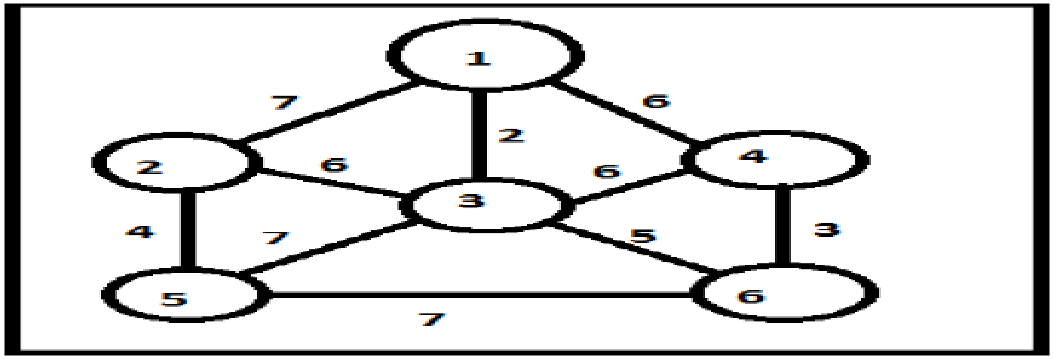
2. Attempt any three of the following: **(10×3=30)**

(a) Solve the following By Recursion Tree Method

T(n)=n+T(n/5)+T(4n/5)

(b) Insert the following information **F,S,Q,K,C,L,H,T,V,W,M,R,N,P,A,B,X,Y,D,Z,E,G,I.** Into an empty B-tree with degree t=3.

(c) What is Minimum Cost Spanning Tree? Explain Kruskal's Algorithm and Find MST of the Graph. Also write its Time-Complexity



(d) What is Red-Black tree? Write an algorithm to insert a node in an empty red-black tree explain with suitable example.

(e) Explain HEAP-SORT on the array. Illustrate the operation of HEAP-SORT on the array

A= {6, 14, 3, 25, 2, 10, 20, 7, 6}

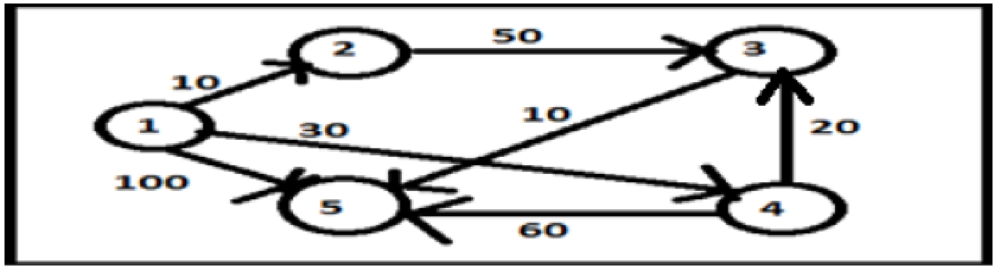
Old Question Papers(2017-2018)

SECTION C

3. Attempt any one part of the following: (10 x 1=10)

(a) Explain Convex –Hull problem.

(b) Find the shortest path in the below graph from the source vertex 1 to all other vertices by using Dijkstra's algorithm.



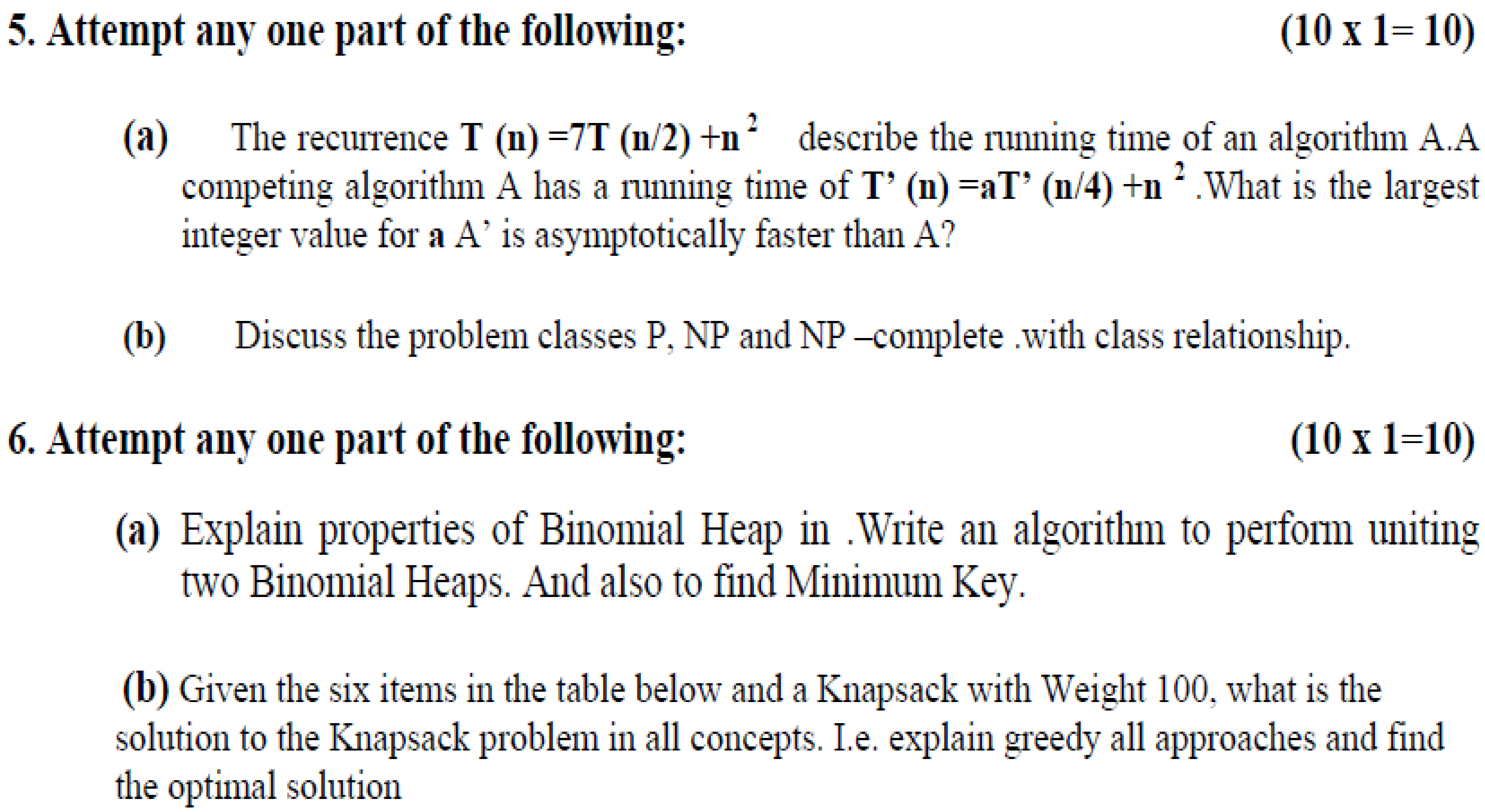
4. Attempt any one part of the following: (10 x 1=10)

(a) What is backtracking? Discuss sum of subset problem with the help of an example.

(b) Write down an algorithm to compute Longest Common Subsequence (LCS) of two given strings and analyze its time complexity.

NIET Greater Noida GET FUTURE READY

Old Question Papers(2017-2018)



5. Attempt any one part of the following: **(10 x 1= 10)**

(a) The recurrence **T (n) =7T (n/2) +n ^{2}** describe the running time of an algorithm A.A competing algorithm A has a running time of **T' (n) =aT' (n/4) +n ^{2}** .What is the largest integer value for **a** A' is asymptotically faster than A?

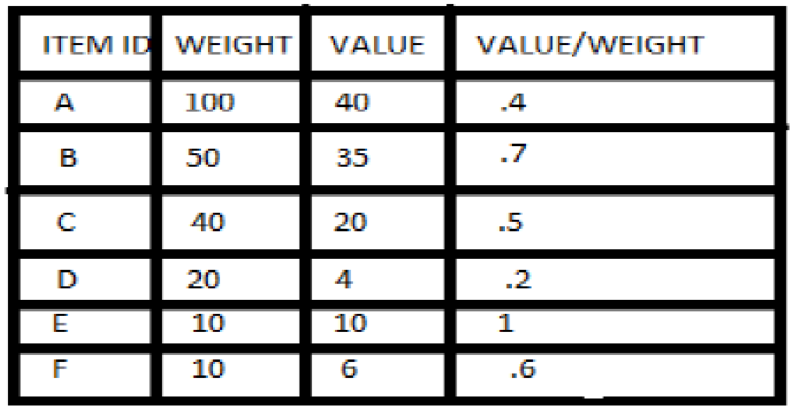
(b) Discuss the problem classes P, NP and NP –complete .with class relationship.

6. Attempt any one part of the following: **(10 x 1=10)**

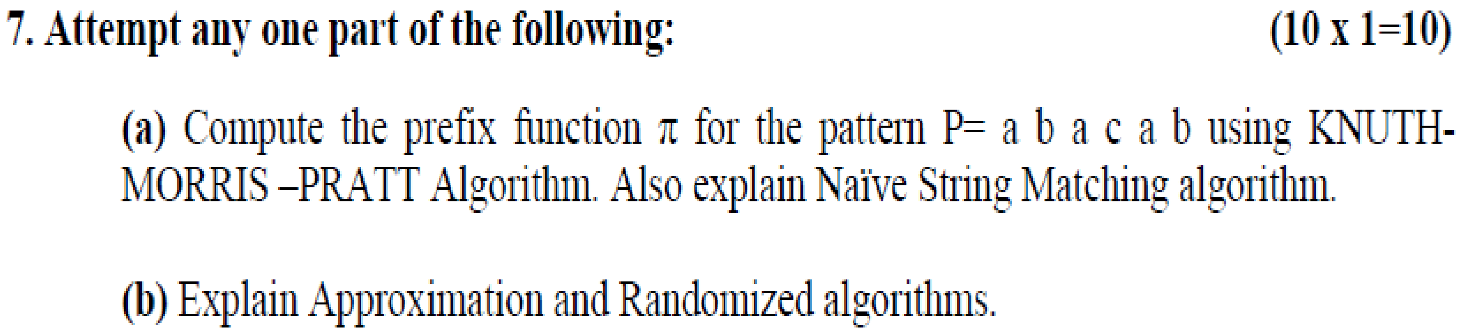
(a) Explain properties of Binomial Heap in .Write an algorithm to perform uniting two Binomial Heaps. And also to find Minimum Key.

(b) Given the six items in the table below and a Knapsack with Weight 100, what is the solution to the Knapsack problem in all concepts. I.e. explain greedy all approaches and find the optimal solution

Old Question Papers(2017-2018)



ITEM ID	WEIGHT	VALUE	VALUE/WEIGHT
A	100	40	.4
B	50	35	.7
C	40	20	.5
D	20	4	.2
E	10	10	1
F	10	6	.6



7. Attempt any one part of the following: **(10 x 1=10)**

(a) Compute the prefix function π for the pattern P= a b a c a b using KNUTH-MORRIS –PRATT Algorithm. Also explain Naïve String Matching algorithm.

(b) Explain Approximation and Randomized algorithms.

Expected Questions for University Exam

Q1 Differentiate between asymptotic notation O and Ω. [CO1]

Q2 Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant. [CO1]

Q3 Find the time complexity of the recurrence relation

$T(n) = n + T(n/10) + T(7n/5)$ [CO1]

Q4 Compare Time Complexity with Space Complexity. [CO1]

Q5 What are the characteristics of the algorithm? [CO1]

Q6 Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \lg n)$.[CO1]

Q7 Solve the recurrence: $T(n) = 50\,T(n/49) + \log n!$ [CO1]

Q8 Solve the recurrence using recursion tree method: [CO1]

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n$$

Expected Questions for University Exam

Q9. Write an algorithm to sort the given array of dement using Quick-sort. Illustrate the operation of PARTITION procedure on the array = < 2, 8, 7, 1, 3, 5, 6, 4 > . [CO1]

Q10. Apply BUCKET SORT algorithm on the following array 0 · 78, 0 · 17, 0 · 39, 0 · 26, 0 · 72, 0 · 94, 0 · 21, 0 · 21, 0 · 12, 0 · 23, 0 · 68 [CO1]

Q11. Why Counting sort is called stable sort. [CO1]

Q12. Distinguish between Quick sort and Merge sort, and arrange the following numbers in increasing order using merge sort 18, 29, 68, 32, 43, 37, 87, 24, 47, 50. [CO1]

Q13. What is divide and conquer strategy and explain the binary search with suitable example. [CO1]

Recap Of Unit

This is an introductory chapter of Design & Analysis of Algorithm covering the concept, importance and characteristics of algorithms. The complexity and its calculation has been explained. Further, recursion and different methodologies to solve them have also been provided.
In other part of unit different sorting algorithm – Insertion, Heap sort, Quick sort, bucket sort, Radix Sort and Counting sort have been discussed along with their complexities.

.

Thank You