Question1

```
#include <bits/stdtr1c++.h>
using namespace std;
void select(int start[], int end[], int n)
{
   printf("The following activities are selected:
\n");
   int j = 0;
   printf("%d ", j);
   int i;
   for (i = 1; i < n; i++)
   {
      if (start[i] >= end[j])
      {
         printf("%d ", i);
```

```
            j = i;
        }
    }
}
int main()
{
    int n;
    cin >> n;
    int start[n];
    for(int i=0;i<n;i++) {
        cin >> start[i];
    }
    for(int i=0;i<n;i++) {
        cout << start[i] +" ";
    }
```

```
    int END[n];
    for(int i=0;i<n;i++) {
        cin >> END[i];
    }
    for(int i=0;i<n;i++) {
        cout << END[i] +" ";
    }
    select(start, END, n);
    return 0;
}
```

Question2.

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define MAX 50
struct node
{
   int freq;
   char symbol;
   struct node *left, *right;
};

struct heap
{
   int size;
   int capacity;
   struct node **array;
};
```

```
struct node *newNode(char symbol, int freq)
{
   struct node *temp = (struct node *)
malloc(sizeof(struct node));

   temp->left = temp->right = NULL;
   temp->symbol = symbol;
   temp->freq = freq;
   return temp;
}

struct heap *createMinH(int capacity)
{
   struct heap *minHeap = (struct heap *)
malloc(sizeof(struct heap));
```

```
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct node **)
malloc(minHeap->capacity * sizeof(struct node
*));
    return minHeap;
}

void print(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        cout << arr[i];

    cout << "\n";
```

```
}

void swap(struct node **a, struct node **b)
{
    struct node *t = *a;
    *a = *b;
    *b = t;
}

void heapify(struct heap *minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
```

```
    if (left < minHeap->size && minHeap->
array[left]->freq < minHeap->array[smallest]->
freq)
        smallest = left;

    if (right < minHeap->size && minHeap->
array[right]->freq < minHeap->array[smallest]->
freq)
        smallest = right;

    if (smallest != idx)
    {
        swap(&minHeap->array[smallest],
&minHeap->array[idx]);
        heapify(minHeap, smallest);
    }
```

```
}

int checkSizeOne(struct heap *minHeap)
{
    return (minHeap->size == 1);
}

struct node *extractMin(struct heap *minHeap)
{
    struct node *temp = minHeap->array[0];
    minHeap->array[0] = minHeap->
array[minHeap->size - 1];
    --minHeap->size;
    heapify(minHeap, 0);
    return temp;
```

```
}


void insertMinHeap(struct heap *minHeap,
struct node *minHeapNode)
{
   ++minHeap->size;
   int i = minHeap->size - 1;
   while (i && minHeapNode->freq < minHeap->
array[(i - 1) / 2]->freq)
   {
      minHeap->array[i] = minHeap->array[(i - 1)
/ 2];
      i = (i - 1) / 2;
   }


   minHeap->array[i] = minHeapNode;
```

```
}

void buildMinHeap(struct heap *minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        heapify(minHeap, i);
}

int isLeaf(struct node *root)
{
    return !(root->left) && !(root->right);
}
```

```
struct heap *createMinHeap(char symbol[], int
freq[], int size)
{
   struct heap *minHeap = createMinH(size);
   for (int i = 0; i < size; ++i)
      minHeap->array[i] = newNode(symbol[i],
freq[i]);
   minHeap->size = size;
   buildMinHeap(minHeap);
   return minHeap;
}

struct node *buildHfTree(char symbol[], int
freq[], int size)
{
   struct node *left, *right, *top;
```

```
    struct heap *minHeap =
createMinHeap(symbol, freq, size);
    while (!checkSizeOne(minHeap))
    {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}
void printHCodes(struct node *root, int arr[], int
top)
```

```
{
    if (root->left)
    {
        arr[top] = 0;
        printHCodes(root->left, arr, top + 1);
    }

    if (root->right)
    {
        arr[top] = 1;
        printHCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root))
    {
        cout << root->symbol << "  | ";
```

```
        print(arr, top);
    }
}

void HuffmanCodes(char symbol[], int freq[], int
size)
{
    struct node *root = buildHfTree(symbol, freq,
size);
    int arr[MAX], top = 0;
    printHCodes(root, arr, top);
}

int main()
{
```

```
    char arr[] = {'Q', 'P', 'T', 'a','d'};
    int freq[] = {3, 23, 30, 12, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Char  |  Huffman code ";
    cout << "\n--------------------\n";
    HuffmanCodes(arr, freq, size);
}
```

Question3.

```
#include <bits/stdc++.h>

using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], n, m;
pair<long long, pair<int, int>> p[MAX];
```

```
void initialize()
{
    for (int i = 0; i < MAX; ++i)
        id[i] = i;
}

int root(int x)
{
    while (id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
```

```
}

void union1(int x, int y)
{
   int p = root(x);
   int q = root(y);
   id[p] = id[q];
}

long long kruskal(pair<long long, pair<int, int>>
p[])
{
   int x, y;
   long long cost, minimumCost = 0;
   for (int i = 0; i < m; ++i)
```

```
    {
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;

        if (root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
```

```
{
    int x, y;
    long long weight, minimumCost;
    initialize();
    cin >> n >> m;
    for (int i = 0; i < m; ++i)
    {
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }
    sort(p, p + m);
    minimumCost = kruskal(p);
    cout << minimumCost << endl;
    return 0;
}
```