# Contest Templates - Index

```cpp
// 8Queens
pair < int, int > queen[9];
bool isvalid ( int row, int column ) {
    for ( int i = 1; i < row; i++ )  {
        if ( queen[i].ss == column || queen[i].ss - queen[i].ff == column - row || queen[i].ss + queen[i].ff == column + row )
            return false;
    }

    return true;
}
int cases = 0;
void generate ( int row ) {
    if ( row == 9 ) {
        cout << endl << "Way N.O. = " << ++cases << endl << endl;

        for ( int i = 1; i <= 8; i++ ) {
            cout << "Row = " << i << " | Column = " << queen[i].ss << endl;
        }

        return;
    }

    FOR ( i, 1, 8 ) {
        if ( isvalid ( row, i ) ) {
            queen[row] = mp ( row, i );
            generate ( row + 1 );
        }
    }
}

int main() {
    //freopen ("in.txt", "r", stdin);
    generate ( 1 );
    return 0;
}
// 8 Queens ends


// DFS
void dfs ( int source, bool visited[], vi graph[] ) {
    visited[source] = true;
    int l = graph[source].size();
    rep ( i, l ) {
        int v = graph[source][i];

        if ( !visited[v] )
            dfs ( v, visited, graph );
    }
}
// DFS ends


// Divisors
int numDiv ( int N ) {
    int i = 0, factor = primes[i], ans = 1;

    while ( factor * factor <= N ) {
        int power = 0;

        while ( N % factor == 0 ) {
            N /= factor;
            power++;
        }

        ans *= ( power + 1 );
        factor = primes[++i];
    }

    if ( N != 1 )
        ans *= 2;

    return ans;
}
// Divisors End


//Fenwick Tree Begins
int t;
long tree[1234567];

int next ( int n ) {
    return n + ( n & -n );
}
```

```cpp
int prev ( int n ) {
    return n - ( n & -n );
}

void update ( int i, long val ) {
    while ( i <= t ) {
        tree[i] += val;
        i = next ( i );
    }
}

long getsum ( int x ) {
    long sum = 0;

    while ( x > 0 ) {
        sum += tree[x];
        x = prev ( x );
    }

    return sum;
}

long sum ( int a, int b ) {
    if ( a > b )
        swap ( a, b );

    return getsum ( b ) - getsum ( a - 1 );
}
//Fenwick Tree Ends


//Floyed Warshall Begins
//Initially mat 2D array should hold inf in all of its indexes
for ( int k = 0; k < n; k++ ) {
    for ( int i = 0; i < n; i++ ) {
        for ( int j = 0; j < n; j++ ) {
            if ( mat[i][k] + mat[k][j] < mat[i][j] )
                mat[i][j] = mat[i][k] + mat[k][j];
        }
    }
}

//Floyed Warshall Ends


//isCycle (DFS Undirected) begins
vector <int> graph[123456];

// Initially int parent is infinity.
bool mark[123456];

bool iscycle ( int n, int parent ) {
    mark[n] = true;
    int length = graph[n].size();

    for ( int i = 0; i < length; i++ ) {
        int v = graph[n][i];

        if ( !mark[v] ) {
            if ( iscycle ( v, n ) )
                return true;
        } else if ( v != parent )
            return true;
    }

    return false;
}
//isCycle ends


// Kadanel Algorithm's Begins
int best = 0, sum = 0;

for ( int k = 0; k < n; k++ ) {
    sum = max ( array[k], sum + array[k] );
    best = max ( best, sum );
}

cout << best << "\n";
// Kadane Algorithm's ends


//Kruskal MST Begins
int parent[1234567];
```

```cpp

void setInitialParent() {
    for ( int i = 0; i < 1234567; i++ )
        parent[i] = i;
}

int fin ( int r ) {
    if ( parent[r] == r )
        return r;

    return parent[r] = fin ( parent[r] );
}

void unio ( int x, int y ) {
    int u = fin ( x );
    int v = fin ( y );
    parent[u] = parent[v];
}

struct node {
    int u, v, cost;

    node ( int x, int y, int z ) {
        u = x, v = y, cost = z;
    };

    bool operator < ( node other ) const {
        return cost < other.cost;
    };

};

int main() {
    //freopen ("in.txt", "r", stdin);
    //freopen ("out.txt", "w", stdout);
    //ios_base::sync_with_stdio(0);cin.tie(0);
    int m, n; //m is number of nodes & n is number of edges

    while ( scii ( m, n ) && m != 0 && n != 0 ) {
        setInitialParent();
        int x, y, z;
        vector < node > edge;

        for ( int i = 0; i < n; i++ ) {
            scii ( x, y );
            sci ( z );
            edge.push_back ( node ( x, y, z ) );
        }

        sort ( edge.begin(), edge.end() );
        int realCost = 0;

        for ( int i = 0; i < n; i++ ) {
            int u = edge[i].u;
            int v = edge[i].v;
            int cost = edge[i].cost;
            x = fin ( u ), y = fin ( v );

            if ( x != y ) {
                unio ( x, y );
                realCost += cost;
            }
        }

        printf ( "%d\n", realCost );
    }

    return 0;
}
// Kruskal MST ENDS


// MagicSquare(3x3) the sum of any row, column, or diagonal = 15
#include <bits/stdc++.h>

using namespace std;

int magic [9][3][3];

void process ( int i, int j, int num, int index ) {
    if ( !magic[index][i][j] ) {
        magic[index][i][j] = num;

        if ( num % 3 == 0 ) {
```

```
            j++;
            j = j % 3;
        } else {
            i++, j--;
            i = i % 3;
            j = ( j + 3 ) % 3;
        }

        num += 1;
        process ( i, j, num, index );
    }
}

void generateMagicSquare() {
    memset ( magic, 0, sizeof magic );
    int index = 0;

    for ( int i = 0; i < 3; i++ ) {
        for ( int j = 0; j < 3; j++ ) {
            process ( i, j, 1, index );
            index++;
        }
    }

    for ( int i = 0; i < 9; i++ ) {
        cout << "Index = " << i << endl;

        for ( int j = 0; j < 3; j++ ) {
            for ( int k = 0; k < 3; k++ ) {
                cout << magic[i][j][k] << " ";
            }

            cout << endl;
        }

        cout << endl << endl;
    }
}

int main() {
    generateMagicSquare();
    return 0;
}
// MagicSquare ends


// Max Sub Array Sum  Begins
#include<cstdio>
#include<cmath>
#include<iostream>
#include<climits>

using namespace std;

int max ( int a, int b ) {
    return ( ( a > b ) ? a : b );
}

int max ( int a, int b, int c ) {
    return max ( max ( a, b ), c );
}

int MaxCrossingArray ( int a[], int b, int m, int c ) {
    int left_sum = INT_MIN;
    int sum = 0;

    for ( int i = m; i >= b; i-- ) {
        sum += a[i];

        if ( sum > left_sum ) {
            left_sum = sum;
        }
    }

    int right_sum = INT_MIN;
    sum = 0;

    for ( int i = m + 1; i <= c; i++ ) {
        sum += a[i];

        if ( sum > right_sum )
            right_sum = sum;
    }

```

```
336         return ( left_sum + right_sum );
337     }
338
339     int MaxSubArray ( int a[], int c, int d ) {
340         if ( c == d )
341             return a[c];
342
343         int m = ( c + d ) / 2;
344         return max ( MaxSubArray ( a, c, m ), MaxSubArray ( a, m + 1, d ), MaxCrossingArray
    ( a, c, m, d ) );
345     }
346
347     int main() {
348         while ( true ) {
349             int a;
350             scanf ( "%d", &a );
351
352             if ( a == 0 )
353                 break;
354
355             int b[100005];
356
357             for ( int j = 0; j < a; j++ )
358                 cin >> b[j];
359
360             int max = MaxSubArray ( b, 0, a - 1 );
361
362             if ( max > 0 )
363                 printf ( "The maximum winning streak is %d.\n", max );
364             else
365                 printf ( "Losing streak.\n" );
366         }
367
368         return 0;
369     }
370     // Max Sub Array Ends
371
372
373     // MST tree from Graph
374     int parent[1234567];
375
376     void setInitialParent() {
377         for ( int i = 0; i < 1234567; i++ )
378             parent[i] = i;
379     }
380
381     int fin ( int r ) {
382         if ( parent[r] == r )
383             return r;
384
385         return parent[r] = fin ( parent[r] );
386     }
387
388     void unio ( int x, int y ) {
389         int u = fin ( x );
390         int v = fin ( y );
391         parent[u] = parent[v];
392     }
393
394     struct node {
395         int u, v, cost;
396
397         node ( int x, int y, int z ) {
398             u = x, v = y, cost = z;
399         };
400
401         bool operator < ( node other ) const {
402             return cost < other.cost;
403         };
404
405     };
406
407     int main() {
408         //freopen ("in.txt", "r", stdin);
409         //freopen ("out.txt", "w", stdout);
410         //ios_base::sync_with_stdio(0);cin.tie(0);
411         int m, n; //m is number of nodes & n is number of edges
412
413         while ( scii ( m, n ) && m != 0 && n != 0 ) {
414             setInitialParent();
415             int x, y, z;
416             vector < node > edge;
417             vector < pair < int, int > > graph [110];
418
```

```
        for ( int i = 0; i < n; i++ ) {
            scii ( x, y );
            sci ( z );
            edge.push_back ( node ( x, y, z ) );
        }

        sort ( edge.begin(), edge.end() );

        for ( int i = 0; i < n; i++ ) {
            int u = edge[i].u;
            int v = edge[i].v;
            int cost = edge[i].cost;
            x = fin ( u ), y = fin ( v );

            if ( x != y ) {
                unio ( x, y );
                graph[u].push_back ( make_pair ( v, cost ) );
                graph[v].push_back ( make_pair ( u, cost ) );
            }
        }
    }

    return 0;
}
// MST tree from Graph ENDS


// NCR Begins
L dp[110][110];

L nCr ( int n, int r )  {
    if ( n == r || r == 0 )
        return 1;

    if ( r == 1 )
        return n;

    if ( dp[n][r] )
        return dp[n][r];

    return dp[n][r] =  nCr ( n - 1, r - 1 ) + nCr ( n - 1, r );
}
// NCR ends


//Segment Tree Begins
int A[1234567];
int Tree[12345678];

void buildTree ( int p, int left, int right ) {
    if ( left == right ) {
        Tree[p] = A[left];
    } else {
        int mid = ( left + right ) / 2;
        buildTree ( p << 1, left, mid );
        buildTree ( ( p << 1 ) + 1, mid + 1, right );
        Tree[p] = Tree[p << 1] + Tree[ ( p << 1 ) + 1];
    }
}

int fquery ( int p, int left, int right, int x, int y ) {
    if ( x > right || y < left )
        return -1;

    if ( left >= x && right <= y )
        return Tree[p];

    int mid = ( left + right ) / 2;
    int p1 = fquery ( p << 1, left, mid, x, y );
    int p2 = fquery ( ( p << 1 ) + 1, mid + 1, right, x, y );

    if ( p1 == -1 )
        return p2;

    if ( p2 == -1 )
        return p1;

    return p1 + p2;
}

void updateTree ( int p, int left, int right, int x, int y ) {
    if ( left == right ) {
        A[left] = y;
        Tree[p] = y;
```

```
    } else {
        int mid = ( left + right ) / 2;

        if ( x >= left  && x <= mid )
            updateTree ( p << 1, left, mid, x, y );
        else
            updateTree ( ( p << 1 ) + 1, mid + 1, right, x, y );

        Tree[p] = Tree[p << 1] + Tree[ ( p << 1 ) + 1];
    }
}
//Segment Tree ends


// Seive Prime Begins
int prime[Local];
bool chk[Global];
int koto;
void pr() {
    prime[koto++] = 2;
    int n = sqrt ( Global ) + 1;

    for ( int i = 3; i <= n; i += 2 ) {
        if ( !chk[i] ) {
            for ( int j = i * i; j < Global; j += i * 2 ) {
                chk[j] = true;
            }
        }
    }

    for ( int i = 3; i < Global; i += 2 )
        if ( !chk[i] )
            prime[koto++] = i;
}
// Seive prime Ends


// Structure with Compare Function
struct node {
    int u, v, cost;

    node ( int x, int y, int z ) {
        u = x, v = y, cost = z;
    };

    bool operator < ( node other ) const {
        return cost < other.cost;
    };

};
// Structure with Compare Function ENDS


// Top Sort Begins
vector < int > lst;

void topsort ( int source, bool visited[], vector < int> graph[] ) {
    visited[source] = true;
    rep ( i, ( int ) graph[source].size() )

    if ( !visited[graph[source][i]] )
        topsort ( graph[source][i], visited, graph );

    lst.pb ( source );
}

// Top Sort Ends


//Union Finder Begins
int parent[200010];

int fin ( int r ) {
    if ( parent[r] == r )
        return r;

    return parent[r] = fin ( parent[r] );
}

void unio ( int x, int y ) {
    int u = fin ( x );
    int v = fin ( y );
    parent[u] = parent[v];
}
```