Algorithm Design and Analysis

First Edition [January, 2021]
Second Edition [March, 2024]
Third Edition [July, 2024]

By

MD. Aftab Uddin Alif
M.Sc. in ICT, BUP (Ongoing)
B.Sc. in CSE, BAIUST
Lecturer, Department of CSE
CCN University of Science and Technology
Former Assistant Engineer
IDS Tech Solution (July, 2023 – January, 2024)

This book contains information obtained from the class lectures of **Algorithm Design and Analysis** course taken under the **B.Sc. in Computer Science and Engineering (CSE)** program conducted **by CCN University of Science and Technology**.

Contents

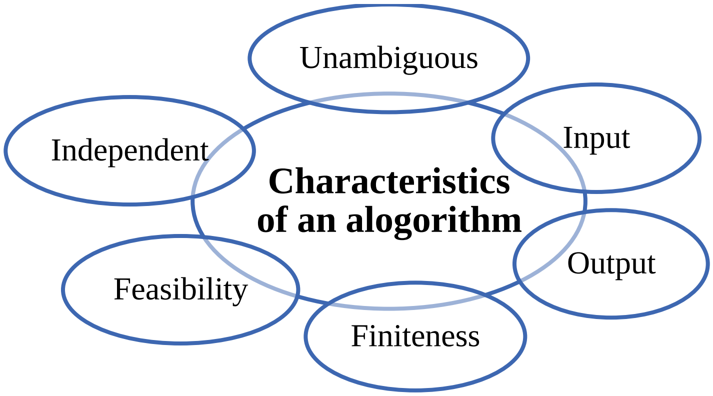
Chapter-1
Algorithm Basic

❖ What is Algorithms?

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

Design
Experiment
Algorithms
Analyze
Implement

❖ Characteristics of an algorithm



- **Unambiguous:** An algorithm should be clear and not open to interpretation. Each step or phase of the algorithm, as well as its inputs and outputs, should be explicitly defined. This ensures that there is no confusion or ambiguity in understanding the algorithm's logic.
- **Input:** An algorithm should have well-defined inputs. These inputs are the data or information that the algorithm processes to produce the desired output. Having clearly defined inputs is essential for the algorithm to operate predictably and consistently.

- **Output:** Similarly, an algorithm should have one or more well-defined outputs. The outputs represent the results or outcomes generated by the algorithm based on the provided inputs. The outputs should align with the expected or desired outcome.
- **Finiteness:** Algorithms must terminate after a finite number of steps. This ensures that the algorithm doesn't run indefinitely, leading to an infinite loop or an unresolved process. Finiteness is crucial for the algorithm to be practical and for it to produce results within a reasonable timeframe.
- **Feasibility:** The algorithm should be feasible with the available resources. This means that the algorithm should be implementable using the resources, such as computing power and memory, that are available.
- **Independent:** An algorithm should provide step-by-step directions that are independent of any specific programming code. This characteristic emphasizes that the logic of the algorithm should be described in a way that is not tied to a particular programming language.

❖ How to design an algorithm?

1. Obtain a description of the problem.
2. Analyze the problem.
3. Develop a high-level algorithm.
4. Refine the algorithm by adding more detail.
5. Review the algorithm.

❖ Algorithm Complexity

- **Space complexity:** Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle.
- **Time complexity:** Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion.

❖ Asymptotic Notations

Asymptotic 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.

There are mainly three asymptotic notations:

- **O 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.
- ***θ* 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.
- **Ω 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.

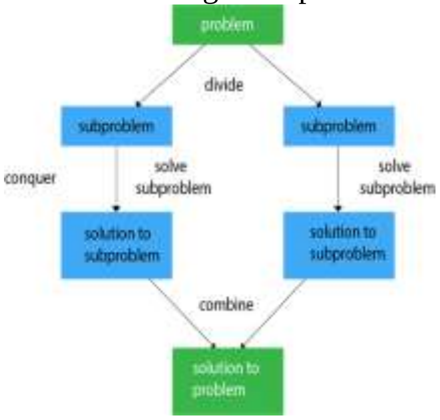
O	**Ω**	***θ***
Big Oh	**Omega**	**Theta**
Upper Bound	Lower Bound	Upper & Lower Bound
Worst Case	Best Case	Average 'ish'

❖ Divide and Conquer

Divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.

This technique can be divided into the following three parts:

1. Divide: This involves dividing the problem into smaller sub-problems.
2. Conquer: Solve sub-problems by calling recursively until solved.
3. Combine: Combine the sub-problems to get the final solution of the whole problem.



❖ Exercise

1.1. What is algorithm and write down the characteristics of an algorithm.

1.2. Explain the types of algorithm complexity.

1.3. Explain asymptotic notations.

1.4. State the “Divide and Conquer” theorem.

Chapter-2
Searching Algorithm

❖ Linear Search

- Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

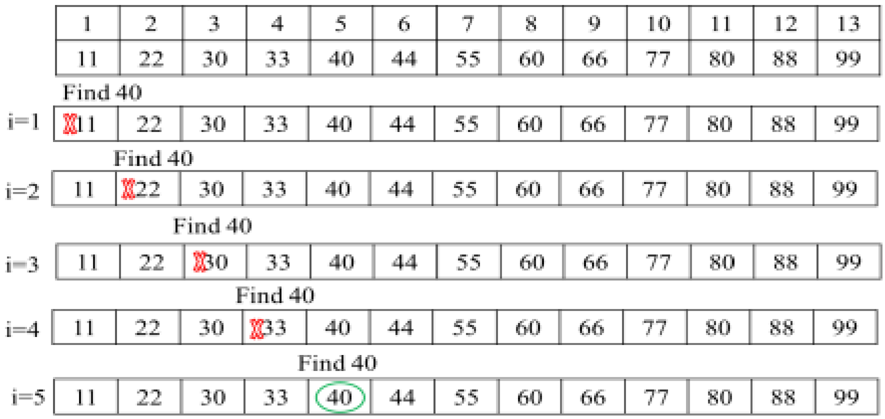
Rule: In the case of linear search, the values that will be given first, starting from the first value to the last value, should be compared one by one with the value that is asked to be found. In this way the value can be found at one stage.

নিয়মঃ Linear search এর ক্ষেত্রে প্রথমে যে value গুলো দেওয়া থাকবে তার প্রথম Value থেকে শুরু করে শেষ অবধি যে Value টা থাকবে তার সাথে যে Value টা খুজে বের করতে বলছে তাকে মিলিয়ে দেখতে হবে একটা একটা করে। এইভাবে এক পর্যায়ে Value টা খুজে পাওয়া যাবে।

❖ Example 2.1

Let DATA be the following sorted 13 element array DATA=11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99 now find 40 using Linear search.

Solution:



	1	2	3	4	5	6	7	8	9	10	11	12	13
	11	22	30	33	40	44	55	60	66	77	80	88	99

Find 40

i=1	✗11	22	30	33	40	44	55	60	66	77	80	88	99

Find 40

i=2	11	✗22	30	33	40	44	55	60	66	77	80	88	99

Find 40

i=3	11	22	✗30	33	40	44	55	60	66	77	80	88	99

Find 40

i=4	11	22	30	✗33	40	44	55	60	66	77	80	88	99

Find 40

i=5	11	22	30	33	◯40	44	55	60	66	77	80	88	99

After searching the value is find in Location 5.

❖ Linear Search Algorithm

```
LINEAR (DATA, N, SEARCH, LOC)
Here DATA is N elements, and we find SEARCH
1. [Initialize] Set Temp: =1 and LOC: =0
2. Repeat steps 3 and 4 while LOC:=0 and K≤N
3. If SEARCH=DATA[Temp], Then: set LOC: =K
4. Set Temp: =Temp+1 [Increment temporary value]
        [End of step 2 loop]
5. [Successful]
        If LOC: =0, then:
                SEARCH is not in the array DATA
        Else
                LOC is the Location of SEARCH
        [End of the structure]
6. Exit
```

❖ Linear Search Big-O notation

Suppose P_k is the probability that SEARCH appears in DATA[k] and q is the probability that SEARCH does not appear in DATA

$$f(n) = 1.P_1 + 2.P_2 + 3.P_3 + ----------+n.P_n + (n+1).q$$

Then q=0 and $P_1 = \frac{1}{n}$ accordingly,

$$f(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + 3.\frac{1}{n} + --------- + n.\frac{1}{n} + (n+1).0$$

$$= (1 + 2 + ----+).\frac{1}{n}$$

$$= \frac{n(n+1)}{2}.\frac{1}{n} = \frac{n+1}{2} = \frac{n}{2} + \frac{1}{2}$$

Remove all constant then n

∴Big-O of Linear search is O(n)

❖ Binary Search

- Binary search is an efficient algorithm to find a target in a sorted array. It repeatedly divides the search range in half, comparing the middle element with the target and narrowing down the search until the target is found or the search range is empty.

Rule: In the case of binary search, the sum of BEG and END of the values given first should be divided by 2 to see the value of that location. If it is greater than the value that is being sought, then the value of BEG is added to MID by 1. If it is small, then the value of END should be reduced by subtracting 1 from MID. In this way the value can be found at one stage.

নিয়মঃ Binary search এর ক্ষেত্রে প্রথমে যে value গুলো দেওয়া থাকবে তার BEG ও END এর যোগফলকে ২ দিয়ে ভাগ করে ঐ Location এর Value এর মান দেখতে হবে যদি সেটা যে Value টা খুজে বের করতে বলছে তার থেকে বড় হয় তাহলে BEG এর মান MID এর সাথে ১ যোগ করে বাড়াতে হবে এর যদি ছোট হয় তাহলে END এর মান MID থেকে ১ বিয়োগ করে কমাতে হবে। এইভাবে এক পর্যায়ে Value টা খুজে পাওয়া যাবে।

❖ Example 2.2

Let DATA be the following sorted 13 element array DATA=11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99 now find 40 using binary search.

Solution:

1	2	3	4	5	6	7	8	9	10	11	12	13
11	22	30	33	40	44	55	60	66	77	80	88	99

1. Initially BEG=1 and END=13, Hence MID= (1+13)/2=7 and DATA[MID]=55
2. Since 40<55 so END=MID-1=7-1=6 and BEG=1, Hence MID= (1+6)/2=4 and DATA[MID]=33
3. Since 40>33 so BEG=MID+1=4+1=5 and END=6, Hence MID= (5+6)/2=6 and DATA[MID]=44
4. Since 40<44 so END=MID-1=6-1=5 and BEG=5, Hence MID= (5+5)/2=5 and DATA[MID]=40
5. 40=40 so we have found 40 in Location 5
6. **Successful**

❖ Example 2.3

Let DATA be the following sorted 13 element array DATA=11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99 now find 41 using binary search.

Solution:

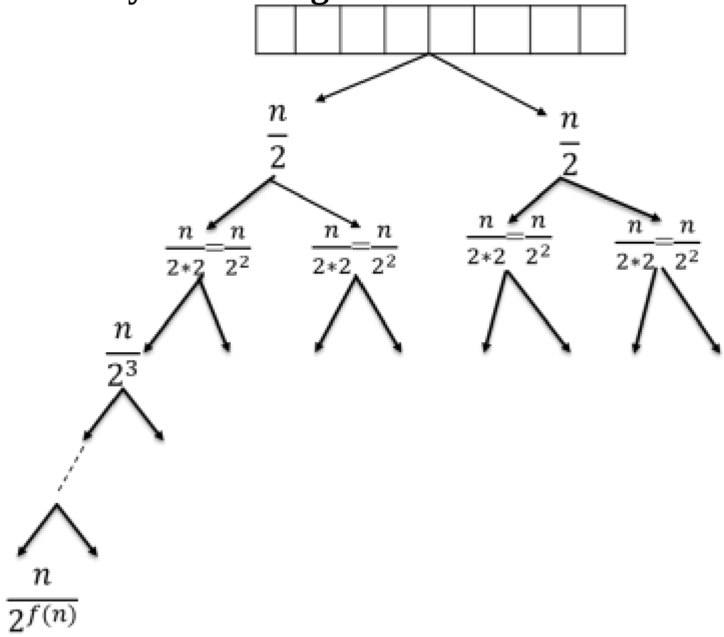
1	2	3	4	5	6	7	8	9	10	11	12	13
11	22	30	33	40	44	55	60	66	77	80	88	99

1. Initially BEG=1 and END=13, Hence MID= (1+13)/2=7 and DATA[MID]=55
2. Since 41<55 so END=MID-1=7-1=6 and BEG=1, Hence MID= (1+6)/2=4 and DATA[MID]=33
3. Since 41>33 so BEG=MID+1=4+1=5 and END=6, Hence MID= (5+6)/2=6 and DATA[MID]=44
4. Since 41<44 so END=MID-1=6-1=5 and BEG=5, Hence MID= (5+5)/2=5 and DATA[MID]=40
5. Since 41>40 so BEG=MID+1=5+1=6 and END=5, Now BEG>END its wrong condition so we don't find 41 in this array
6. **Unsuccessful**

❖ Binary Search Algorithm

```
BINARY (DATA, LB, UB, SEARCH, LOC).
1. [Initialize segment variables] Set START: =LB, END: =UB and
MID=INT(START+END)/2
2. Repeats steps 3 and 4 while START≤END and
DATA[MID]≠SEARCH
3. If SEARCH<DATA[MID] then: Set END: =MID-1
   Else Set START: =MID+1
   [End of if structure]
4. Set MID: =INT((START+END)/2)
   [End of step 2 loop]
5. If DATA[MID]=Search, then: Set LOC: =MID
   Else Set LOC: =NULL
   [End of if structure]
6. Exit
```

❖ Binary Search Big-O notation



$\frac{n}{2}$
$\frac{n}{2}$
$\frac{n}{2*2}=\frac{n}{2^2}$
$\frac{n}{2*2}=\frac{n}{2^2}$
$\frac{n}{2*2}=\frac{n}{2^2}$
$\frac{n}{2*2}=\frac{n}{2^2}$
$\frac{n}{2^3}$
$\frac{n}{2^{f(n)}}$

Here we find,

$$\frac{n}{2^{f(n)}}$$

Now,

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

∴Big-O of Binary search is O($logn$)

❖ Exercise

2.1. Write down the short notes of Linear Search and find out the Big-O notation of Linear Search.

2.2. State the Linear Search algorithm.

2.3. Let DATA be the following sorted 10 element array DATA=14, 29, 13, 23, 60, 41, 75, 59, 77, 81 now find 75 using Linear search.

2.4. Let DATA be the following sorted 10 element array DATA=14, 29, 13, 23, 60, 41, 75, 59, 77, 81 now find 83 using Linear search.

2.5. Write down the short notes of Binary Search and find out the Big-O notation of Binary Search.

2.6. State the Binary Search algorithm.

2.7. Let DATA be the following sorted 12 element array DATA=11, 25, 32, 36, 43, 49, 55, 62, 69, 75, 83, 88 now find 55 using binary search.

2.8. Let DATA be the following sorted 12 element array DATA=11, 25, 32, 36, 43, 49, 55, 62, 69, 75, 83, 88 now find 85 using binary search.

Chapter-3
Sorting Algorithm

❖ Merge Sort

- Merge sort is one the most efficient sorting techniques and it's based on the "divide and conquer" paradigm.
  - In marge sort the problem is divided into two sub problems in every iteration.
  - Hence efficiency is increased drastically.
- It follows the "divide and conquer" drastically-
  - Divide break the problem into two sub problem which continues until the problem set is left with one element only.
  - Conquer basically merges the two sorted arrays into the original array.

Rule: In the case of Merge sort, the given array should be divided into two every time and at which point to divide it should be found out with the formula MID=(LOW+HIGH)/2. By dividing in this way, when the value becomes 1, then they should be arranged from the smallest to the largest and make an array as before.

নিয়মঃ Merge sort এর ক্ষেত্রে যে array দেওয়া থাকবে তাকে প্রতিবার দুই ভাগ করে লিখতে হবে এবং কোন point এ ভাগ করব তা MID=(LOW+HIGH)/2 এই সুত্র দিয়ে বের করতে হবে। এভাবে ভাগ করে যখন ১টা করে value হয়ে যাবে তখন তাদের ছোটো থেকে বড় সাজিয়ে পূর্বের মতো একটা array করে ফেলতে হবে।

❖ How to find MID value?

LB UB

0	1	2	3	4	5	6	7
38	27	43	6	9	82	10	5

MID = (Lower bound + Higher bound) / 2

Here,

MID = floor {(0+7)/2}

MID=floor (7/2)

MID=floor (3.5)

MID=4th value which is 6, that means array [3] position

❖ Example 3.1

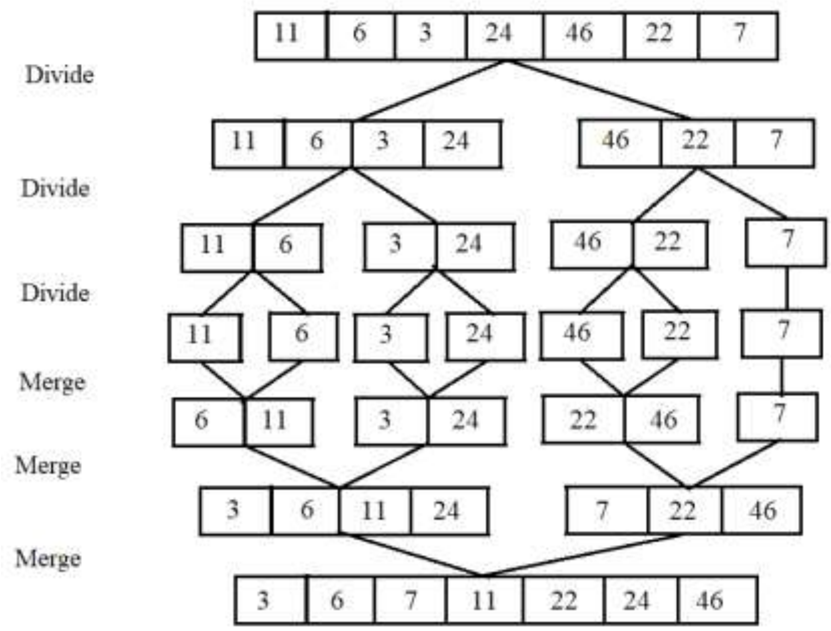
Suppose (11, 6, 3, 24, 46, 22, 7) these are the value of an array. Now sort the array in ascending order using Merge Sort.

Solution:

0	1	2	3	4	5	6
11	6	3	24	46	22	7

Here,

MID = floor {(0+6)/2}
MID=floor (6/2)
MID=floor (3)
MID=4th value which is 24, that means array [3] position



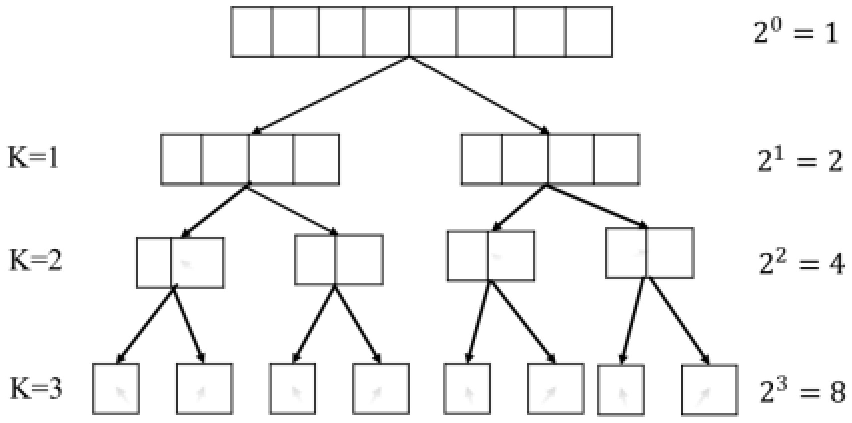
After sorting the array is (3, 6, 7, 11, 22, 24, 46)

❖ Merge Sort Algorithm

```
Merge_SORT(low,high)
1.if low<high, then;
        MID=(low+high)/2
        Merge_SORT(low,MID)
        Merge_SORT(MID+1, high)
        Merge (low,MID,high)
[End of if structure]

Merge (low,MID,high)
1.Set i=low,j=MID+1,k=low
2.Repeat while i<=MID and j<=high
        if a[i]<=a[j], then:
                set: b[k]=a[i]
        else
                set: b[k]=a[j]
        [End of if structure]
[End of loop]
3.if i=MID+1
        Repeat while j<=high
                set: b[k]=a[j]
        [End of loop]
[End of if structure]
4.if j>high
        Repeat while j>high
                set: b[k]=a[i]
        [End of loop]
[End of if structure]
5.Repeat for i=low to high
        set: a[i]=b[i]
[End of loop]
```

❖ Merge Sort Big-O notation



Here,

$when\ K = 0\ then\ 2^0 = 1$
$when\ K = 1\ then\ 2^1 = 2$
$when\ K = 2\ then\ 2^2 = 4$
$when\ K = 3\ then\ 2^3 = 8$
That means, 2^K

In merge sort $K = logn$

For K level,

$= K * 2^k$
$= logn * 2^{logn}$
$= \text{logn} * \text{n}\ \ [2^{logn} = n]$

∴Big-O of merge sort is O($nlogn$)

❖ Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.
- In Bubble Sort algorithm, traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so on until the data is sorted.

Rule: In the case of bubble sort, all the values from the 1st to the last of the given array should be compared two by two, if the 1st value is smaller than the 2nd value then they should be swapped. In this case, the pass number will be Number of value -1. In this case, the iteration in each iteration will be Number of value – pass number. After all the passes, the array will be sorted.

নিয়মঃ Bubble sort এর ক্ষেত্রে যে array দেওয়া থাকবে তার ১ম থেকে শেষ পর্যন্ত যতগুলো value থাকবে সব গুলো value কে দুইটা করে compare করে দেখতে হবে, যদি ১ম value থেকে ২য় value ছোট হয় তাহলে তাদেরকে swap করতে হবে। এক্ষেত্রে pass সংখ্যা হবে Number of value –1. এক্ষেত্রে প্রত্যেক pass এ iteration হবে Number of value – pass number. সবগুলো pass শেষ হলে array টা সাজানো হয়ে যাবে।

❖ How to find number of pass & iteration?

0	1	2	3	4	5	6	7
38	27	43	6	9	82	10	5

Here,

Total number = 8

So, number of pass = 8-1 = 7

In 1st iteration, number of iteration = 8-1 = 7

In 2nd iteration, number of iteration = 8-2 = 6

.

.

In 7th iteration, number of iteration = 8-7 = 1

❖ Example 3.2

Suppose (23, 11, 5, 32, 19) these are the value of an array. Now sort the array in ascending order using Bubble Sort.

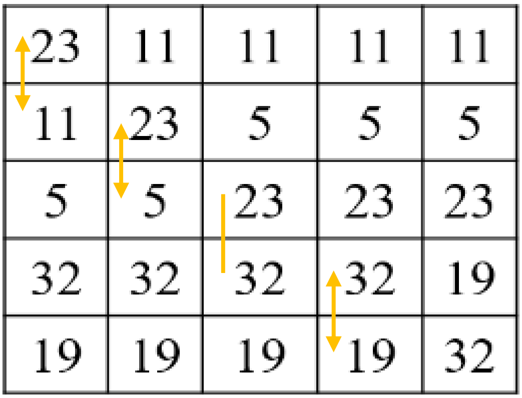
Solution:

Here,

Total number = 5

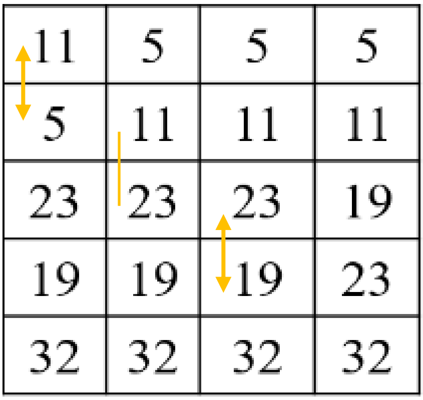
So, number of pass = 5-1 = 4

Pass-1



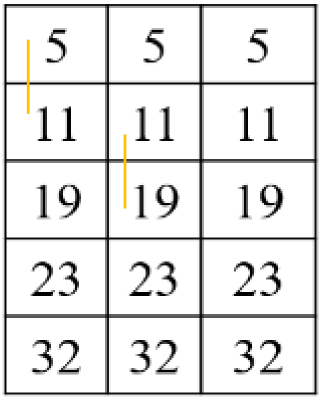
23	11	11	11	11
11	23	5	5	5
5	5	23	23	23
32	32	32	32	19
19	19	19	19	32

Pass-2



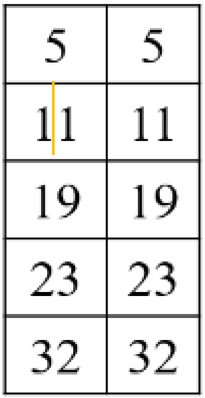
11	5	5	5
5	11	11	11
23	23	23	19
19	19	19	23
32	32	32	32

Pass-3



5	5	5
11	11	11
19	19	19
23	23	23
32	32	32

Pass-4



5	5
11	11
19	19
23	23
32	32

After sorting the array is (5, 11, 19, 23, 32)

❖ Bubble Sort Algorithm

BUBBLE (DATA, N)
Here DATA is an array with N elements.
This algorithm sorts the elements in DATA

```
1. Repeat steps 2 and 3 for k=1 to N – 1
2. Set PTR: =1 [Initializes pass pointer PTR]
3. Repeat while PTR $\le$ N $-$ k [Executes pass]
   a. If DATA[PTR] $>$ DATA[PTR $+$ 1] Then;
      Interchange DATA[PTR] and DATA[PTR+1]
      [End of if structure]
   b. Set PTR: = PTR+1
      [End of inner loop]
   [End of Step 1 outer loop]
4. Exit
```

❖ Bubble Sort Big-O notation

The number $f(n)$ of comparisons in the bubble sort is easily completed.
There are $n-1$ comparisons during the first pass.

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

$$= \frac{(n-1)\times(n-1-1)}{2}$$

$$= \frac{(n-1)\times(n-2)}{2}$$

$$= \frac{n^2-2n-n+2}{2} = \frac{n^2-3n+2}{2}$$

For Big-O notation remove all constant then, $n^2 - n$
Now received only upper value n^2
∴Big-O of bubble sort is O(n^2)

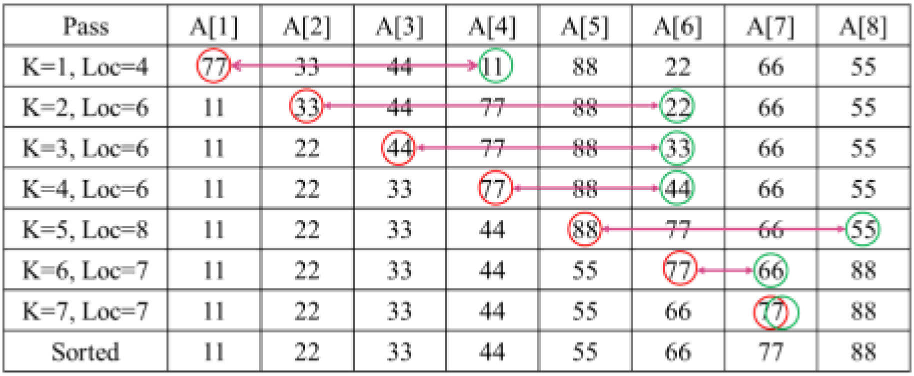
❖ Selection Sort

- Selection Sort is a simple comparison-based sorting algorithm. The basic idea of this algorithm is to divide the input list into two parts: a sorted sub list and an unsorted sub list. The algorithm repeatedly selects the smallest (or largest, depending on the sorting order) element from the unsorted sub list and swaps it with the first element of the unsorted sub list. This process continues until the entire list is sorted.

❖ Example 3.3

Sort the array 77, 33, 44, 11, 88, 22, 66, 55 in ascending order using Selection sort technique

Solution:



Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1, Loc=4	77	33	44	11	88	22	66	55
K=2, Loc=6	11	33	44	77	88	22	66	55
K=3, Loc=6	11	22	44	77	88	33	66	55
K=4, Loc=6	11	22	33	77	88	44	66	55
K=5, Loc=8	11	22	33	44	88	77	66	55
K=6, Loc=7	11	22	33	44	55	77	66	88
K=7, Loc=7	11	22	33	44	55	66	77	88
Sorted	11	22	33	44	55	66	77	88

After sorting the array is 11, 22, 33, 44, 55, 66,77, 88

❖ Selection Sort Algorithm

```
1. Repeat steps 2 and 3 for k1 to N-1
2. Call MIN (DATA, K, N, LOC)
3. [Interchange A[K] and A[LOC]
   Set TEMP: =A[K], A[K]=A[LOC] and A[LOC]: =TEMP
   [End of step 1 loop]
4. Exit
```

```
MIN (DATA, K, N, LOC)
1. Set MIN: =A[K] and LOC: =K [Initializes pointers]
2. Repeat for j=K+1 to N
    If MIN>A[j] then: set MIN: =A[j] and LOC: =j
    [End of loop]
3. Return
```

❖ Selection Sort Big-O notation

The number $f(n)$ of comparisons in the bubble sort is easily completed.

There are $n-1$ comparisons during the first pass.

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

$$= \frac{(n-1)\times(n-1-1)}{2}$$

$$= \frac{(n-1)\times(n-2)}{2}$$

$$= \frac{n^2-2n-n+2}{2} = \frac{n^2-3n+2}{2}$$

For Big-O notation remove all constant then, $n^2 - n$

Now received only upper value n^2

∴Big-O of bubble sort is O(n^2)

❖ Quick sort

- Quick sort is a “divide and conquer” algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick sort that pick pivot in different ways.
  1. Always pick first element as pivot.
  2. Always pick last element as pivot.
  3. Pick a random element as pivot.
  4. Pick median as pivot

Rule: In Quick sort algorithm firstly take a pivot value (last vale). Then compare the other values and sort then.
নিয়মঃ Quick sort algorithm এর ক্ষেত্রে একটি pivot value (শেষের value) ধরে নিতে হবে। তারপর তার সাথে compare করে করে sort করতে হবে।

❖ Example 3.4

Sort the array 2, 8, 7, 1, 3, 5, 6, 4 in ascending order using Quick sort technique

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

Call-1(Initial call): QUICK-SORT(1,8)
low=1, high=8, q=partition(1,8)
pivot=a[8]=4,i=1-1=0
for j=1 to 7
when j=1,a[1]=2;As 2<=4 true, i=0+1=1, swap a[1],a[1]

2	8	7	1	3	5	6	4

when j=2,a[2]=8;As 8<=4 false, continue loop
when j=3,a[3]=7;As 7<=4 false, continue loop
when j=4,a[4]=1;As 1<=4 true, i=1+1=2, swap a[2],a[4]

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

when j=5,a[5]=1;As 3<=4 true, i=2+1=3, swap a[3],a[5]

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

when j=6,a[6]=5;As 5<=4 false, continue loop
when j=7,a[7]=6;As 6<=4 false, continue loop
when j=8 break loop.

swap a[4],a[8]

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

return 4
q=4
Call-2(line 3): QUICK-SORT(1,3)
low=1,high=3,q= partition(1,3)
pivot=a[3]=3,i=1-1=0
for j=1 to 2
when j=1,a[1]=2;As 2<=3 true, i=0+1=1, swap a[1],a[1]

2	1	3	4	7	5	6	8

when j=2,a[2]=1;As 1<=3 true, i=1+1=2, swap a[2],a[2]

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

when j=3 break loop.
swap a[3],a[3]

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

return 3
q=3
Call-3(line 3): QUICK-SORT(1,2)
low=1,high=2,q= partition(1,2)
pivot=a[2]=1,i=1-1=0
for j=1 to 1
when j=1,a[1]=2;As 2<=1 false, continue loop.
swap a[1],a[2]

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

return 1
q=1
Call-4(line 3): QUICK-SORT(1,0)
low=1,high=0
condition false
Done Call-4
So, control returned to Call-3
Call-5(line 4): QUICK-SORT(2,2)
low=2,high=2
condition false
Done Call-5
So, control returned to Call-3
Done also Call-3

So, control returned to Call-2
Call-6(line 4): QUICK-SORT(4,3)
low=4,high=3
condition false
Done Call-6
So, control returned to Call-2
Done also Call-2
So, control returned to Call-1
Call-7(line 4): QUICK-SORT(5,8)
low=5,high=8, q= partition(5,8)
pivot=a[8]=8,i=5-1=4
for j=5 to 7
when j=5,a[5]=7;As 7<=8 true, i=4+1=5, swap a[5],a[5]

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

when j=6,a[6]=5;As 5<=8 true, i=5+1=6, swap a[6],a[6]

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

when j=7,a[7]=6;As 6<=8 true, i=6+1=7, swap a[7],a[7]

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

when j=8 break loop.
swap a[8],a[8]

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

return 8
q=8
Call-8(line 3): QUICK-SORT(5,7)
low=5,high=7, q= partition(5,7)
pivot=a[7]=6,i=5-1=4
for j=5 to 6
when j=5,a[5]=7;As 7<=6 false, continue loop.
when j=6,a[6]=5;As 5<=6 true, i=4+1=5, swap a[5],a[6]

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

when j=7 break loop.
swap a[6],a[7]

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

return 6
q=6
Call-10(line 4): QUICK-SORT(7,7)
low=7,high=7

condition false
Done Call-10
So, control returned to Call-8
Done also Call-8
So, control returned to Call-7
Call-11(line 4): QUICK-SORT(9,8)
low=9,high=8
condition false
Done Call-11
So, control returned to Call-7
Done also Call-7
So, control returned to Call-1
Done also Call-1
So, QUICK-SORT process is done
Final sorted list

1	2	3	4	5	6	7	8

❖ Quick Sort Algorithm

```
QUICK-SORT (low, high)
1.if low<high, then:
2.set: q=PARTITION (low, high)
3. QUICK-SORT (low, q-1)
4. QUICK-SORT (q+1, high)
   [end if structure]

PARTITION (low, high)
1.set: pivot=a[high]
2. i=low-1
3. repeat for j=low to high-1
4.if a[j]<=pivot
5.i=i+1
6.swap a[i], a[j]
    [end if structure]
 [end of loop]
7. swap a[i+1], a[high]
8.return i+1
```

❖ Heap Sort

- Heap sort is a comparison-based sorting technique based on Binary Heap data structure. 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 the remaining elements.
- There are two kinds of heap sort-
  i. Max heap (For descending order, big→small)
     - Here, root→big & child→small

  ii. Min heap (For ascending order, small→ big)
     - Here, root→small & child→big
- In this part we describe max heap

❖ Max Heap

- We shall use the same example to demonstrate how a Max Heap is created. The procedure to create Min Heap is similar but we go for min values instead of max values. We are going to derive an algorithm for max heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in an already heapified tree.
- **Step 1** − Create a new node at the end of heap.
- **Step 2** − Assign new value to the node.
- **Step 3** − Compare the value of this child node with its parent.
- **Step 4** − If value of parent is less than child, then swap them.
- **Step 5** − Repeat step 3 & 4 until Heap property holds.

Rule: In case of Max heap, a complete binary tree should be created with the given values first, but root will always be big and child will always be small. After the tree is created, if you delete the value from the root, it will be sorted.

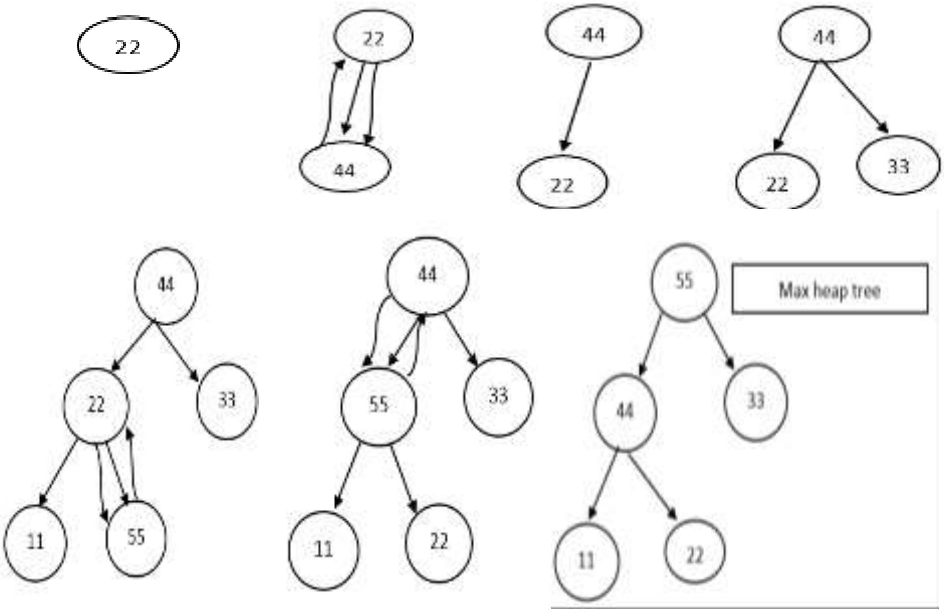
নিয়মঃ Max heap এর ক্ষেত্রে প্রথমে যে value গুলো দেওয়া থাকবে সেগুলো দিয়ে একটি complete binary tree তৈরি করতে হবে কিন্তু root সবসময় big এবং child

সবসময় small ছোট হবে। এরপর tree তৈরি করা হয়ে গেলে root থেকে value delete করলে sort হয়ে যাবে।

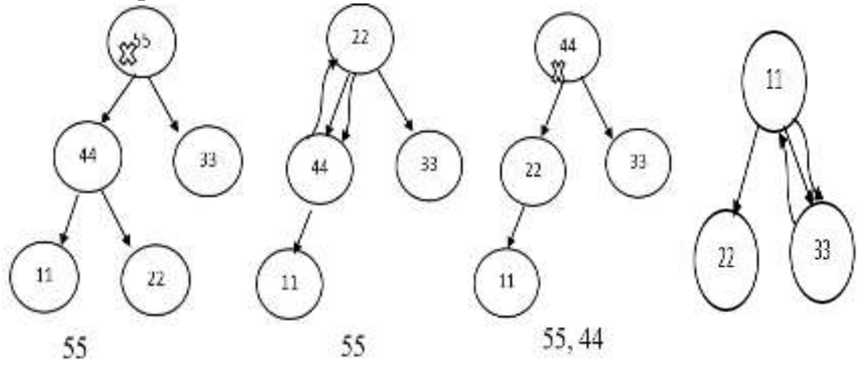
❖ Example 3.5

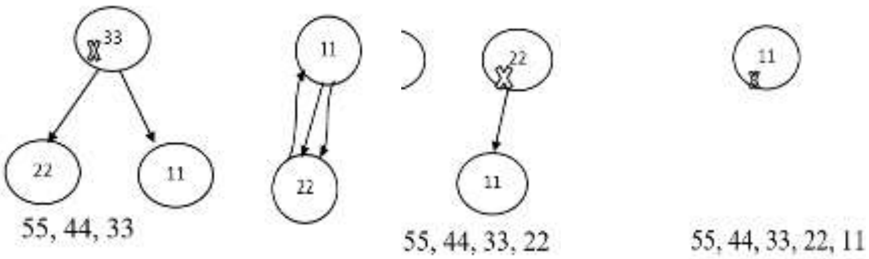
Sort the array 22, 44, 33, 11, 55 using max heap

Solution:
Create max heap



Max heap delete





After sorting the array is 55, 44, 33, 22, 11

❖ Heap Sort Algorithm

```
HEAPSORT (A)
1. BUILD-MAX-HEAP (A)
2. for i=A.length downto 2
3. exchange A[1] with A[i]
4. A.heap-size=A.heap-size-1
5. MAX-HEAPIFY(A)
```

```
BUILD-MAX-HEAP (A)
1. A.heap-size=A.length
2. for i=$\lfloor A.length/2 \rfloor$ downto 1
3. MAX-HEAPIFY (A,i )
```

```
MAX-HEAPIFY (A,i )
1. l=LEFT(i )
2. r=RIGHT(i)
3. if l<=A.heap-size and A[l]>A[i]
4. largest=l
5. else largest=i
6. if r<=A.heap-size and A[r]>A[largest]
7. largest=r
8. if largest≠i
9. exchange A[i] with A[largest]
10. MAX-HEAPIFY(A,largest)
```

❖ Exercise

4.1. Write down the short notes of Merge Sort and find out the Big-O notation of Merge Sort.

4.2. State the Merge Sort algorithm.

4.3. How to find out the MID value for Merge Sort explain with proper example.

4.4. Suppose (13, 93, 27, 18, 9, 55, 70, 83, 60) these are the value of an array. Now sort the array in ascending order using Merge Sort.

4.5. Suppose (13, 93, 27, 18, 9, 55, 70, 83, 60) these are the value of an array. Now sort the array in descending order using Merge Sort.

4.6. Write down the short notes of Bubble Sort and find out the Big-O notation of Bubble Sort.

4.7. State the Bubble Sort algorithm.

4.8. How to find out the number of Pass and Iteration in Bubble Sort explain with proper example.

4.9. Suppose (14, 83, 72, 18, 9, 53, 69, 38, 3) these are the value of an array. Now sort the array in ascending order using Bubble Sort.

4.10. Suppose (14, 83, 72, 18, 9, 53, 69, 38, 3) these are the value of an array. Now sort the array in descending order using Bubble Sort.

4.11. Write down the short notes of Selection Sort and find out the Big-O notation of Selection Sort.

4.12. State the Selection Sort algorithm.

4.13. Suppose (14, 83, 72, 81, 9, 35, 69, 38, 3) these are the value of an array. Now sort the array in ascending order using Selection Sort.

4.14. Suppose (14, 83, 72, 81, 9, 35, 69, 38, 3) these are the value of an array. Now sort the array in descending order using Selection Sort.

4.15. Write down the short notes of Quick Sort and state the Quick Sort Algorithm.

4.16. Suppose (14, 83, 72, 18, 9, 53, 69, 38, 3) these are the value of an array. Now sort the array in descending order using Quick Sort.

4.17. Suppose (14, 83, 72, 18, 9, 53, 69, 38, 3) these are the value of an array. Now sort the array in ascending order using Quick Sort.

4.18. Write down the short notes of Heap Sort and explain the types of Heap Sort.

4.19. Sort the array 33, 45, 13, 9, 60, 54 in descending using max heap

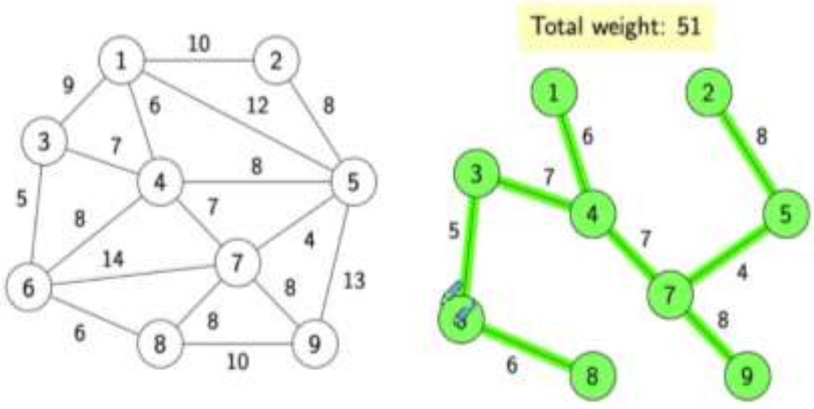
4.20. Sort the array 33, 45, 13, 9, 60, 54 in ascending using min heap

Chapter-4
Greedy Algorithm

Greedy algorithms are like dynamic programming algorithms that are often used to solve optimal problems (find best solutions of the problem according to a particular criterion). Greedy algorithms implement optimal local selections in the hope that those selections will lead to an optimal global solution for the problem to be solved. Greedy algorithms are often not too hard to set up, fast (time complexity is often a linear function or very much a second-order function). Besides, these programs are not hard to debug and use less memory. But the results are not always an optimal solution.

❖ Minimum Spanning Tree

Minimum spanning tree since T is acyclic and connects and all of the vertices, it must from a tree, which we call a spanning tree since it “spans” the graph G. We call the problem of determining the tree T the minimum-spanning-tree problem.



❖ Prims Method

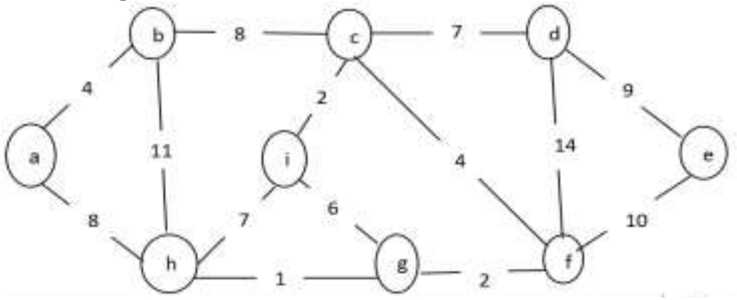
Prims algorithm is used to find the minimum spanning tree from a graph. Prims algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized. The edges with the minimal weights causing no cycles in the graph got selected.

Rule: In case of Prims algorithm, first go from the starting vertices to the next vertices with the lowest cost. After that, among the vertices that have been visited, the vertices that can go to other vertices at the lowest cost must be selected and if the cost of multiple vertices is equal, anyone can be taken, but no loop can be done. After visiting all the vertices, add the costs of the visited edges and calculate the total cost.

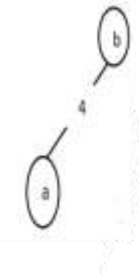
নিয়মঃ Prims algorithm এর ক্ষেত্রে প্রথমে starting vertices থেকে সবচেয়ে কম cost দিয়ে পরের vertices এ যেতে হবে। এর পর যতগুলো vertices visit করা আছে তার মধ্যে যে vertices দিয়ে সবচেয়ে কম cost এ অপর vertices এ যাওয়া যায় সে vertices কে select করতে হবে ও একাধিক vertices এর cost সমান হলে যে কোনটি নেওয়া যাবে, তবে কোনো loop হতে পারবে না। সবগুলো vertices visit করা শেষে visit করা edge এর cost গুলো যোগ করে total cost বের করতে হবে।

❖ Example 4.1

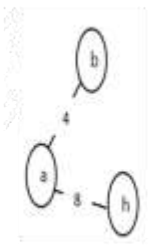
Find the minimum shortest path from the following graph using Prims Algorithm



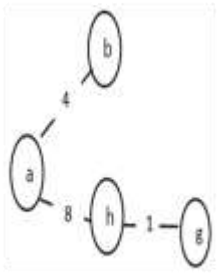
Solution:



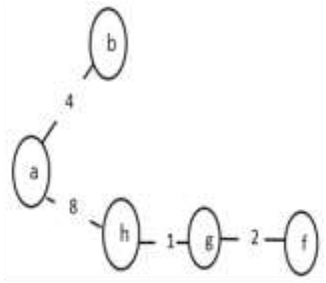
1st pass



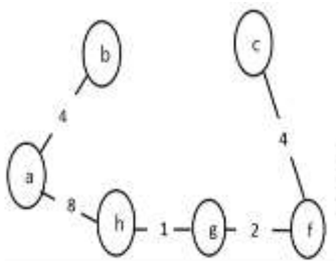
2nd pass



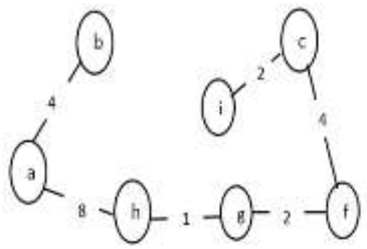
3rd pass



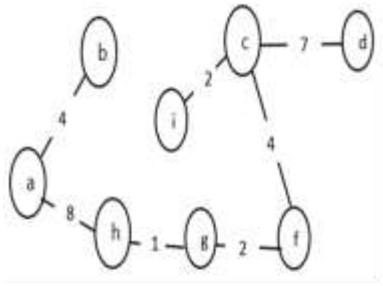
4th pass



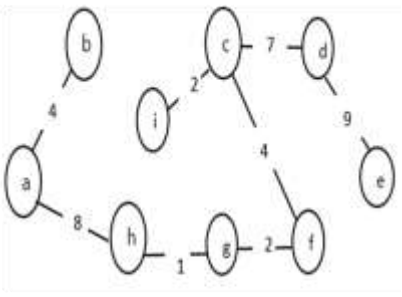
5th pass



6th pass



7th pass



8th pass

Total minimum cost = 4+8+1+2+4+2+7+9 = 37

❖ Prims Algorithm

```
MST-Prim (G, w, r)
1. for each u∈G.V
2. u.key=∞
3. u.π=Nil
4. r.key=0
5. Q=G.V
6. while Q≠ ∞
7. u=EXTRACT-MIN(Q)
8. for each v∈G.Adj[u]
9. If v∈ Q and w(u,v)<v.key
10. v. π=u
11. v.key=w(u, v)
```

❖ Prims Algorithm Big-O notation

```
MST-Prim (G, w, r)
1. for each $u \in G.V \longrightarrow O(V)$
2. $u.key=\infty$
3. $u.\pi$=Nil
4. r.key=0
5. Q=G.V $\longrightarrow O(V)$
6. while $Q \neq \infty$
7. u=EXTRACT-MIN(Q) $\longrightarrow O(VlogV)$
8. for each $v \in G.Adj[u] \longrightarrow O(V)$
9. If $v \in Q$ and w(u,v)<v.key
10. v. $\pi$=u
11. v.key=w(u,v) $\longrightarrow O(ElogV)$
```

O(V)+O(V)+O(VlogV)+O(V)+O(ElogV)
=3O(V)+O(VlogV)+O(ElogV)
After remove all constant
O(V)+O(VlogV)+O(ElogV)
Accept only higher value O(ElogV)
∴ Big-O of Prims algorithm is O(ElogV)

❖ Kruskal Method

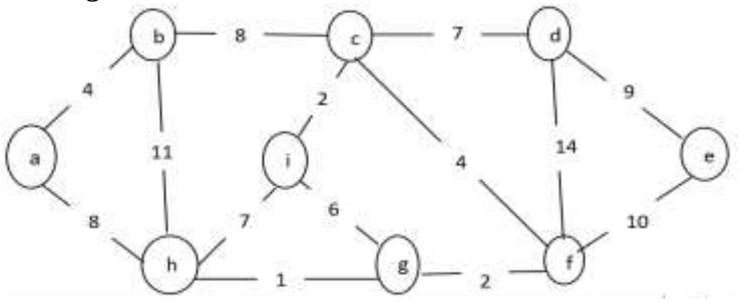
It is an algorithm for finding the minimum cost spanning tree of the given graph. In kruskal's algorithm, edges are added to the spanning tree in increasing order of cost. If the edge E forms a cycle in the spanning it is discarded.

Rule: In the case of Kuskal's algorithm, first all the costs of the edge should be arranged from the smallest to the largest. After that the vertices have to be visited from small cost. But if any cost becomes a loop then it should be removed.

নিয়মঃ Kuskal's algorithm এর ক্ষেত্রে প্রথমে সবগুলো edge এর cost গুলো ছোটো থেকে বড় সাজিয়ে নিতে হবে। এর পর ছোটো cost থেকে vertices গুলো visit করতে হবে। কিন্তু কোনো cost নিলে যদি loop হয়ে যায় তাহলে তা বাদ দিতে হবে।

❖ Example 4.2

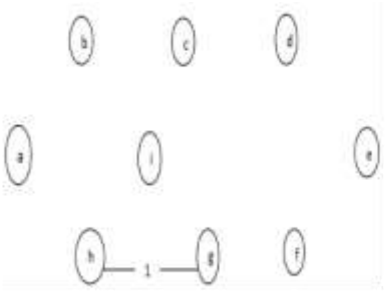
Find the minimum shortest path from the following graph using Kruskal Algorithm



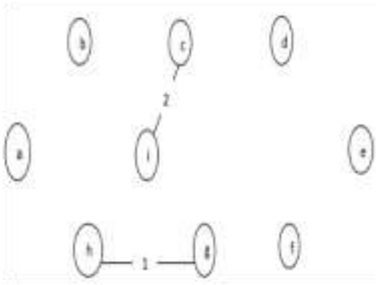
Solution:

Cost	1	2	2	4	4	6	7	7	8	8
Edge	h-g	c-i	g-f	a-b	c-f	i-g	c-d	h-i	b-c	a-h

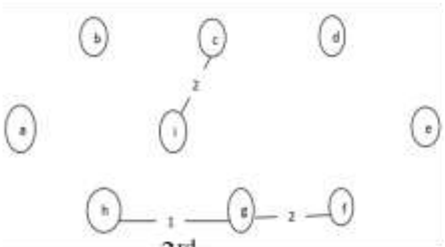
Cost	9	10	11	14
Edge	d-e	e-f	b-h	d-f



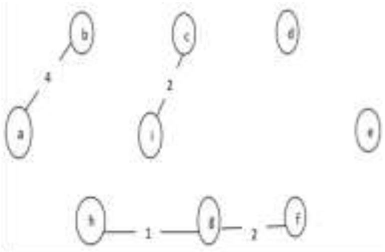
1st pass



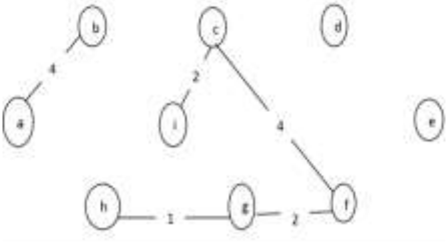
2nd pass



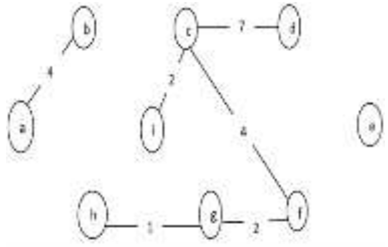
3rd pass



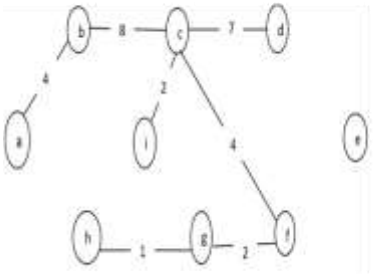
4th pass



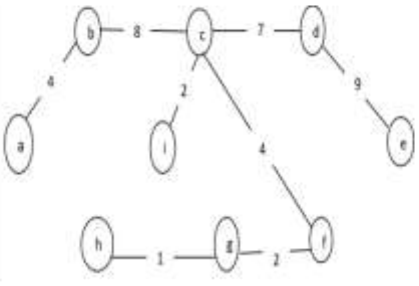
5th pass



6th pass



7th pass



8th pass

Total minimum cost = 1+2+2+4+4+7+8+8 = 37

❖ Kruskal Algorithm

```
MST-Kuskal(G,w)
1. $A=\emptyset$
2. for each $u \in G.V$
3. MAKE-SET (v)
4. Sort the edges of G.E into nondecreasing order by weight w.
5. for each edge $(u,v) \in G.E$, taken in nondecreasing order by
weight w.
6. If FIND-SET(u) $\neq$ FIND-SET(v)
7. $A = A \cup \{(u,v)\}$
8. UNION(u,v)
9. Return A
```

❖ Kruskal Algorithm Big-O notation

```
                  MST-Kuskal (G, w)
O (1)        {  1.  $A=\emptyset$
O (V)        {  2.  for each $u \in G.V$
             {  3.  MAKE-SET (v)
O (E logE)   {  4.  Sort the edges of G.E into no decreasing order by
             {      weight w.
             ┌  5.  for each edge (u, v) $\in G.E$, taken in no decreasing
             │      order by weight w.
O (V logV)   │  6.  If FIND-SET(u) $\neq$ FIND-SET(v)
             │  7.  $A = A \cup \{(u,v)\}$
             │  8.  UNION (u, v)
             └  9.  Return A
```

T(n) = O(1) + O(V) + O(E logE) + O(V logV)

= O(E logE) + O(V logV)

as $|E| >= |V| - 1$ T(n) = E logE + E logE = 2E logE

Accept only higher value O (E logE)

∴ Big-O of Prims algorithm is O (E logE)

❖ Knapsack Problem Fractional

The knapsack problem is a problem in combinational optimization: Given a set of items, each with a weight 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. It derives its name from the problem faced by someone who is constrained by a fixed size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

Rule: In the case of 0/1 Knapsnak, we have to work with the largest Max, but if the Weight of Max is more than the Weight of the Remaining, then we will work with the Weight of the Remaining and calculate the Profit according to the following rule.

নিয়মঃ 0/1 Knapsnak এর ক্ষেত্রে আমাদের কে সবচেয়ে বড় Max কে নিয়ে কাজ করতে হবে তবে যদি Remaining এ থাকা Weight এর থেকে Max এর Weight বেশি হয় তবে Remaining এ যে Weight আছে তা নিয়ে কাজ করব এবং নিচের নিয়মে Profit বের করব।

❖ Example 4.3

Object	1	2	3	4	5	6	7
Profit	12	5	16	7	9	11	6
Weight	3	1	4	2	9	4	3

In this problem the maximum weight is 15KG. Now, find out the maximum profit using 0/1 Knapsack Fractional method.

Solution:

Object	1	2	3	4	5	6	7
Profit	12	5	16	7	9	11	6
Weight	3	1	4	2	9	4	3
Max (P/W)	4	5	4	3.5	1	2.75	2

Object	Profit	Weight	Remaining
2	5	1	15-1=14
1	12	3	14-3=11
3	16	4	11-4=7
4	7	2	7-2=5
6	11	4	5-4=1
7	2	1	1-1=0
Total	53 Taka	15 KG	0

Maximum profit is 53 Taka

❖ Knapsack Problem Fractional Algorithm

```
Fractional Knapsack (Array v, Array w, int W)
1.for i=1 to size (v)
2.do p [i]=v[i]/w[i]
3.Sort-Descending (p)
4.i←1
5.while (W>0)
6.do amount=min (W, w[i])
7.solution[i]=amount
8.W=W-amount
9.i ← i+1
10.return solution
```

❖ Dijkstra Method

Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph G=(V, E) for the case in which all edge weights are nonnegative. In this section, therefore, we assume that w(u,v)>=0 for each edge (u,v)∈E. As we shall see, with a good implementation, the running time of Dijkstra's algorithm is lower than that of the Bellman-Ford algorithm. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex u∈V S with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. In the following implementation, we use a min-priority queue Q of vertices, keyed by their d values.

Rule:

In case of Dijkstra algorithm, all vertices must be visited. It can be for both directed/undirected graph.

Conditions:

1. A Vertices once visited cannot be visited again.
2. If the current distance is less than the previous distance, the current distance should be taken. If[d(u)+c(u,v)<=d(v)] then d(v)=d(u)+c(u,v)] will calculate the distance.
3. The smallest distance should always be selected for the visit. In this case, if the distance of more than one vertices is the same, the order of vertices should be followed.
4. Negative distance cannot exist.

নিয়ম:

Dijkstra algorithm এর ক্ষেত্রে সবগুলো Vertices visit করতে হবে। এটি Directed/ undirected graph দুইটার জন্যই হতে পারে। শর্তঃ

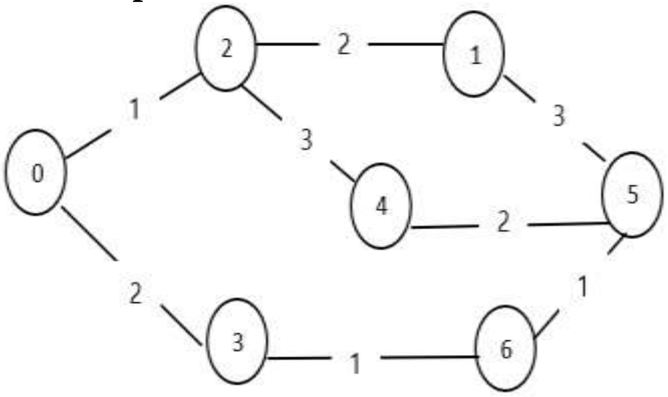
১। কোনো Vertices একবার visit করলে তা আর visit করা যাবে না।

২। বর্তমান distance পূর্বের distance এর ছোটো হলে বর্তমান distance নিতে হবে। এক্ষেত্রে If[d(u)+c(u,v)<=d(v)] then d(v)=d(u)+c(u,v)] সুত্র প্রয়োগ করে distance বের করবো।

৩। Visit এর জন্য সবসময় সবচেয়ে ছোটো Distance কে select করতে হবে এক্ষেত্রে একের অধিক Vertices এর distance same হলে vertices এর ক্রমানুসারে যেতে হবে।

৪। Negative distance থাকতে পারবে না।

❖ Example 4.4



Find single source shortest path using Dijkstra algorithm

Solution:

If[d(u)+c(u,v)<=d(v)] then d(v)=d(u)+c(u,v) formula apply							
Visit	0	1	2	3	4	5	6
0	**0**	∝	∝	∝	∝	∝	∝
2		∝	**1/0**	2/0	∝	∝	∝
3		3/2		**2/0**	4/2	∝	∝
1		**3/2**			4/2	∝	3/3
6					4/2	6/1	**3/3**
4					**4/2**	4/6	
5						**4/6**	

u=0,v=2
d(0)+c(0,2)=0+1=1<∝ d(2)=1
u=0,v=3
d(0)+c(0,3)=0+2=2<∝ d(3)=2
u=2,v=1
d(2)+c(2,1)=1+2=3<∝ d(1)=3
u=2,v=4
d(2)+c(2,4)=1+3=4<∝ d(4)=4
u=3,v=6
d(3)+c(3,6)=2+1=3<∝ d(6)=3

u=1,v=5	
d(1)+c(1,5)=3+3=6<∝ d(5)=6	
u=6,v=5	
d(6)+c(6,5)=3+1=4<6 d(5)=4	
u=4,v=5	
d(4)+c(4,5)=4+2=6>4	
Distance & Path	
Distance(1)=3	Path: 1->2->0
Distance(2)=1	Path: 2->0
Distance(3)=2	Path: 3->0
Distance(4)=4	Path: 4->2->0
Distance(5)=4	Path: 5->6->3->0
Distance(6)=3	Path: 6->3->0

❖ Dijkstra Algorithm

```
1.  DIJKSTRA(G,w,s)
2.  1.INITIALIZE-SINGLE-SOURCE(G,s)
3.  2.S=∅
4.  3.Q=G.V
5.  4.while Q≠∅

    5.u=EXTRACT-MIN(Q)
6.  6.S=S∪{u}
7.  7.for each vertex v∈G.Adj[u]
8.  8.RELAX(u,v,w)
```

❖ Exercise

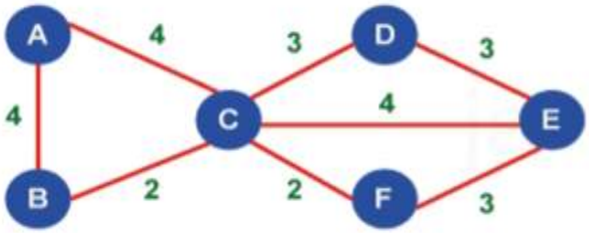
4.1. Write down the short notes of Minimum Spanning Tree and explain the types of Minimum Spanning Tree with proper example.

4.2. Find out the Big-O notation of Prims method and state the Prims algorithm.

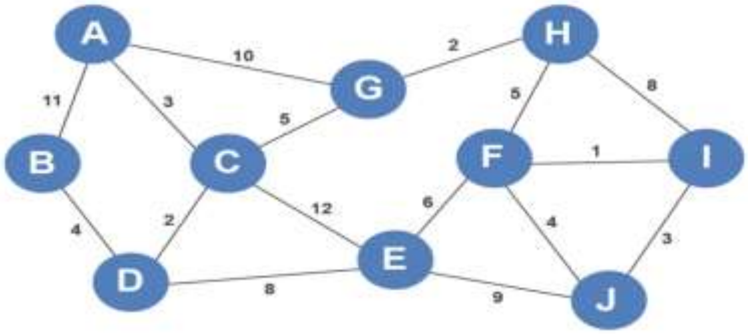
4.3. Find out the Big-O notation of Kruskal method and state the Kruskal algorithm.

4.4. Differentiate between Prims method and Kruskal Method

4.5. Find the minimum shortest path from the following graph using Prims Algorithm

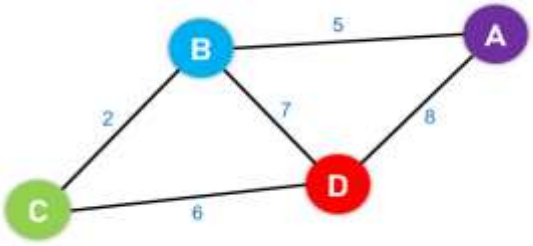


4.6. Find the minimum shortest path from the following graph using Kruskal Algorithm



4.7. Write down the short notes of Dijkstra method and state the Dijkstra algorithm.

4.8. Find single source shortest path from the following graph using Dijkstra algorithm



4.9. In this problem the maximum weight is 15KG. Now, find out the maximum profit using 0/1 Knapsack Fractional method.

Object	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1

Chapter-5
Dynamic Algorithm

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of sub problems, time complexity reduces to linear.

❖ 0/1 Knapsack

Fractional Knapsack problem using Greedy approach. We have shown that Greedy approach gives an optimal solution for Fractional Knapsack. However, this chapter will cover 0-1 Knapsack problem and its analysis. In 0-1 Knapsack, items cannot be broken which means the thief should take the item as a whole or should leave it. This is reason behind calling it as 0-1 Knapsack. Hence, in case of 0-1 Knapsack, the value of xi can be either 0 or 1, where other constraints remain the same. 0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution. In many instances, Greedy approach may give an optimal solution. Now we solve this problem using dynamic approach.

Rule: In the case of 0/1 Knapsack, if you take any one object, you either have to take the whole thing or you can't take it. But no fraction can be taken. In this case, a table should be created in which weight (up to max weight) should be written along the row and item should be written along the column and its value should be set to 0 at the beginning. After that, all the values up to the weight of the item to be worked on should be written and the rest of the values should be added to the profit of that item in order from the first value of the previous row. However, if the previous value is larger, it should be taken. Or, B[i,w]=max(B[i-1,w],B[i-1,w-w[i]+v[i]) can also be done with this rule.

নিয়মঃ 0/1 Knapsack এর এক্ষেত্রে কোনো একটি object নিলে ওইটা হয় পুরোটা নিতে হবে নাহয় নিতে পারব না। কিন্তু কোনো Fraction নেওয়া যাবে না। এক্ষেত্রে একটি টেবিল তৈরি করতে হবে যার row বরাবর weight (max weight পর্যন্ত) এবং column বরাবর item লিখতে হবে এবং শুরুতে এর মান 0 করে দিতে হবে। এরপর যে item নিয়ে কাজ করব তার weight এর আগ পর্যন্ত সব value লিখে ফেলতে হবে এবং বাকি value গুলো তার আগের row এর প্রথম value থেকে ক্রমানুসারে ঐ item এর profit এর সাথে যোগ করে লিখতে হবে। তবে আগের value বড় হলে সেটা নিতে হবে। অথবা, B[i,w]=max(B[i-1,w],B[i-1,w-w[i]+v[i]) এই rule দিয়েও করা যাবে।

❖ Example 5.1

Items/objects:	1	2	3	4		max weight=5
Weight:	3	2	5	4		n=4
Value/Profit:	4	3	6	5		

Find out the maximum profit using 0/1 Knapsack method.

Solution:

	0	1	2	3	4	5
0	0	0	0	0	0	0
✓ 1	0	0	0	4	4	4
✓ 2	0	0	3	4	4	7
✗ 3	0	0	3	4	4	7
✗ 4	0	0	3	4	5	7

B[4,4]=max(B[4-1,4],B[4-1,4-w[4]+v[4])
= max(B[3,4],B[3,4-4]+5)
=max(B[3,4], B[3,0]+5)
=max(4,0+5)=max(4,5)=5
B[4,5]=max(B[4-1,5],B[4-1,5-w[4]+v[4])
= max(B[3,5],B[3,5-4]+5)
=max(B[3,5],B[3,1]+5)
=max(7,0+5)=max(7,5)=7

Profit=4+3=7

Item:	1	2	3	4	5-2=3
	1	1	0	0	3-3=0

Maximum profit is 7 Taka

❖ 0/1 Knapsack Algorithm

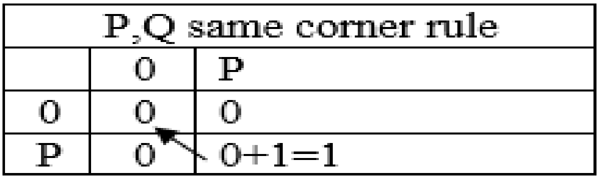
```
Dynamic-0-1-knapsack (v, w, n, W)
1.for w = 0 to W do
2.   c[0, w] = 0
3.for i = 1 to n do
4.   c[i, 0] = 0
5.   for w = 1 to W do
6.      if wi ≤ w then
7.         if vi + c[i-1, w-wi] then
8.            c[i, w] = vi + c[i-1, w-wi]
9.         else c[i, w] = c[i-1, w]
10.     else
11.        c[i, w] = c[i-1, w]
```

❖ Longest Common Subsequence

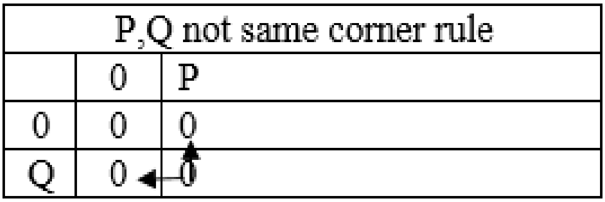
The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring problem: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences. The longest common subsequence problem is a classic computer science problem, the basis of data comparison programs such as the diff utility, and has applications in computational linguistics and bioinformatics. It is also widely used by revision control systems such as Git for reconciling multiple changes made to a revision-controlled collection of files.

Rule: For the longest common subsequence, first write X and Y in a row (starting with 0) and column (starting with 0) in a table, and the first row and column will be 0. Then if its value matches corner then they have to add 1 to the value of corner and if they don't match then they have to write the larger of front/top value.

নিয়মঃ Longest common subsequence এর ক্ষেত্রে প্রথমে X এবং Y কে একটি টেবিলে সারি (শুরুতে 0) এবং কলামে (শুরুতে 0) লিখতে হবে এবং প্রথম সারি ও কলাম 0 হবে। এরপর যদি এর মান কোনাকুনি মিলে যায় তাহলে তাদের কর্নার এর মানের সাথে ১ যোগ করে লিখতে হবে আর যদি না মিলে তাহলে তাদের সামনের/উপরের মানের মধ্যে যেটি বড় সেটি লিখতে হবে।



P,Q same corner rule		
	0	P
0	0	0
P	0	0+1=1



P,Q not same corner rule		
	0	P
0	0	0
Q	0	0

❖ Example 5.2

X=P R E S I D E N T

Y=P R O V I D E N C E

Find the longest common subsequence.

Solution:

	0	P	R	O	V	I	D	E	N	C	E	Check
0	0	0	0	0	0	0	0	0	0	0	0	
P	0	↖1	←1	←1	←1	←1	←1	←1	←1	←1	←1	
R	0	↑1	↖2	←2	←2	←2	←2	←2	←2	←2	←2	
E	0	↑1	↑2	←2	←2	←2	←2	↖3	←3	←3	↖3	
S	0	↑1	↑2	←2	←2	←2	←2	↑3	←3	←3	←3	PRIDEN=6
I	0	↑1	↑2	←2	←2	↖3	←3	←3	←3	←3	←3	
D	0	↑1	↑2	←2	←2	↑3	↖4	←4	←4	←4	←4	
E	0	↑1	↑2	←2	←2	↑3	↑4	↖5	←5	←5	↖5	
N	0	↑1	↑2	←2	←2	↑3	↑4	↑5	↖6	←6	←6	
T	0	↑1	↑2	←2	←2	↑3	↑4	↑5	↑6	←6	←6	

❖ Longest Common Subsequence Algorithm

```
LCS-Length-Table-Formulation (X, Y)
1.m :=length(X)
2.n := length(Y)
3.for i = 1 to m
4.   c[i, 0] := 0
5.for j = 1 to n
6.   c[0, j] := 0
7.for i = 1 to m
8.   for j = 1 to n
9.      if xi = yj
10.        c[i, j] := c[i - 1, j - 1] + 1
11.        b[i, j] := ‘ ’
12.     else if c[i -1, j] ≥ c[i, j -1]
13.           c[i, j] := c[i - 1, j]
14.           b[i, j] := ‘↑’
15.     else
```

```
16.         c[i, j] := c[i, j - 1]
17.         b[i, j] := '←'
18.return c and b

Print-LCS (B, X, i, j)
1.if i = 0 and j = 0
2.   return
3.if B[i, j] = 'D'
4.   Print-LCS(B, X, i-1, j-1)
5.   Print(xi)
6.else if B[i, j] = 'U'
7.   Print-LCS(B, X, i-1, j)
8.else
9.   Print-LCS(B, X, i, j-1)
```

❖ Bellman Ford

The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative. Given a weighted, directed graph G=(V,E) with source s and weight function w:E-->R, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

নিয়মঃ Bellman Ford এর ক্ষেত্রে iteration এর সংখ্যা হবে (n-1) সংখ্যক যেখানে n=vertices সংখ্যা। তবে যদি পরপর দুই বার iteration এর result same হয় তাহলে আর iteration করতে হবে না।

শর্তঃ

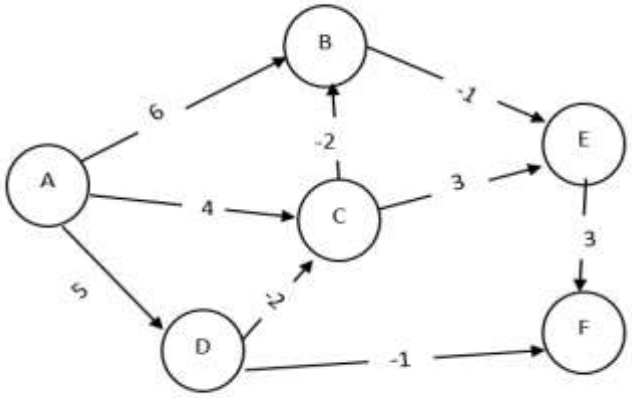
১। Graph এ কোনো loop থাকতে পারবে না।

২। If[d(u)+c(u,v)<d(v)] then d(v)=d(u)+c(u,v) এই সুত্র প্রয়োগ করে distance বের করবো।

৩। Directional graph হতে হবে ও negative cost থাকতে পারবে।

৪। প্রতিবার সব Vertices visit করতে হবে।

❖ Example 5.3



Find single source shortest path using Bellman Ford algorithm

Solution:

Here n=6 so, iteration=(6-1)=5							
1st iteration							
	A	B	C	D	E	F	u=A,v=B d(A)+c(A,B)=0+6=6<∝ d(B)=6 u=A,v=C d(A)+c(A,C)=0+4=4<∝ d(C)=4 u=A,v=D d(A)+d(A,D)=0+5=5<∝ d(D)=5 u=B,v=E d(B)+c(B,E)=6+(-1)=5<∝ d(E)=5 u=C,v=B d(C)+c(C,B)=4+(-2)=2<6 d(B)=2
A	**0**	∝	∝	∝	∝	∝	
B	0	**6**	4	5	∝	∝	
C	0	6	**4**	5	5	∝	
D	0	2	4	**5**	5	∝	
E	0	2	3	5	**5**	4	
F	0	2	3	5	5	**4**	
	0	2	3	5	5	4	
u=C,v=E d(C)+c(C,E)=4+3=7>5 d(E)=5 u=D,v=C d(D)+c(D,C)=5+(-2)=3<4 d(C)=3 u=D,v=F d(D)+c(D,F)=5+(-1)=4<∝ d(F)=4 u=E,v=F d(E)+c(E,F)=5+3=8>4 d(F)=4							

2nd iteration

	A	B	C	D	E	F
A	**0**	2	3	5	5	4
B	0	**2**	3	5	5	4
C	0	2	**3**	5	1	4
D	0	1	3	**5**	1	4
E	0	1	3	5	**1**	4
F	0	1	3	5	1	**4**
	0	1	3	5	1	4

u=A,v=B
d(A)+c(A,B)=0+6=6>2 d(B)=2
u=A,v=C
d(A)+c(A,C)=0+4=4>3 d(C)=3
u=A,v=D
d(A)+d(A,D)=0+5=5=5 d(D)=5
u=B,v=E
d(B)+c(B,E)=2+(-1)=1<5 d(E)=1
u=C,v=B
d(C)+c(C,B)=3+(-2)=1<2 d(B)=1
u=C,v=E
d(C)+c(C,E)=3+3=6>1 d(E)=1
u=D,v=C
d(D)+c(D,C)=5+(-2)=3=3 d(C)=3
u=D,v=F
d(D)+c(D,F)=5+(-1)=4=4 d(F)=4
u=E,v=F
d(E)+c(E,F)=1+3=4=4 d(F)=4

3rd iteration

	A	B	C	D	E	F
A	**0**	1	3	5	1	4
B	0	**1**	3	5	1	4
C	0	1	**3**	5	0	4
D	0	1	3	**5**	0	4
E	0	1	3	5	**0**	4
F	0	1	3	5	0	**3**
	0	1	3	5	0	3

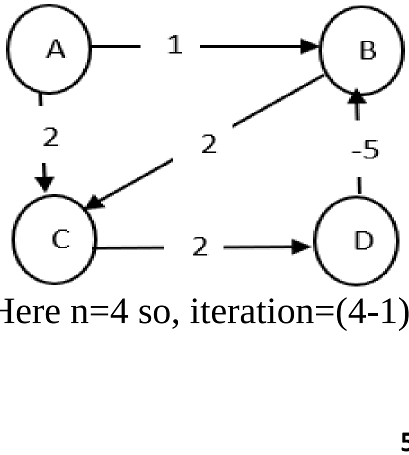
u=A,v=B
d(A)+c(A,B)=0+6=6>1 d(B)=1
u=A,v=C
d(A)+c(A,C)=0+4=4>3 d(C)=3
u=A,v=D
d(A)+d(A,D)=0+5=5=5 d(D)=5
u=B,v=E
d(B)+c(B,E)=1+(-1)=0<1 d(E)=0
u=C,v=B
d(C)+c(C,B)=3+(-2)=1=1 d(B)=1
u=C,v=E
d(C)+c(C,E)=3+3=6>0 d(E)=0
u=D,v=C
d(D)+c(D,C)=5+(-2)=3=3 d(C)=3
u=D,v=F
d(D)+c(D,F)=5+(-1)=4=4 d(F)=4
u=E,v=F
d(E)+c(E,F)=0+3=3<4 d(F)=3

4th iteration							
	A	B	C	D	E	F	u=A,v=B d(A)+c(A,B)=0+6=6>1 d(B)=1 u=A,v=C d(A)+c(A,C)=0+4=4>3 d(C)=3 u=A,v=D d(A)+d(A,D)=0+5=5=5 d(D)=5 u=B,v=E d(B)+c(B,E)=1+(-1)=0=0 d(E)=0 u=C,v=B d(C)+c(C,B)=3+(-2)=1=1 d(B)=1
A	**0**	1	3	5	0	3	
B	0	**1**	3	5	0	3	
C	0	1	**3**	5	0	3	
D	0	1	3	**5**	0	3	
E	0	1	3	5	**0**	3	
F	0	1	3	5	0	**3**	
	0	**1**	**3**	**5**	**0**	**3**	
u=C,v=E d(C)+c(C,E)=3+3=6>0 d(E)=0 u=D,v=C d(D)+c(D,C)=5+(-2)=3=3 d(C)=3 u=D,v=F d(D)+c(D,F)=5+(-1)=4>3 d(F)=3 u=E,v=F d(E)+c(E,F)=0+3=3=3 d(F)=3							

Here 3rd and 4th iteration are same so don't calculate 5th iteration.

Distance(A)=0 Distance(B)=1 Distance(C)=3	Distance(D)=5 Distance(E)=0 Distance(F)=3

❖ Drawback: কেনো হতে Loop পারবে না।



Here n=4 so, iteration=(4-1)=3

1st iteration					
	A	B	C	D	u=A,v=B d(A)+c(A,B)=0+1=1<∝ d(B)=1 u=A,v=C d(A)+c(A,C)=0+2=2<∝ d(C)=2 u=B,v=C d(B)+c(B,C)=1+2=3>2 d(C)=2 u=C,v=D d(C)+c(C,D)=2+2=4<∝ d(D)=4 u=D,v=B d(D)+c(D,B)=4+(-5)=-1<1 d(B)=-1
A	**0**	∝	∝	∝	
B	0	**1**	2	∝	
C	0	1	**2**	∝	
D	0	1	2	**4**	
	0	-1	2	5	

2nd iteration					
	A	B	C	D	u=A,v=B d(A)+c(A,B)=0+1=1<∝ d(B)=1 u=A,v=C d(A)+c(A,C)=0+2=2<∝ d(C)=2 u=B,v=C d(B)+c(B,C)=1+2=3>2 d(C)=2 u=C,v=D d(C)+c(C,D)=2+2=4<∝ d(D)=4 u=D,v=B d(D)+c(D,B)=4+(-5)=-1<1 d(B)=-1
A	**0**	∝	∝	∝	
B	0	**1**	2	∝	
C	0	1	**2**	∝	
D	0	1	2	**4**	
	0	-1	2	5	

3rd iteration					
	A	B	C	D	u=A,v=B d(A)+c(A,B)=0+1=1>-2 d(B)=-2 u=A,v=C d(A)+c(A,C)=0+4=4>1 d(C)=1 u=B,v=D d(B)+c(B,C)=-2+2=0<1 d(C)=0 u=C,v=D d(C)+c(C,D)=0+2=2<3 d(D)=2 u=D,v=B d(D)+c(D,B)=2+(-5)=-3<-2 d(B)=-3
A	**0**	-2	1	3	
B	0	**-2**	1	3	
C	0	-2	**0**	3	
D	0	-2	0	**2**	
	0	-3	0	2	

4th iteration					
	A	B	C	D	u=A,v=B d(A)+c(A,B)=0+1=1>-3 d(B)=-3 u=A,v=C d(A)+c(A,C)=0+4=4>0 d(C)=0 u=B,v=D d(B)+c(B,C)=-3+2=-1 d(C)=-1 u=C,v=D d(C)+c(C,D)=-1+2=1<2 d(D)=1 u=D,v=B d(D)+c(D,B)=1+(-5)=-4<-3 d(B)=-4
A	**0**	-3	0	2	
B	0	**-3**	0	2	
C	0	-3	**-1**	2	
D	0	-3	-1	**1**	
	0	-4	-1	1	

এখানে দেখা যাচ্ছে 3rd iteration এর পরও value change হচ্ছে কিন্তু নিয়ম অনুসারে 3rd iteration এর পর value change হতে পারবে না তাই graph এ loop থাকলে Bellmen Ford use করা যায় না

❖ Bellman Ford Algorithm

```
BELLMAN-FORD(G,w,s)
1.INITIALIZE-SINGLE-SOURCE(G,s)
2. for i=1 to |G.V|-1
3.    for each edge(u,v) ∈ G.E
4.        RELAX(u,v,w)
5.for each edge(u,v) ∈ G.E
6.    if v.d >u.d+w(u,v)
7.        return FALSE
8.return TRUE
```

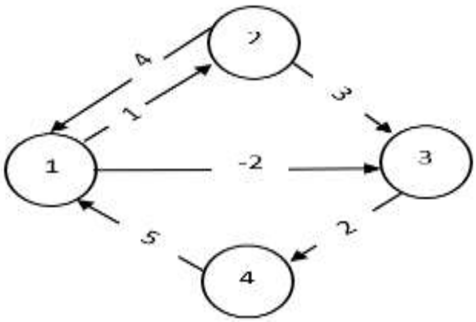
❖ Floyd Warshall

The Floyd Warshall algorithm (also known as Floyd's algorithm, the Roy Warshall algorithm, the Roy–Floyd algorithm, or the WFI algorithm) is an algorithm for finding shortest paths in a directed weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm. Versions of the algorithm can also be used for finding the transitive closure of a relation, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

Rule: Floyd warshall algorithm works for both positive & negative edge but Floyd warshall algorithm cannot be used for any negative cycle. It works for both directed & undirected graph.
Conditions:
1. All vertices must be visited with one vertices each time.
2. At the beginning a distance matrix has to be created.
3. After that, the distance matrix should be created as many vertices as there are and the row, column & diagonal value of the vertices to be worked with should be taken at the beginning. In this case the formula D^k[i,j]=min{D^{k-1}[i,j],D^{k-1}[i,k]+ D^{k-1}[k,j]} must be used.

নিয়ম: Floyd warshall algorithm positive & negative edge দুইটার জন্যই কাজ করে কিন্তু কোনো negative cycle এর ক্ষেত্রে Floyd warshall algorithm use করা যায় না। এটি directed & undirected graph দুইটার জন্যই কাজ করে।
শর্তঃ
১। প্রতিবার একটি vertices দিয়ে সবগুলো vertices visit করতে হবে।
২। শুরুতে একটি distance matrix তৈরি করতে হবে।
৩। এরপর যতগুলো vertices আছে ততগুলো distance matrix তৈরি করতে হবে এবং যে vertices নিয়ে কাজ করবো শুরুতে তার row, column & diagonal value নিয়ে নিতে হবে। এক্ষেত্রে এই সুত্র,
D^k[i,j]=min{D^{k-1}[i,j],D^{k-1}[i,k]+ D^{k-1}[k,j]} ব্যবহার করতে হবে।

❖ Example 5.4



Find all pair shortest path from the following graph using Floyd Warshall algorithm.

Solution:

D0	1	2	3	4
1	0	1	-2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	5	∞	∞	0

D1	1	2	3	4
1	0	1	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	5	6	3	0

$D^1[2,3]=\min\{D^0[2,3],D^0[2,1]+D^0[1,3]\}$
$=\min\{3,4+(-2)\}=\min\{3,2\}=2$

$D^1[2,4]=\min\{D^0[2,4],D^0[2,1]+D^0[1,4]\}$
$=\min\{\infty,4+\infty\}=\min\{\infty,\infty\}=\infty$

$D^1[3,2]=\min\{D^0[3,2],D^0[3,1]+D^0[1,2]\}$
$=\min\{\infty,\infty+1\}=\min\{\infty,\infty\}=\infty$

$D^1[3,4]=\min\{D^0[3,4],D^0[3,1]+D^0[1,4]\}$
$=\min\{2,\infty+\infty\}=\min\{2,\infty\}=2$

$D^1[4,2]=\min\{D^0[4,2],D^0[4,1]+D^0[1,2]\}$
$=\min\{\infty,5+1\}=\min\{\infty,6\}=6$

$D^1[4,3]=\min\{D^0[4,3],D^0[4,1]+D^0[1,3]\}$
$=\min\{\infty,5+(-2)\}=\min\{\infty,3\}=3$

D2	1	2	3	4
1	0	1	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	5	6	3	0

$D^2[1,3]=\min\{D^1[1,3],D^1[1,2]+ D^1[2,3]\}$
$=\min\{-2,1+2\}=\min\{-2,3\}=-2$
$D^2[1,4]=\min\{D^1[1,4],D^1[1,2]+ D^1[2,4]\}$
$=\min\{\infty,1+\infty\}=\min\{\infty,\infty\}=\infty$
$D^2[3,1]=\min\{D^1[3,1],D^1[3,2]+ D^1[2,1]\}$
$=\min\{\infty,\infty+4\}=\min\{\infty,\infty\}=\infty$
$D^2[3,4]=\min\{D^1[3,4],D^1[3,2]+ D^1[2,4]\}$
$=\min\{2,\infty+\infty\}=\min\{2,\infty\}=2$
$D^2[4,1]=\min\{D^1[4,1],D^1[4,2]+ D^1[2,1]\}$
$=\min\{5,6+4\}=\min\{5,10\}=5$
$D^2[4,3]=\min\{D^1[4,3],D^1[4,2]+ D^1[2,3]\}$
$=\min\{3,6+2\}=\min\{3,8\}=3$

D3	1	2	3	4
1	0	1	-2	0
2	4	0	2	4
3	∞	∞	0	2
4	5	6	3	0

$D^3[1,2]=\min\{D^2[1,2],D^2[1,3]+ D^2[3,2]\}$
$=\min\{1,-2+\infty\}=\min\{1,\infty\}=1$
$D^3[1,4]=\min\{D^2[1,4],D^2[1,3]+ D^2[3,4]\}$
$=\min\{\infty,-2+2\}=\min\{\infty,0\}=0$
$D^3[2,1]=\min\{D^2[2,1],D^2[2,3]+ D^2[3,1]\}$
$=\min\{4,2+\infty\}=\min\{4,\infty\}=4$
$D^3[2,4]=\min\{D^2[2,4],D^2[2,3]+ D^2[3,4]\}$
$=\min\{\infty,2+2\}=\min\{\infty,4\}=4$
$D^3[4,1]=\min\{D^2[4,1],D^2[4,3]+ D^2[3,1]\}$
$=\min\{5,3+\infty\}=\min\{5,\infty\}=5$
$D^3[4,2]=\min\{D^2[4,2],D^2[4,3]+ D^2[3,2]\}$
$=\min\{6,3+\infty\}=\min\{6,\infty\}=6$

D4	1	2	3	4
1	0	1	-2	0
2	4	0	2	4
3	7	8	0	2
4	5	6	3	0

$D^4[1,2]=\min\{D^3[1,2],D^3[1,4]+D^3[4,2]\}$
$=\min\{1,0+6\}=\min\{1,6\}=1$
$D^4[1,3]=\min\{D^3[1,3],D^3[1,4]+D^3[4,3]\}$
$=\min\{-2,0+0\}=\min\{-2,0\}=-2$
$D^4[2,1]=\min\{D^3[2,1],D^3[2,4]+D^3[4,1]\}$
$=\min\{4,4+5\}=\min\{4,9\}=4$
$D^4[2,3]=\min\{D^3[2,3],D^3[2,4]+D^3[4,3]\}$
$=\min\{2,4+3\}=\min\{2,7\}=2$
$D^4[3,1]=\min\{D^3[3,1],D^3[3,4]+D^3[4,1]\}$
$=\min\{\infty,2+5\}=\min\{\infty,7\}=7$
$D^4[3,2]=\min\{D^3[3,2],D^3[3,4]+D^3[4,2]\}$
$=\min\{\infty,2+6\}=\min\{\infty,8\}=8$

Which is required answer of all pair shortest path.

❖ Floyd Warshall Algorithm

```
FLOYD-WARSHALL(W)
1.n=W.rows
2.$D^{(0)}$=W
3.for k=1 to n
4.   let $D^{(k)}=d_{ij}{}^{(k)}$ be a new n×n matrix
5.   for i=1 to n
6.      for j=1 to n
7.         $d_{ij}{}^{(k)}=\min(d_{ij}{}^{(k-1)}, d_{ik}{}^{(k-1)}+d_{kj}{}^{(k-1)})$
8.return $D^{(n)}$
```

❖ Tower of Hanoi

The Tower of Hanoi (also called The problem of Benares Temple or Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers, or simply pyramid puzzle) is a mathematical game or puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C).

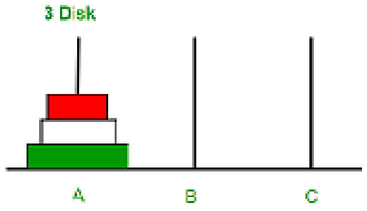
Rule:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk. ($2^n - 1$ moves for n disks)

নিয়ম:

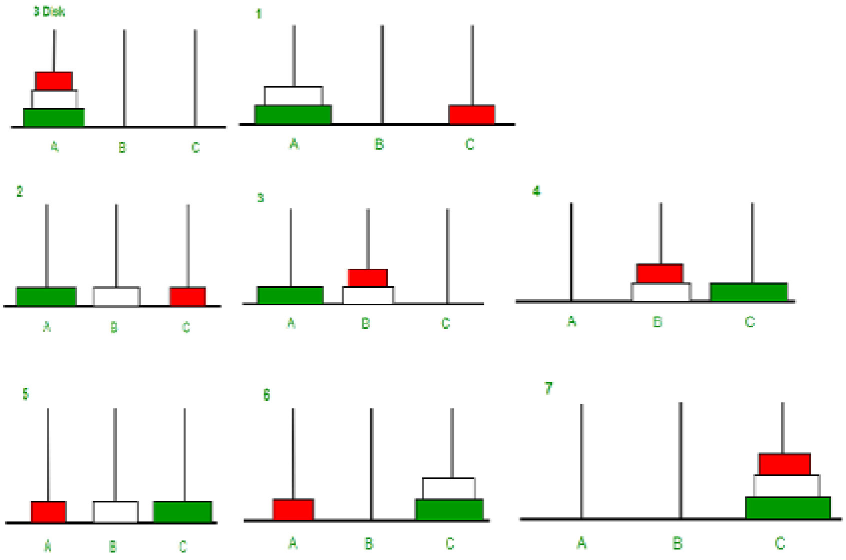
1. একবারে শুধুমাত্র একটি ডিস্ক সরানো যায়।
2. প্রতিটি পদক্ষেপে স্ট্যাকের একটি থেকে উপরের ডিস্কটি নেওয়া এবং এটিকে অন্য স্ট্যাকের উপরে স্থাপন করা হয় অর্থাৎ একটি ডিস্ক কেবল তখনই সরানো যেতে পারে যদি এটি একটি স্ট্যাকের উপরের ডিস্ক হয়।
3. একটি ছোট ডিস্কের উপরে কোন ডিস্ক স্থাপন করা যাবে না।

❖ Example 5.5



Move all disk from A to C

Solution:

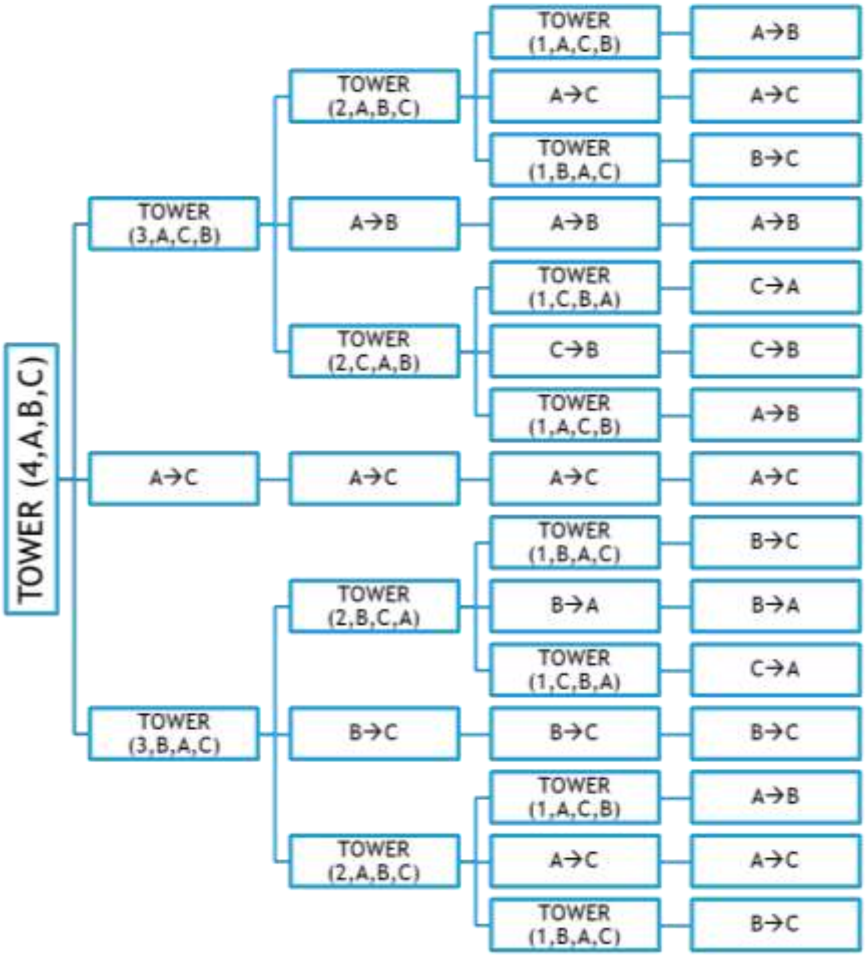


Disk 1 moved from A to C
Disk 2 moved from A to B
Disk 1 moved from C to B
Disk 3 moved from A to C
Disk 1 moved from B to A
Disk 2 moved from B to C
Disk 1 moved from A to C

❖ Tower of Hanoi Algorithm

```
Tower (N, BEG, AUX, END)
    1.  If N=1, then:
    (a) Write: BEG→END
    (b) Return
        [End of If structure]
    2.  [Move N-1 disks from peg BEG to peg AUX]
        Call TOWER (N-1, BEG, END, AUX)
    3.  Write: BEG→END
    4.  [Move N-1 disks from peg AUX to peg END]
        Call TOWER (N-1, AUX, BEG, END)
    5.  Return
```

❖ Tower (4, A, B, C)



TOWER (4,A,B,C)
TOWER (3,A,C,B)
A→C
TOWER (3,B,A,C)
TOWER (2,A,B,C)
A→B
TOWER (2,C,A,B)
A→C
TOWER (2,B,C,A)
B→C
TOWER (2,A,B,C)
TOWER (1,A,C,B)
A→C
TOWER (1,B,A,C)
A→B
TOWER (1,C,B,A)
C→B
TOWER (1,A,C,B)
A→C
TOWER (1,B,A,C)
B→A
TOWER (1,C,B,A)
B→C
TOWER (1,A,C,B)
A→C
TOWER (1,B,A,C)
A→B
A→C
B→C
A→B
C→A
C→B
A→B
A→C
B→C
B→A
C→A
B→C
A→B
A→C
B→C

❖ N-Queen

The n-queen's problem is a classic combinatorial problem in computer science and mathematics. It involves placing n queens on an n×n chessboard in such a way that no two queens threaten each other. This means that no two queens can be on the same row, the same column, or the same diagonal.

The n-queen's problem is not only a fascinating puzzle but also a useful benchmark for various algorithmic techniques in constraint satisfaction and combinatorial optimization.

Rule:

One Queen per Row: Each row must contain exactly one queen.

One Queen per Column: Each column must contain exactly one queen.

No Two Queens on the Same Diagonal:

Main Diagonals: These are diagonals that run from the top-left to the bottom-right. For two queens at positions (i, j) and (k, l), they are on the same main diagonal if |i - k| = |j - l|.

Anti-Diagonals: These are diagonals that run from the top-right to the bottom-left. For two queens at positions (i, j) and (k, l), they are on the same anti-diagonal if |i - k| = |j - l|.

নিয়ম:

প্রতি সারিতে একজন রানী: প্রতিটি সারিতে অবশ্যই একজন রাণী থাকতে হবে।

প্রতি কলামে একটি রানী: প্রতিটি কলামে অবশ্যই একটি রাণী থাকতে হবে।

একই কর্ণের উপর দুইটি রাণী থাকা যাবে না।

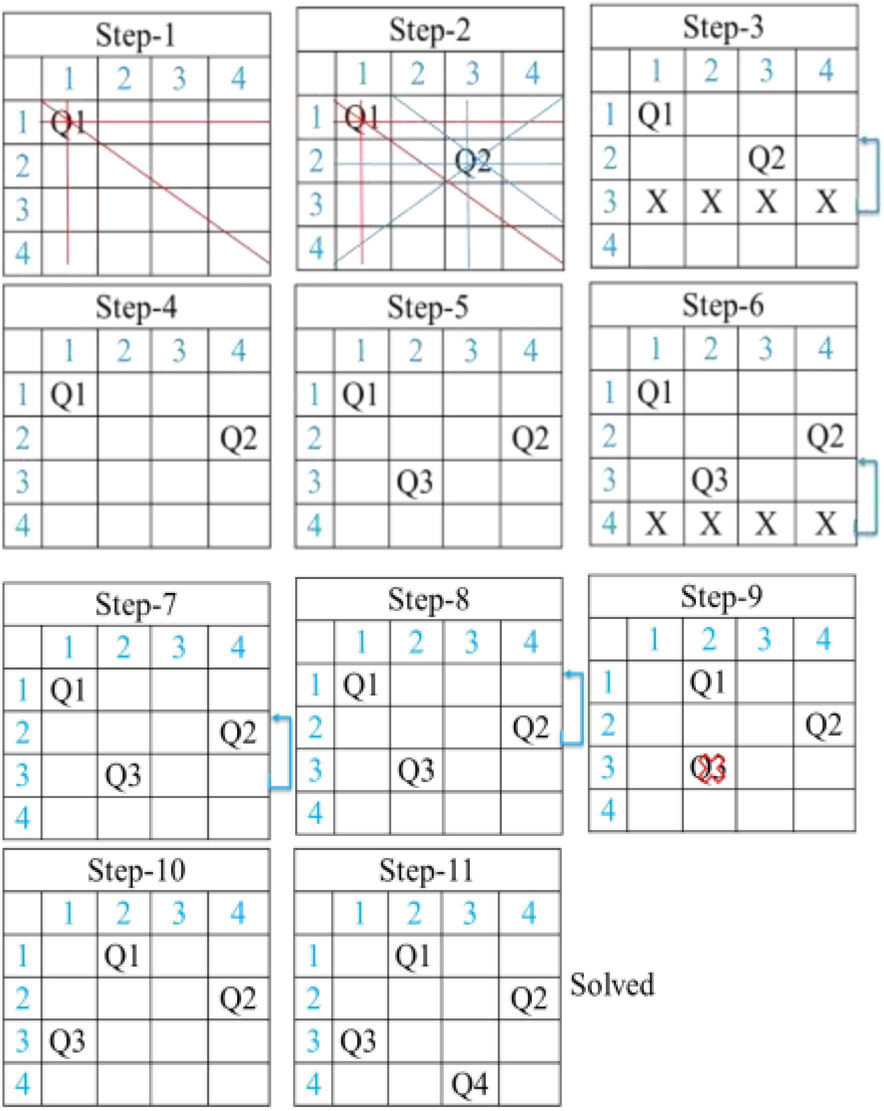
প্রধান কর্ণ: এগুলি কর্ণ যা উপরের-বাম থেকে নীচে-ডানদিকে চলে। (i, j) এবং (k, l) অবস্থানে দুটি রানীর জন্য, তারা একই প্রধান কর্ণের উপর থাকে যদি |i - k| = |j - l|

অ্যান্টি-ডায়াগোনাল: এগুলি কর্ণ যা উপরের-ডান থেকে নীচে-বাম দিকে চলে। (i, j) এবং (k, l) অবস্থানে দুটি রানীর জন্য, তারা একই অ্যান্টি-ডায়াগন্যালে থাকে যদি |i - k| = |j - l|

❖ Example 5.6

Solve 4*4 Queen problem using N-Queen algorithm

Solution:



Step-1

	1	2	3	4
1	Q1			
2				
3				
4				

Step-2

	1	2	3	4
1	Q1			
2			Q2	
3				
4				

Step-3

	1	2	3	4
1	Q1			
2			Q2	
3	X	X	X	X
4				

Step-4

	1	2	3	4
1	Q1			
2				Q2
3				
4				

Step-5

	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4				

Step-6

	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4	X	X	X	X

Step-7

	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4				

Step-8

	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4				

Step-9

	1	2	3	4
1		Q1		
2				Q2
3		~~Q3~~		
4				

Step-10

	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4				

Step-11

	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

Solved

❖ N-Queen Algorithm

```
1. set left and right diagonal counters to 0 for i= 1 to n
2. left_diagonal[i+qi] ++ right_diagonal[n-i+qi] ++
3. end
4. sum = 0
5. for i = 1 to (2n-1)
6. counter = 0
7. if (left_diagonal[i] > 1) counter += left_diagonal[i] - 1 if (right_diagonal[i] > 1)
8. counter + right_diagonal[i] - 1 sum += counter / (n-abs(i-n))
9. end
```

❖ Exercise

5.1. Write down the short notes of 0/1 Knapsack problem with algorithm.

5.2.

Items/objects:	1	2	3	4	max weight=8
Weight:	2	3	4	5	n=4
Value/Profit:	1	4	4	5	

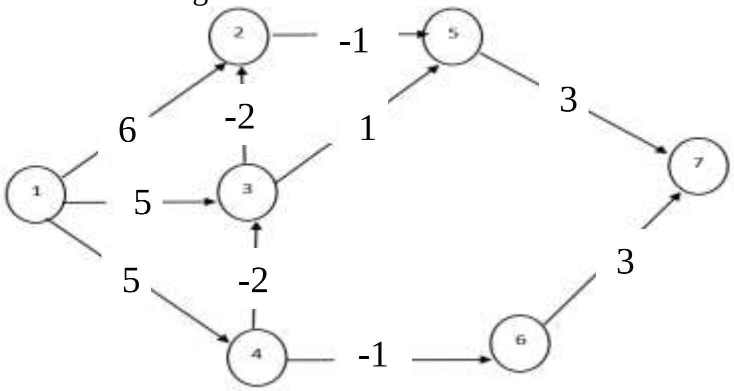
Find out the maximum profit using 0/1 Knapsack method.

5.3. State the Longest Common Subsequence algorithm.

5.4. X=BDCABA and Y=ABCBDAB Now, find the longest common subsequence.

5.5. Explain the drawbacks of Bellman Ford algorithm.

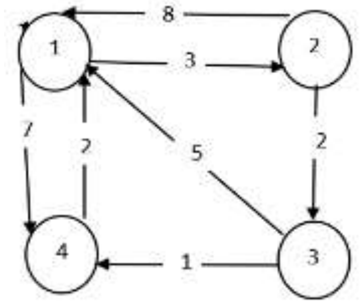
5.6. Find single source shortest path from the following graph using Bellman Ford algorithm



5.7. State the Bellman Ford algorithm.

5.8. Write down the short notes of Floyd Warshall method and state the algorithm.

5.9. Find all pair shortest path from the following graph using Floyd Warshall algorithm.



5.10. Write down the short notes of Tower of Hanoi method with algorithm.

5.11. Draw the Tower of Hanoi tree for Tower (3, A, B, C)

5.12. Solve 5*5 Queen problem using N-Queen algorithm

Previous Year Question

CCN University of Science & Technology
Department of Computer Science & Engineering
Class Test-1, Spring 2024
Course Code: CSE-117
Course Title: Algorithms Design and Analysis
Time: 30 Minutes Full Marks: 12

Set-A

1. Find out the Big-O notation of Selection sort. 4
2. Sort the array (36, 23, 47, 5, 8, 67, 11, 6, 19, 55, 22) in descending order using merge sort. 8

Set-B

1. Find out the Big-O notation of Merge sort. 4
2. Sort the array (36, 55, 23, 81, 63, 11, 51) in ascending order using bubble sort. 8

Set-C

1. Find out the Big-O notation of bubble sort. 4
2. Sort the array (71, 37, 45, 9, 78, 19, 59, 51, 17) in ascending order using selection sort. 8

Set-D

1. Find out the Big-O notation of Merge sort. 4
2. Sort the array (25, 45, 35, 15, 55) in descending order using heap sort. 8

Previous Year Question

CCN University of Science & Technology
Department of Computer Science & Engineering
Class Test-2, Spring 2024
Course Code: CSE-117
Course Title: Algorithms Design and Analysis
Time: 30 Minutes **Full Marks: 12**

Set-A

Find the maximum profit using 0/1 Knapsack Fractional algorithm from the following criteria:

Object	1	2	3	4	5	6	7	Maximum Weight 20 KG
Weight	5	3	6	4	8	6	5	
Profit	10	7	12	8	5	13	9	

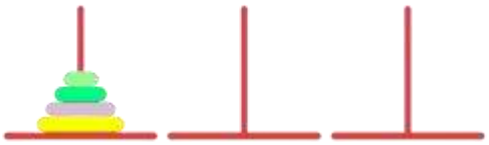
Set-B

Find the maximum profit using 0/1 Knapsack algorithm from the following criteria:

Object	1	2	3	4	5	6	7	Maximum Weight 8 KG
Weight	5	3	6	4	8	6	5	N=7
Profit	10	7	12	8	5	13	9	

Set-C

Move all disk from Peg-A to Peg-C where Peg-B is intermediate Peg.



Set-D

Find the Longest Sequence using Longest Common Subsequence algorithm from the following criteria:
X=MEHERUN
Y=KHAIRUN

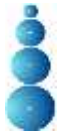
Previous Year Question

CCN University of Science & Technology
Department of Computer Science & Engineering
Mid Term Examination, Spring 2024
Course Code: CSE-117
Course Title: Algorithms Design and Analysis
Time: 1.5 Hours **Full Marks: 10×3=30**

[Answer any three Questions. Figures in the right margin indicate full marks]

1	**a**	During the lab final, "Thompson" saw that his question asked Almelo some numbers (55, 11, 22, 33) and asked him to find the number 22. He used the method to find the number by matching all the numbers one by one. Write the algorithm of the method used by "Thompson".	**5**
	b	Write down the procedure of Max Heap.	**2**
	c	Write down the Big-O notation of Merge sort.	**3**
2	**a**	During the Algorithm Lab final, "Dennis Ritchie" went to his laptop and saw that some random numbers (76, 32, 45, 13, 22) were given to him in the question and asked him to find the number 45 from there. But there were some conditions that the number should be used in such a way that the numbers are sorted in ascending order and for this he had to use the selection sort method. Arrange the numbers in ascending order using the methods discussed here and find the number.	**4+4**
	b	Write down the short notes of Time Complexity and Space Complexity.	**1+1**

3 **a** Describe the characteristics of Algorithm. **5**

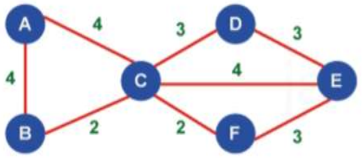
b 

DATA: 32, 51, 27, 85, 23 **5**

Sort the above DATA in ascending order using the algorithm shown in the figure.

4 **a** “James Gosling” was given some random numbers (23, 48, 37, 13). Firstly, he constructed a tree to show the numbers in descending order and he saw that when broke the tree the numbers were actually organized in descending order. **6**

Using “James Gosling” approach, sort the numbers in descending order.

b 

Find out the shortest path from this graph using Prim’s Algorithm with step by step. **4**

Previous Year Question

CCN University of Science & Technology
Department of Computer Science & Engineering
Final Examination, Spring 2024
Course Code: CSE-117
Course Title: Algorithms Design and Analysis
Time: 3.0 Hours **Full Marks: 12×5=60**

[Answer any Five Questions but the Q-1 is mandatory. Figures in the right margin indicate full marks]

1 **a** Find out the Binary Search complexity. **4**

b Write down the algorithm for Tower of Hanoi. **4**

c Sort the array [23, 42, 9, 11, 3, 33] using merge sort. **4**

2 **a** Write down the procedure of Max Heap. **4**

b Find the maximum profit using 0/1 Knapsack Fractional algorithm from the following criteria: Maximum Weight 18 KG **6**

Object	1	2	3	4	5	6	7
Weight	6	4	5	3	7	8	6
Profit	12	8	9	7	6	10	13

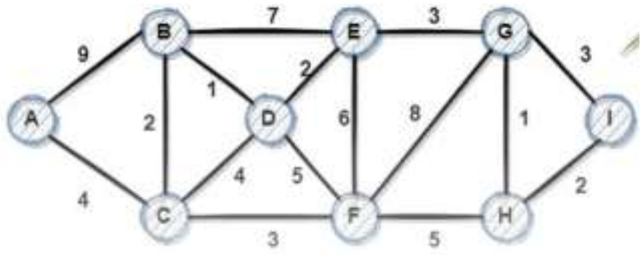
c Write down the rules of Tower of Hanoi algorithm. **2**

3 **a** Find the Longest Sequence using Longest Common Subsequence algorithm from the following criteria: **7**
X= SNAPCHAT
Y= FACEBOOK

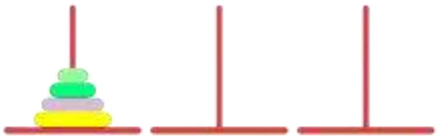
b During the lab final, "Thompson" saw that his question here some random numbers (32, 12, 44, 5, 8, 66, 11, 6, 19, 51). The examiner said arrange the number in **5**

descending order but must be used to divide and conquer technique. Now sort the number using the method used by "Thompson".

4 **a** Find minimum spanning tree using Kruskal Algorithm from the following graph- **7**

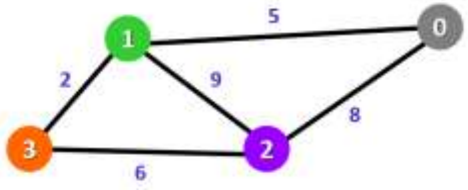


b Move all disk from Peg-A to Peg-C using Tower of Hanoi algorithm, where Peg-B is intermediate Peg. **5**



Peg-A Peg-B Peg-C

5 **a** Find all pair shortest path using Dijkstra Algorithm from the following graph- **6**

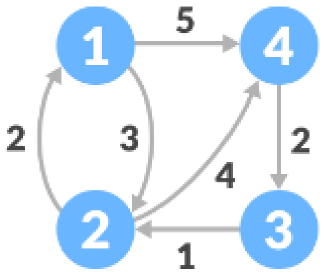


Suppose DATA is an integer array and it's have 10 elements. These are 12, 21, 28, 31, 38, 43, 56, 59, 63, 72. Please answer the question b and c from this.

b Find 57 from the following sorted element array using binary search. **4**

c Find 38 from the following sorted 12 element array using linear search. **2**

6 **a** Find D^k matrix where k is number of node using Floyd Warshall Algorithm from the following graph, where k=number of nodes. **6**



b "Thompson" was given some random numbers (7, 18, 10, 24, 15). Firstly, he constructed a tree to show the numbers in ascending order and he saw that when broke the tree the numbers were actually organized in ascending order.
Using "James Gosling" approach, sort the numbers in ascending order. **4**

c Write down the rules of N-Queen problem. **2**

7 **a** "James Gosling" was given some random numbers (33, 44, 55, 22, 66). Firstly, he constructed a tree to show the numbers in descending order and he saw that when broke the tree the numbers were actually organized in descending order.
Using "James Gosling" approach, sort the numbers in descending order. **6**

b Suppose Q1, Q2, Q3, Q4, Q5 are 5 queens now place the all queens in appropriate location using N-Queens algorithm. But, ensure that one of the queens are not conflicted another queens in Vertically, Horizontally and Diagonally. Please use the graph table which is attached in question. **6**

For question 6(C)

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Step-

	1	2	3	4	5
1					
2					
3					
4					
5					

Previous Year Question

CCN University of Science & Technology
Department of Computer Science & Engineering
Lab Final Quiz, Spring 2024
Course Code: CSE-118
Course Title: Algorithms Design and Analysis Sessional
Time: 15 Minutes **Full Marks: 12**

1. In Binary Search SEARCH<DATA[MID] than
a) START=MID+1 b) END=MID+1
c) START=MID-1 d) END=MID-1

2. In Max Heap the 2nd step is
a) Assign new value to the node b) Create a new node
c) Compare with child node d) Swap child and root

3. In Bubble sort algorithm the Big-O notation is-
a) O(nlogn) b) O(n)c) O(n^2) d) O(logn)

4. In Prims algorithm the Big-O notation is-
a) O(ElogV) b) O(ElogE) c) O(VlogV) d) O(VlogE)

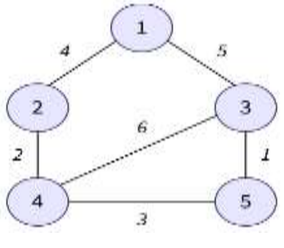
5. In Prims algorithm used for find out the-
a) Minimum path b) Pair of path
c) Maximum path d) Distance table

9.

Object	1	2	3	4
Profit	7	9	11	6
Weight	2	9	4	3

If the maximum weight is 7 then find maximum profit using 0/1 Knapsack Fractional-
a) 21 b) 20 c) 18 d) 19

10.



Find minimum distance using Prims algorithm-
a) 19 b) 18 c) 13 d) 15

6. Which algorithm is not suitable for sorting the large number of dataset?
a) Binary b) Linear c) Heap d) Bubble

7. In Linear Search algorithm the Big-O notation is-
a) O(nlogn) b) O(n)c) O(n^2) d) O(logn)

8. In Bubble sort number of maximum iteration is-
a) 2^n-1 b) 2^n+1 c) n-1 d) n