Week-8

ign an algorithm which will find maximum cost required connect cities (Prim's Algorithm).

orithm :→

```
1) START
2) Input n
3) if i>=n goto step 8
4) if j>=n goto step 7
5) input a[i][j]
6) if j<n goto step 4
7) if i<n goto step 3
8) prims (a,n)
9) STOP
```

```
prims (int ** arr, int n) {
 vector<bool> visited (n, false);
 vector <int> weight (n, INF);
 priority_queue <pair <int, int >>, vector <pair<int, int>>,
     greater <pair <int, int>> min_heap;
 int src = 0;
  weight [src] = 0;
  min_heap.push (make_pair (weight [src], src));
  while (! min_heap.empty ()) {
    u = min_heap.top().second;
    min_heap.pop();
    if (!visited [u]) {
      visited [u] = True;
   for (int v=0; v<u ; ++v) {
    if (!visited [v] && arr[u][v] ! = 0 && arr [u][v] <
                                            weight [v])
     {
       weight [v] = arr[u][v];
       min_heap.push (make_pair (weight [v], v));
     }
   }
}
int sum = 0;
for (auto i : weight )
    sum + = i;
   return sum;
}
```

on minimum cost required
nect cities (Kruskal Algorithm)

```
ithm :→ 1) START
        2) Input n
        3) if i>=n goto step 8
        4) if j>=n goto step 7
        5) input graph[i][j]
        6) if j<n goto step 4
        7) if i<n goto step 5
        8) Kruskals (graph, n)
        9) Stop

Kruskals (graph, n) {
vector <pair <int , pair <int , int>>> G ;
for (i=0 ; i<n ; i++)
 for (j=0 ; j<n ; j++)
   if (graph[i][j] != 0)
     G.push-back (make-pair (graph [i][j] , make-pair (i,j)));
     sort (G.begin() , G.end()) ;
     vector <int> parent (n, NIL) ;
       s=0;
      for (auto i : G) {
       u = i.second.first;
       v = i.second.second;
       w = i.first;
      if (UnionByWeight (parent , u,v))
              s+ = w;
       }
         return s;
    }
```

③ Design an algorithm ... for a project maximum budget required

Algorithm :→
```
1) START
2) input n
3) if i>=n goto step 8
4) if j>=n goto step 7
5) input graph[i][j]
6) if j<n goto step 4
7) if i<n goto step 3
8) Kruskals (graph, n)
9) STOP
```

```
Kruskals(graph, n) {
  for (i=0; i<n; i++)
    for (j=0 ; j<n ; j++)
      if (graph[i][j]! = 0)
        G.push-back (make-pair (graph [i][j] , make-pa
    sort (G.begin(), G.end(), greater<pair <int , pair

    vector.<int> parent (n, NIL);
        s = 0;
    for (auto i : G) {
        u = i.second.first ;
        v = i.second.second;
        w = i.first;
    if ( UnionByWeight (parent , u, v) )
          s+ = w;
      }
      return s;
  }
```