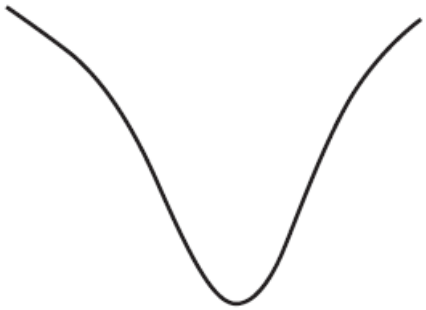
ASSIGNMENT – III

SECTION - A

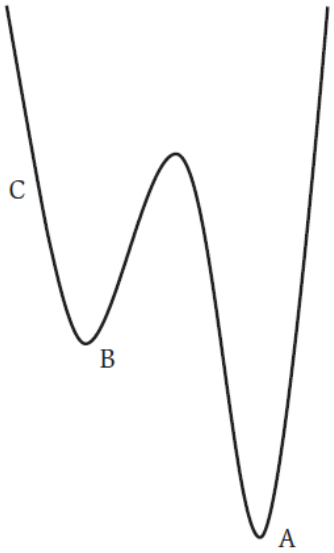
Q 1.

a) It starts with the full vertex set *V*, and keeps deleting nodes until there are none left. Indeed, the set of vertex covers for this edge-less graph corresponds naturally to the funnel we drew in given below fig.

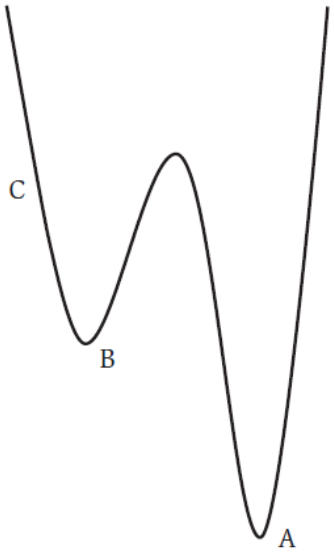


The unique local minimum is the global minimum, and there is a downhill path to it from any point.

b) The minimum vertex cover for *G* is the singleton set {*x*1}, and gradient descent can reach this solution by successively deleting *y*1, . . . , *yn*−1 in any order. So, {x1} will be our global minimum. But, if gradient descent deletes the node *x*1 first, then it is immediately stuck: No node *yi* can be deleted without destroying the vertex cover property. Thus the algorithm has become trapped in the local minimum {*y*1, *y*2, . . . , *yn*−1}, which has very high cost relative to the global minimum.



c) In this question there are one global minima and one local minima. Here the global minima is {x1, x2, … , xk} and the local minima is {y1, y2, …, yl} since k<l, we can achieve local minima and global minima by deleting nodes from xi and yi respectively. For this we use gradient descent method on any of the two sets.



Q2. In a general energy landscape, there may be a very large number of local minima that make it hard to find the global minimum. When a system trapped in a local minimum it requires a certain amount of energy to come out of that local minimum. If the system doesn't have the required energy, it can never reach to the global minimum. For ex: in the jagged funnel there are local minima waiting to trap the system all the way along its journey to the bottom.

Q3. State-Flipping Algorithm: -

While the current configuration is not stable
    There must be an unsatisfied node
    Choose an unsatisfied node *u*
    Flip the state of *u*
Endwhile

This algorithm is used to covert to convert any unstable configuration to a stable configuration. When the algorithm terminates, we will have a stable configuration. This algorithm found stable configuration in time polynomial in n and W = $\sum_e |we|$. Here w_e is the weight of the edge e.

Q4. True. Simulated Annealing is inspired by the physical process of annealing in metallurgy, where a material is heated and gradually cooled to reduce its defects and reach a low-energy state. In the context of optimization problems, Simulated Annealing explores the solution space by iteratively perturbing the current solution and accepting new solutions, even if they are worse (higher energy) than the current solution. This behavior enables the algorithm to escape local optima, which are lower-energy states that are not the global minimum.

Q6. Consider three nodes *a*, *b*, *c* all mutually connected to one another by edges of weight 1. Then, no matter what configuration we choose, two of these nodes will have the same state and thus will be violating the requirement that they have opposite states.

Q7. If all nodes have the same state, they will all be unsatisfied; and if one node has a different state from the other two, then the node directly in front of it will be unsatisfied. Thus, there is no configuration of this directed network in which all nodes are satisfied.

SECTION – B

Q1. One view is to consider the world as behaving randomly: One can consider traditional algorithms that confront randomly generated input. This approach is often termed *average-case analysis*, since we are studying the behaviour of an algorithm on an "average" input (subject to some underlying random process), rather than a worst-case input. A second view is to consider algorithms that behave randomly: The world provides the same worst-case input as always, but we allow our algorithm to make random decisions as it processes the input.

Q6.

a) True. In this algorithm we choose an element $ai \in S$, the "splitter," and form the sets $S- = \{aj : aj < ai\}$ and $S+ = \{aj : aj > ai\}$. We can then

determine which of $S-$ or $S+$ contains the kth largest element, and iterate only on this one. This algorithm is always called recursively on a strictly smaller set, so it must terminate and the termination condition is $|S^-| = K - 1$. Finally, from the choice of which recursive call to make, it's clear by induction that this algorithm will always return the kth largest element of S.

b) The running time $T(n)$ would be bounded by the recurrence

$T(n) \leq T(n/2) + cn$

$T(n) = O(n)$

c) Here the size of the sets in the recursive call is shrink by a factor of at least $(1- \varepsilon)$ each time. Thus, the recursion relation is

$T(n) \leq T((1- \varepsilon)n) + cn$

$T(n) = O(n)$

Q8. The principal criteria to choose a good hash function is to develop a hashing scheme where we can prove that it results in efficient dictionary operations with high probability. To need to implements randomization in the dictionary data structure. For every element $u \in U$, when we go to insert u into S, we select a value $h(u)$ uniformly at random in the set $\{0, 1, \ldots, n - 1\}$, independently of all previous choices. In this case, the probability that two randomly selected values $h(u)$ and $h(v)$ are equal (and hence cause a collision) is quite small.