Ques 1 :

a). Worst case of quick sort will arise if smallest or largest element is choosen as pivot.

Probability of selecting smallest element : $\frac{1}{50}$

Probability of selecting largest element : $\frac{1}{50}$

Probability of worst case : $\frac{1}{50} + \frac{1}{50}$: $\frac{2}{50}$ = 0.04

b).

Sieve of Eratosthenes

```
int *arr = new int[n+1] ;

// place true at all places
for(int i = 0 ; i < n ; i++)
    arr[i] = true ;

arr[0] = false ;
arr[1] = false ;

for(int table = 2 ; table * table <= n ; table++)
{
    if(arr[table] == false)
        continue ;

    for(int multiplier = table ; table * multiplier <= n ; multiplier++)
    {
        arr[table * multiplier] = false ;
    }
}

for(int i = 0 ; i <= n ; i++)
{
    if(arr[i])
        cout << i << "\t" ;
}
cout << endl ;
```

Time Complexity : $O(n \log \log n)$

Space Complexity : $O(n)$

OR

```
for(int num = 2 ; num <= n ; num++)
{
    int flag = 0 ;
    for(int i = 2 ; i <= sqrt(num) ; i++)
    {
        if(num % i == 0)
        {
            flag = 1 ;
            break ;
        }
    }

    if(flag == 0)
        cout << num << endl ;
}
```

Time Complexity : $O(n\sqrt{n})$

Space Complexity : $O(1)$

c).

```
int i,j,k=0 ;
for (i=n/2 ; i≤n; i++)              ——— n times
    for (j=2 ; j<=n ; j=j*2)        ——— logn times
        k=k+n/2 ;
return k
```

Statement is executed nlogn times

$\frac{n}{2}$ is added nlogn times

Final value of k= $\frac{n}{2} \times n\log n$

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

d). Pruning improves the performance by eliminating the candidate solutions that will not lead to an optimal solution.

e).

no. of vertices in MST = 100
no. of edges in MST = 99

older weight of MST = 500
new weight of each edge is increased by 5.

new weight of MST = 500 + 99*5 = 995

Ques 2 :

a).

```
bool isItSafeToPlaceQueen(bool **board, int row,
                          int col, int n)
{
    // vertically up
    int r = row - 1;
    int c = col ;
    while(r >= 0)
    {
        if(board[r][c] == true)
            return false ;

        r-- ;
    }

    // horizontally left
    r = row ;
    c = col - 1 ;
    while(c >= 0)
    {
        if(board[r][c] == true)
            return false ;

        c-- ;
    }

    // diagonally left
    r = row - 1;
    c = col - 1 ;
    while(r >= 0 && c >= 0)
    {
        if(board[r][c] == true)
            return false ;

        r-- ;
        c-- ;
    }

    // diagonally right
    r = row - 1;
    c = col + 1 ;
    while(r >= 0 && c < n)
    {
        if(board[r][c] == true)
            return false ;

        r-- ;
        c++ ;
    }

    return true ;
}
```

```
void queen(bool **board, int row, int n, string ans)
{
    if(row == n)
    {
        cout << ans << endl ;
        return ;
    }

    if(row == n)
        return ;

    for(int col = 0 ; col < n ; col++)
    {
        if(isItSafeToPlaceQueen(board,row,col,n))
        {
            board[row][col] = true ;
            queen(board, row+1, n, ans + "{" + to_string(row)
                                 + "," + to_string(col) + "}") ;
            board[row][col] = false ;
        }
    }
}

int main()
{
    int n = 4 ;
    bool **board = new bool*[n] ;

    for(int i = 0 ; i < n ; i++)
    {
        board[i] = new bool[n] ;

        for(int j=0 ; j < n ; j++)
            board[i][j] = false ;
    }

    queen(board, 0, n, "") ;

    return 0 ;

}
```

b). optimised Solution :

```
class Pair
{
    public:

    int data;
    int array_num;
    int idx_num;

    Pair(int data, int array_num, int idx_num)
    {
        this->data = data ;
        this->array_num = array_num ;
        this->idx_num = idx_num ;
    }
};

struct Comp{
    bool operator()(const Pair& a, const Pair& b)
    {
        return a.data > b.data ;
    }
};
```

Time Complexity: $O(nk \log k)$

Space Complexity: $O(k)$

```
int main()
{
    int n = 4 ; // no. of elements in each array
    int k = 3 ; // no. of arrays

    int arr[][4] = {{1,3,5,7},{2,4,6,8},{0,9,10,11}} ;

    vector<int> ans ;
    priority_queue< Pair, vector<Pair>, Comp> heap;  // create a min heap

    // add 0th index element from each array in min heap
    for (int i = 0; i < k; i++)
    {
        Pair new_pair(arr[i][0], i, 0) ;
        heap.push(new_pair);
    }

    while (!heap.empty())
    {
        Pair rp = heap.top(); // rp = removed_pair, remove the element with minimum value
        heap.pop() ;
        ans.push_back(rp.data); // add the minimum element in ans vector

        // edit the removed pair and add in heap again
        rp.idx_num ++;

        if (rp.idx_num < n)
        {
            rp.data = arr[rp.array_num][rp.idx_num];
            heap.push(rp);
        }
    }

    for(int i = 0 ; i < ans.size() ; i++)
        cout << ans[i] << endl ;

    return 0 ;
}
```

(If student has written any other algorithms like merging two lists recursively then grade accordingly)

Ques 3:

a).

Rod Cutting Problem:

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Storage	1	5	8	10	13	17	18	22

Length 1: Sell as it is, profit = 1

Length 2: Sell as it is, profit = 5
Sell as (1,1), profit = 2
} 5 is better

Length 3: Sell as it is, profit = 8
Sell as (1,2), profit = $\underline{1+5}$ = 6
from storage
} 8 is better

Length 4: Sell as it is, profit = 9
Sell as (1,3), profit = 1+8 = 9
Sell as (2,2), profit = 5+5 = 10
} 10 is better

Length 5: Sell as it is, profit = 10
Sell as (1,4), profit = 1+10 = 11
Sell as (2,3), profit = 5+8 = 13
} 13 is better

Length 6: Sell as it is, profit = 17
Sell as (1,5), profit = 1+13 = 14
Sell as (2,4), profit = 5+10 = 15
Sell as (3,3), profit = 8+8 = 16
} 17 is better

Length 7: Sell as it is, profit = 17
Sell as (1,6), profit = 1+17 = 18
Sell as (2,5), profit = 5+13 = 18
Sell as (3,4), profit = 8+10 = 18
} 18 is better

length 8:

Sell as it is, profit : 20
Sell as (1,7), profit = 1+18=19
Sell as (2,6), profit = 5+17=22
Sell as (3,5), profit = 8+13=21
Sell as (4,4), profit = 10+10=20

22 is better

Max Profit = 22

length 8

length 2 → as it is sell

length 6 → as it is sell

b).

40X20 20X30 30X10 10X30

A B C D

no. of matrix operations required for multiplying 2 matrices:

A·B = 40X20X30 = 24000
B·C = 20X30X10 = 6000
C·D = 30X10X30 = 9000

no. of matrix operations required for multiplying 3 matrices:

A·B·C

ABC

(A)(BC)
0 6000
40x20 20x10
40x20x10 = 8000

(AB)(C)
24000 0
40x30 30x10
40x30x10 = 12000

0+6000+8000
= 14000

24000+0+12000
= 36000

minimum multiplications needed for multiplying ABC = 14000

B·C·D

BCD

(B)(CD)

0 9000

20x30 30x30

20x30x30 = 18000

(BC)(D)

6000 0

20x10 10x30

20x10x30 = 6000

0+9000+18000 = 27000

6000+0+6000 = 12000

minimum multiplications needed for multiplying BCD = 12000

ABCD

ABCD

(A)(BCD)

0 12000

40x20 20x30

40x20x30 = 24000

(AB)(CD)

24000 9000

40x30 30x30

40x30x30 = 36000

(ABC)(D)

14000 0

40x10 10x30

40x10x30 = 12000

0+12000+24000 = 36000

24000+9000+36000 = 69000

14000+0+12000 26000

minimum multiplications needed for multiplying ABCD = 26000

Ques 4:

a).

Jobs	J1	J2	J3	J4	J5	J6	J7
Profits	3	5	20	18	1	6	30
Deadlines	1	3	4	3	2	1	2

- Sort the jobs in decreasing order of profit

Jobs	J7	J3	J4	J6	J2	J1	J5
Profits	30	20	18	6	5	3	1
Deadlines	2	4	3	1	3	1	2

- Iterate over the jobs and assign the last slot available

Profit

0 1 2 3 4 5 6 7

0

J7:

	J7					

0 1 2 3 4 5 6 7

30

J3 :

	J7		J3			

0 1 2 3 4 5 6 7

30+20

J4 :

	J7	J4	J3			

0 1 2 3 4 5 6 7

30+20+18

J6 :

J6	J7	J4	J3			

0 1 2 3 4 5 6 7

30+20+18+6

J2, J1, J5 cannot be completed because deadlines are 3, 1, 2 respectively and all slots are occupied till 3.

Profit = 74

b).

$n = 10$

relax every edge 9 times

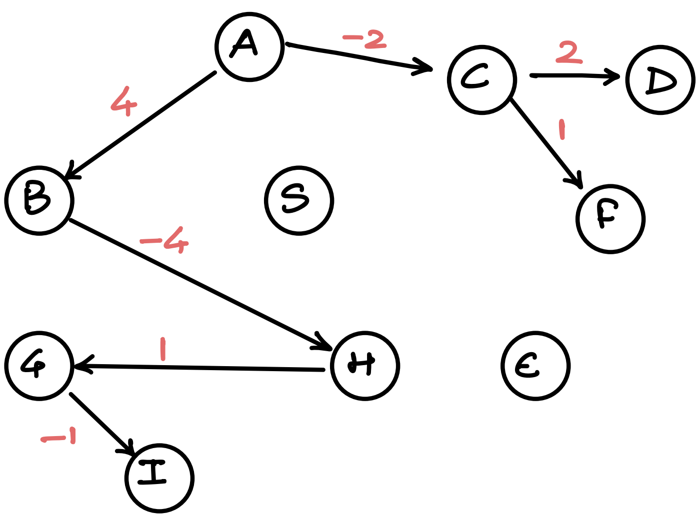
Edges

$A \rightarrow C$: -2
$A \rightarrow B$: 4
$C \rightarrow D$: 2
$C \rightarrow F$: 1
$S \rightarrow A$: 7
$S \rightarrow C$: 6
$S \rightarrow F$: 5
$S \rightarrow E$: 6
$E \rightarrow F$: -2
$E \rightarrow H$: 3
$B \rightarrow G$: -2
$B \rightarrow H$: -4
$H \rightarrow G$: 1
$G \rightarrow I$: -1
$I \rightarrow H$: 1
$F \rightarrow D$: 3

Cost

	Initial	Relax 1^{st}	Relax 2^{nd}
A	$\rightarrow 0$	0	0
B	$\rightarrow \infty$	4	4
C	$\rightarrow \infty$	-2	-2
D	$\rightarrow \infty$	0	0
E	$\rightarrow \infty$	∞	∞
F	$\rightarrow \infty$	-1	-1
G	$\rightarrow \infty$	1	1
H	$\rightarrow \infty$	0	0
I	$\rightarrow \infty$	0	0
S	$\rightarrow \infty$	∞	∞

No change in weight after relaxing 2^{nd} time. stop.



Ques 5: a).

i).

```cpp
int main()
{
    TOH(3, "S", "D", "H") ;
    return 0 ;
}

void TOH(int n, string src, string dst, string helper)
{
    if(n == 0)
        return ;

    TOH(n-1, src, helper, dst) ;
    cout << "Move disc " << n << " from " << src << " to " << dst << endl ;
    TOH(n-1, helper, dst, src) ;
}
```

ii).

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

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

$$\vdots$$

$$T(1) = 1$$

$$\Downarrow$$

$$T(n) = 2\cancel{T(n-1)} + 1$$

$$\cancel{2T(n-1)} = \cancel{2^2T(n-2)} + 2$$

$$\cancel{2^2T(n-2)} = \cancel{2^3T(n-3)} + 2^2$$

$$\vdots$$

$$\cancel{2^{n-1}\,T(n-(n-1))} = 2^{n-1}$$

$$T(n) = 1 + 2 + 2^2 + \cdots\cdots + 2^{n-1}$$

$$T(n) = 1\left(\frac{2^n - 1}{2-1}\right) = 2^n - 1$$

$$T(n) = O(2^n)$$

iii). no of moves required : $2^n - 1$

if $n = 3$ then moves = 7

if $n = 4$ then moves = 15

b).

```
int delete()
{
    swap(arr[0], arr[N-1]) ;
    int rv = arr[N-1] ;
    N-- ;
    downheapify(0) ;

    return rv ;
}

void downheapify(int pi)
{
    int lci = 2*pi+1 ;
    int rci = 2*pi+2 ;

    int mini = pi ;

    if(lci < N && arr[mini] > arr[lci])
        mini = lci ;

    if(rci < N && arr[mini] > arr[rci])
        mini = rci ;

    if(mini != pi)
    {
        swap(arr[mini], arr[pi]) ;
        downheapify(mini) ;
    }

}
```