Q1. DPBIT #1

A shop sells 20 distinct items numbered 1 to 20 and customer may either buy 0 or 1 of each of the items. Which of the following is possible and the most space efficient way to represent the shopping cart of the customer?

An array of 20 integers where ith element is 1 if ith item is in shopping cart else 0.

An array of 20 booleans where ith element is 1 if ith item is in shopping cart else 0.

A 32 bit integer whose bits are used to represent the shopping cart

A 64 bit integer whose bits are used to represent the shopping cart

Q2. DPBIT #2

A shop sells 20 distinct items numbered 1 to 20 and customer may either buy 0 or 1 of each of the items.

We have decided to use int cart(initially = 0) to represent our shopping cart.

Select correct operations for the following :

```
    Customer keeps ith item into the cart.
    customer removes the ith item from the cart.
```

cart = (cart | (1 << i)), cart = (cart & (~(1 << i)))

cart = (cart | i), cart = (cart & (~(1 << i)))

cart = (cart & (1 << i)), cart = (cart & ((1 << i)))

cart = (cart ^ (1 << 2*i)), cart = (cart & ((1 << i)))

Q3. DPBIT #3

There are N persons and N tasks, each task is to be alloted to a single person. We are also given a matrix cost of size NxN, where cost[i][j] denotes, how much person i is going to charge for task j.

Now we need to assign each task to a person in such a way that the total cost is minimum.

Let DP[i][mask] denote the minimum cost to assign the people numbered from i to N with the tasks represented by the mask. If the jth bit of the mask is off then the jth task still needs to be assigned to a person.

Select the correct recurrence:

DP[i][mask] = minimum of (DP[i+1][mask with kth bit set] + cost(i,k)) for all k where kth bit of mask is off.

DP[i][mask] = minimum of (DP[i+1][mask + 1] + cost(i,k)) for all k where kth bit of mask is off.

DP[i][mask] = minimum of (DP[i+1][mask with kth bit set] + cost(i,k)) for all k where kth bit of mask is on.

DP[i][mask] = minimum of (DP[i+1][mask + 1] + cost(i,k)) for all k where kth bit of mask is on.

Q4. DPBIT #4

There are N persons and N tasks, each task is to be alloted to a single person. We are also given a matrix cost of size NxN, where cost[i][j] denotes, how much person i is going to charge for task j. Now we need to assign each task to a person in such a way that the total cost is minimum.

Notice that in the DP state DP[i][mask] as described in the previous problem, i is always equal to the number of set bits in the mask + 1. (both tasks and people are numbered from 1 to N).

We feel that we can describe a 1-D DP[mask] instead of the two-dimensional DP[i][mask].

For top down DP solutions using the two formulations.

Choose the correct option :

both have the same time complexity which is O(Nx2^N).

space complexity for 2D formulation is O(Nx2^N) while for the 1D formulation is O(2^N).

Many of the states in the 2D formulation will never be reached in the top down DP approach.

All of the above.

Q5. DPBIT #5

In reference to the video : T-Shirts Codechef, DP + Bitmasks.

```
Brute Force Algorithm:
    Number the people from 1 to N.
    Generate all possible tuples (s1, s2, s3, ..., sn).
    Here si is a shirt owned by ith person.
    If for a given tuple no two numbers in the tuple are same then add
1 to answer.
```

What is the time complexity for this approach? Let S be the max number of shirts that can be owned by a person.

O(S^2)

O(S^N)

O(S!)

O(NxS)

Q6. DPBIT #6

In reference to the video : T-Shirts Codechef, DP + Bitmasks.

Let us try to solve the problem by defining :

```
MASK : an integer with N bits and ith bit is off, if ith person is yet
to be assigned a shirt.
SHIRT : the maximum number of shirts(100 in this question)
PEOPLE[N][SHIRT] : PEOPLE[i][j] is 1, if ith person owns jth shirt else
0.
DP[SHIRT][MASK] : DP[i][x] is the number of ways to assign shirts from
shirt number i to shirt number SHIRT to the people given by mask x.
```

choose the correct recurrence:

DP[i][x] = summation of { DP[i + 1][x with kth bit set] } over all k = 1 to N given that kth bit of x is off and PEOPLE[k][i] = 1.

DP[i][x] = maximum of { DP[i + 1][x with kth bit set] } over all k = 1 to N given that kth bit of x is off and PEOPLE[k][i] = 1.

DP[i][x] = product of { DP[i + 1][x with kth bit set] } over all k = 1 to N given that kth bit of x is off and PEOPLE[k][i] = 1.

None of these

Q7. DPBIT #7

Choose the correct time complexities for solving the Travelling Salesman Problem(TSP) using brute force and dp solutions respectively :

O(4^N) and O(N^4)

O(N!) and O(N^2 x 2^N)

O(N!) and O(N x 2^N)

O(4^N) and O(N^2 x 2^N)

Q8. DPBIT #8

Let us solve TSP using dynamic programming :

```
Define DP[i][mask] : minimum cost to complete the cycle if we are
currently on city i and we have already been to the cities represented
by the mask.
If the jth bit of the mask is off then we are yet to visit city j.
Let COST[i][j] represent cost of moving from city i to city j.
```

Choose the correct recurrence:

None of these.

DP[i][mask] = minimum of { DP[j][mask + i] + cost[i][j] } for all j =1 to N.

DP[i][mask] = minimum of { DP[j][mask with j bit off] + cost[i][j] } for all j = 1 to N given that j bit of mask is on.

DP[i][mask] = minimum of { DP[j][mask with j bit set] + cost[i][j] } for all j = 1 to N given that j bit of mask is off.

Q9. DPBIT #9

In reference to DPBit - Mahmoud and Ehab - Codeforces 959F.

Let the array be A1, A2, .., AN.

Define DP[i][j] : number of subsequences of the subarray[1..i] with xor-sum equal to j.

Choose the correct brute force DP solution :

DP[i][j] = 2xA[i]xDP[i - 1][j - 1]

DP[i][j] = DP[i - 1][j] + DP[i - 1][j xor A[i]]

DP[i][j] = A[i]xDP[i - 1][j - 1] + DP[i - 1][j xor A[i]]

DP[i][j] = DP[i][A[i] xor j]

Q10. DPBIT #10

In reference to DPBit - Mahmoud and Ehab - Codeforces 959F.

what is the time complexity of the brute force dp solution and the optimal solution described in the video respectively :

Here b is the maximum number of bits in any number of the array.

O(N) and O(N)

O(N^2) and O(N)

```
O(2^b x N) and O(N^b)

O(2^b x N) and O((N + 2^b)*log(2^b))
```