L1: Algorithm 1: Insertion Sort #include <bits/stdc++.h>
using namespace std;void Insertion_sort(int arr[], int n){int k, shift = 0, comp = 1, j;for (int i = 1; i < n; i++)
{k = arr[i];j = (i - 1);// count no. of shifting and comparisonwhile (j >= 0 && arr[j] > k)
{arr[j + 1] = arr[j];j = j - 1;comp++;shift++;}arr[j + 1] = k;if (k > arr[j])comp++; // count number of comparison
when while loop = false}for (int i = 0; i < n; i++){cout << arr[i] << " ";}cout << endl;
cout << "comp: " << comp << endl;cout << "shift: " << shift << endl;}int main(){int n;cin >> n;int arr[n];for (int i =
0; i < n; i++){cin >> arr[i];}Insertion_sort(arr, n);return 0;}

Algorithm 2: Selection Sort #include <bits/stdc++.h>using namespace std;void Selection_sort(int arr[], int n){
int min, swap = 0, comp = 0;for (int i = 0; i < (n - 1); i++){min = i;// compare arr[min] with all other remaining
array element for (int j = i + 1; j < n; j++){comp++;if (arr[j] < arr[min]){min = arr[j];
swap++;}} // swapint temp = arr[min];arr[min] = arr[i];arr[i] = temp;}for (int i = 0; i < n; i++){cout << arr[i] << " ";}
cout << endl;cout << "comp: " << comp << endl;cout << "swap: " << swap << endl;}
int main(){int n;cin >> n;int arr[n];for (int i = 0; i < n; i++){cin >> arr[i];}Selection_sort(arr, n);return 0;}

L2:Algorithm 1: Quick Sort #include <bits/stdc++.h> using namespace std; int comparisons = 0, swap1 = 0;
void swap_to_num(int *n1, int *n2) { int temp = *n1; *n1 = *n2; *n2 = temp; swap1++; } int partition(int*A, int l,
int h) { int pivot = A[l]; int i = l; int j = h; while(i < j) { while(A[i] <= pivot) { comparisons++; i++; } while(A[j] > pivot)
{ comparisons++; j--; } if(i < j) { swap_to_num(&A[i], &A[j]); // swap1++; } } swap_to_num(&A[l], &A[j]); //
swap1++; return j; } void Quick_sort(int *A, int l, int h) { if(l < h) { // p = partition point int p = partition(A, l, h);
Quick_sort(A, l, p-1); Quick_sort(A, p+1, h); } } int main() { int n; cout << "Enter your array size: "; cin >> n;int
A[n]; for(int i = 0; i < n; i++) cin >> A[i]; //call merge sort function Quick_sort(A, 0, n-1); cout << "No of Swaps:
" << swap1 << endl; cout << "No of comparisons: " << comparisons << endl; cout << "Sorted array: ";
for(int i = 0; i < n; i++) cout << A[i] << " "; return 0; }

Algorithm 2: Merge Sort #include<bits/stdc++.h>using namespace std; int Comparisons = 0, shift = 0;
int merge(int *A, int p, int mid, int r) { int i = p, k = 0; int j = mid + 1; int B[r-p+1]; while(i <= mid && j <= r)
{ Comparisons++; if(A[i] < A[j]) { B[k] = A[i]; i++; } else { B[k] = A[j]; j++; }k++; } while(i <= mid) {
Comparisons++; B[k] = A[i]; i++; k++; } while(j <= r) { Comparisons++; B[k] = A[j]; j++; k++; } for(int w = p;
w <= r; w++) { A[w] = B[w-p]; } return(Comparisons); } int merge_sort(int *A, int p, int r) { int mid, count = 0;
if(p < r) { mid = ((p + r)/2); // divide operation merge_sort(A, p, mid); // Recursive call
merge_sort(A, (mid + 1), r); count = merge(A, p, mid, r); // Conquer element
} return count; } int main() { int n; cout << "Enter your array size: "; cin >> n;
int A[n]; for(int i = 0; i < n; i++) cin >> A[i]; //call merge sort function int count = merge_sort(A, 0, n-1);
cout << "No of Swaps/Shifts: " << shift << endl; // no swapping and shifting are present
cout << "No of Comparisons: " << count << endl; cout << "Sorted array: "; for(int i = 0; i < n; i++) cout << A[i]
<< " "; return 0; }

Algorithm 3: Study and Implementation of Maximum sum sub-array problem #include <bits/stdc++.h>
using namespace std; //divide and conquer method cross sum array here using this method we not get
right ans. because our max subarray{5,-3,6} is divide int d1(int a[], int p, int r) {
int sum1 = 0, max1 = 0; for(int i = r; i >= p; i--) { sum1 += a[i]; max1 = max(sum1,max1); }
return max1; } int d2(int a[], int p, int r) { int sum2 = 0, max2 = 0; for(int i = p; i <= r; i++) { sum2 += a[i];
max2 = max(sum2, max2); } return max2; } int sum(int a[], int p, int r) { // p = lower bound & r = higher
bound // In this merge sort logic apply int max_l, max_r, q, maxMid_l, maxMid_r, maxMid, ans; // q = middle
point if(p == r) return a[p]; //find middle value q = floor((p + r)/2); max_l = sum(a,p,q); max_r = sum(a,q+1,r);
maxMid_l = d1(a,p,q); maxMid_r = d2(a,q+1,r); maxMid = (maxMid_l + maxMid_r); // cout << s; int ans1 =
max(max_l,max_r); ans = max(ans1, maxMid); return maxMid; } int main() { int n; cin >> n; int a[n]; // array
name = a for(int i = 0; i < n; i++) cin >> a[i]; int s = sum(a,0,n-1); // sum function call cout << "Ans: " << s <<
endl; return 0; }

L3:1. Fractional Knapsack problem // Name: Hina Jadav // ID: 21CEUBG046 // Roll_no: CE052 // Date:
27/12/"22 #include <bits/stdc++.h> using namespace std; struct knapsack { int value; double weight; double
ratio; // (value/weight) }; int main() { cout << "Enter no. of object: "; int n; cin >> n; struct knapsack obj[n];
for(int i = 0; i < n; i++) { cout << "Enter value & weight of item: "; int x; double y; cin >> x; obj[i].value = x; cin >> y;
obj[i].weight = y; obj[i].ratio = ((double)obj[i].value / obj[i].weight); } //sort knapsack array for(int i = 0; i < n;
i++) { for(int j = i+1; j < n; j++) { if(obj[i].ratio < obj[j].ratio) { struct knapsack temp = obj[i]; obj[i] = obj[j]; obj[j] =
temp; } } } for(int i = 0; i < n; i++) { cout << "Value" << i+1 << ": " << obj[i].value << " Weight" << i+1 << ": " <<
obj[i].weight << endl; }

2) Hamilton Cycle #include <iostream> using namespace std; int graph[5][5]={ {0,0,0,0,0}, {0,0,1,0,1},
{0,1,0,1,0}, {0,0,1,0,1}, {0,1,0,1,0}, }; int x[5]={0,1,0,0,0}; int n=4; void print(){ for(int i=1;i<=n;i++){ cout << x[i] << "
"; } cout << "1"; cout << endl; }void Next_value(int k){ do{ x[k]=(x[k]+1) % (n+1); if(x[k]==0) return; int j;
if(graph[x[k-1]][x[k]]!=0){ for(j=1;j<=(k-1);j++){ if(x[k]==x[j]){ break; } } if(j==k){ if(k<n || (k==n &&
graph[x[n]][x[1]])){ return; } } } }while(true); }void Hc(int k){ do{ Next_value(k); if(x[k]==0) return;
if(k==n){ print(); } else{ Hc(k+1); } }while(true); } using namespace std; int main() {Hc(2); return 0;}

L10:1) Job Assignment using Least Cost BB #include <bits/stdc++.h> using namespace std; const int INF =
1e9; const int MAXN = 20; int n, ans = INF; int a[MAXN][MAXN], vis[MAXN], assign[MAXN]; struct Node { int u,
cost, lb; bool operator < (const Node &other) const { return lb > other.lb; } }; void solve() { int lb = 0;for (int i =
1; i <= n; i++) { int mn = INF; for (int j = 1; j <= n; j++) { mn=min(mn, a[i][j]); } lb += mn; } if (lb >= ans) return;
priority_queue<Node> q; q.push({0, 0, lb}); while (!q.empty()) { auto [u, cost, lb] = q.top(); q.pop(); if (cost + lb
>= ans) break; if (u == n) { ans = cost; break; } for (int i = 1; i <= n; i++) { if (!vis[i]) { vis[i] = 1; assign[u + 1] =
i;int new_lb = lb; for (int j = u + 2; j <= n; j++) { int mn = INF; for (int k = 1; k <= n; k++) { if (!vis[k]) { Lab- 10 2
mn=min(mn, a[j][k]); } } new_lb += mn; } q.push({u + 1, cost + a[u + 1][i], new_lb}); vis[i] = 0; } } } } int main()
{ cin >> n; for (int i = 1; i <= n; i++) { for (int j = 1; j <= n; j++) { cin >> a[i][j];} } memset(vis, 0, sizeof(vis)); solve();
cout << ans << endl; return 0; }

2) 0/1 Knapsack using BB #include<bits/stdc++.h> using namespace std; float profit[4]={10,10,12,18}; float
weight[4]={2,4,6,9}; int capacity=15; int cp=0; int cw=0; float ub=0; vector<int> v(4,0); vector<int> sol(4,0); void
ks(int cp, int cw,int n){ if(n==3){ if(cp>ub){ ub=cp; for(int i=0;i<4;i++){ sol[i]=v[i]; } } return; } else{ float
ub_new=cp + (15-cw)*(profit[n+1]/weight[n+1]);if(ub_new<ub){ return; } if(cw+weight[n+1]<=15){ v[n+1]=1;
ks(cp+profit[n+1],cw+weight[n+1],n+1); } v[n+1]=0; ks(cp,cw,n+1); } } int main(){ ks(cp,cw,-1); cout << ub <<
endl; for(int i=0;i<4;i++){ cout << sol[i] << " "; } return 0; }

double capacity; cout << "Enter the capacity of bag: "; cin >> capacity; double profit = 0.0; for(int i = 0; i < n; i++)
{ if(capacity > 0 && obj[i].weight <= capacity) { capacity -= obj[i].weight; profit += obj[i].value; } else if(capacity
> 0 && obj[i].weight > capacity) { profit += (double)(obj[i].value * (capacity / obj[i].weight)); capacity = 0; } else
{ break; } } cout << "Total_profit: " << profit << endl; return 0; }

2. Making Change problem // making change using greedy method #include <bits/stdc++.h> using
namespace std; int main() { // here, we consider the case in which all notes frequency is one // int notes[] =
{1, 2, 5, 10, 20, 50, 100, 200, 500, 2000}; int n; cin >> n; int notes[n]; for(int i = 0; i < n; i++) { cin >> notes[i]; }
cout << "Enter your amount: " << endl; int amount; cin >> amount; int temp = amount, i = 0, count = 0;
sort(notes, notes + n); int req_notes[n]; for(int k = n-1; k >= 0; k--) { if(notes[k] <= temp) { req_notes[i] =
notes[k]; temp -= notes[k]; i++; }count++; } if(temp == 0) { cout << "Yes, making change is possible!!" << endl;
for(int j = 0; j < i; j++) { cout << req_notes[j] << " "; } } else { cout << "No, making change is not possible!!" <<
endl; cout << "Remaining amount: " << temp; } return 0; }

L4:1. Kruskal's algorithm to find the Minimum Cost Spanning Tree #include <bits/stdc++.h> using
namespace std; const int MAX = 1e6-1; // initially, all nodes value = infinite int root[MAX]; // array of all
nodes const int nodes = 6, edges = 10; pair <long long, pair<int, int> > p[MAX]; int parent(int a) {//find the
parent of the given node while(root[a] != a) { root[a] = root[root[a]]; a = root[a]; } return a; } void union_find(int
a, int b) { //check if the given two vertices are in the same "union" or not int d = parent(a); int e = parent(b);
root[d] = root[e]; } long long kruskal() { int a, b, k = 0; long long cost, min_cost = 0; cout << "Selected edges: "
<< endl; for(int i = 0 ; i < edges ; ++i) { a = p[i].second.first; b = p[i].second.second; cost = p[i].first; if(parent(a)
!= parent(b)) {//only select edge if it does not create a cycle (ie the two nodes forming it have different root
nodes) min_cost += cost; union_find(a, b); k++; cout << "Edge" << k << " : " << a << " " << b << endl; } } return
min_cost; } int main() { int x, y; long long weight, cost, min_cost; for(int i = 0; i < MAX; ++i) {//initialize the
array groups root[i] = i; } p[0] = make_pair(5, make_pair(0, 1)); p[1] = make_pair(3, make_pair(1, 5)); p[2] =
make_pair(2, make_pair(1, 3)); p[3] = make_pair(4, make_pair(0, 2)); p[4] = make_pair(1, make_pair(3, 4)); p[5]
= make_pair(6, make_pair(0, 3)); p[6] = make_pair(2, make_pair(0, 4)); p[7] = make_pair(2, make_pair(5, 3));
p[8] = make_pair(4, make_pair(5, 4)); p[9] = make_pair(3, make_pair(2, 4)); sort(p, p + edges);//sort the array
of edges min_cost = kruskal(); cout << "Minimum cost is: "<< min_cost << endl; return 0; }

L5:1. Making Change #include <bits/stdc++.h> #define N 1e9 using namespace std; // minimize problem int
makingchange(int amount, int n, int d[]) { int mat[n][amount + 1]; for(int i = 0; i < n; i++) {mat[i][0] = 0; }
for(int i = 0; i < n; i++) {for(int j = 0; j < (amount + 1); j++) {if(i == 0) {if(j < d[i]) {mat[i][j] = N;}else {mat[i][j] = 1 +
mat[i][j - d[1]];}}else {if(j < d[i]) {mat[i][j] = mat[i-1][j];}else {mat[i][j] = min(mat[i-1][j], 1+mat[i][j - d[i]]);}}}} return
mat[n-1][amount]; } int main() { int n = 3; int d[n] = {1,4,6}; // d means denomination int amount = 8; int
ans = makingchange(amount, n, d); cout << "Ans: " << ans << endl; return 0; }

2. 0/1 Knapsack #include <bits/stdc++.h> using namespace std; // maximization problem int knapsack(int
n, int m, int p[], int w[]) {int M[n+1][m+1]; // M = matrix for(int i = 0; i <= n; i++) { // if no capacity M[i][0] = 0; }
for(int j = 1; j <= m; j++) { // if no item M[0][j] = 0; } for(int i = 1; i <= n; i++) { for(int j = 1; j <= m; j++)
{if(w[i-1] <= j) {M[i][j] = max(M[i-1][j], p[i-1] + M[i-1][j-w[i-1]]);}else {M[i][j] = M[i-1][j];}}}int profit = M[n][m]; int k
= m, ans[n], t = 0;for(int i = n; i > 0 && profit > 0; i--) {if(profit == M[i-1][k]) {ans[t] = 0;}else {ans[t] = 1;profit -= p[i
-1];k -= w[i-1];}t++;}cout << "Selected item vector: ";for(int i = (n-1); i >= 0; i--) {cout << ans[i] << " "; } cout <<
endl;return(M[n][m]); } int main() { int n = 4; // no. of object int m = 8; // capacity int p[n] = {1,2,5,6}; //
value per object int w[n] = {2,3,4,5}; // weight per object int ans = knapsack(n, m, p, w); cout << "Profit: "
<< ans << endl; return 0; }

L8:1.N-Queen's problem.#include<bits/stdc++.h> using namespace std; int n = 4; int ans[5]; // int n = 5; // int
ans[6]; bool Place(int k, int i, int ans[]) { for(int j = 1; j <= (k-1); j++) { // int t1 = if(ans[j] == i || (abs(j-k) ==
abs(ans[j]- i))) { return false; } } return true; } void N_Queen(int k, int n) { int temp = 0; for(int i = 1; i <= n; i++)
{ if(Place(k,i,ans)) { ans[k] = i; if(k != n) { N_Queen(k+1,n); } else { temp = 1; } } } if(temp == 1) { cout << "<";
for(int j = 1; j <= n; j++) { cout << ans[j] << ","; } cout << ">" << endl; } } int main() { cout << "Enter value of n:";
cout << n << endl; N_Queen(1,n); return 0; }

L9:1) Graph coloring #include <iostream> using namespace std; int graph[5][5]={ {0,0,0,0,0}, {0,0,1,0,1},
{0,1,0,1,0}, {0,0,1,0,1}, {0,1,0,1,0}, }; int m=2; int x[5]={0,0,0,0,0}; int n=4;void print(){ for(int i=1;i<=n;i++){ cout <<
x[i] << " "; } cout << endl; } void Next_value(int k){ do{ x[k]=(x[k]+1) % (m+1); if(x[k]==0) return; int j;
for(j=1;j<=n;j++){ if(graph[k][j]!=0 && x[k]==x[j]){ break; } } if(j==n+1) return; }while(true);} void mcoloring(int
k){ do{ Next_value(k); if(x[k]==0) return; if(k==n){ print(); } else{ mcoloring(k+1); } }while(true); } using
namespace std; int main() { mcoloring(1); return 0;}