TASK1

Find the kth largest element in an unsorted array. Note that is the kth largest element in the sorted order, not the kth distinct element.

For example:
Given [3,2,1,5,6,4] and k=2, return 5.
Note: You may assume k is always valid, 1<=k<=arrays length.
Use the following signature: int task1(vector<int>& nums, int k);

Solution:

```
int task1(vector<int>& nums, int k)
{
	int N = nums.size();
	sort(nums.begin(),nums.end());



	return nums[N-k];
}
```

TASK2

Write an efficient algorithm that searches for a value in a MxN matrix. This matrix has the following properties:
- integers in each row are sorted in ascending from left to right;
- integers in each column are sorted in ascending from top to bottom;

For example:

1 4 7 11 15
2 5 8 12 19
3 6 9 16 22
10 13 14 17 24
18 21 23 26 30

Given target=5, return true.
Given target=20, return false.

Use following signature:
bool task2(vector< vector<int> >& matrix, int target);

Solution:

```
bool task2(vector< vector<int> >& matrix, int target)
{
	int i = 0;
	int j = matrix.size() - 1;
	while (i<=matrix.size()-1 && j>=0)
	{
		if (target == matrix[i][j])
			return true;


		else
			if (target > matrix[i][j])
			i+=1;
		else
			if (target < matrix[i][j])
			j-=1;


	}
	return false;
}
```

TASK3

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once.
You must make sure your result is the smallest in lexicographical order among all posible results.
Example:
Given "bcabc" return "abc"
Given "cbacdcbc" return "abcd"
Use following signature: string task3(string s);

Solution:

```
string task3(string s)
{
	sort(s.begin(), s.end());
	s.erase(unique(s.begin(), s.end()), s.end());
	return s;
}
```

TASK4

Given a binary tree, determite if it is height-balanced. For this problem, a height-balanced binary tree is defined as a binary tree which the depth of the two subtrees of every node never differ by more then 1.
Use following data structure and function structure:

```
struct TreeNode
{
	int data;
	TreeNode *left;
	TreeNode *right;
	TreeNode(int x) :data(x), left(nullptr), right(nullptr){}
};

bool task4(TreeNode *root);
```

Solution:

```
int balanceHeight(TreeNode *root)
{
	if (root == NULL)
		return 0;
	int leftHeight = balanceHeight(root->left);
	if (leftHeight == -1)
	{
		return -1;
	}
	int rightHeight = balanceHeight(root->right);
	if (rightHeight == -1)
		return -1;
	if (abs(leftHeight - rightHeight) > 1)
		return -1;

	return (1 + max(leftHeight, rightHeight));
}

bool task4(TreeNode *root)
{
	if (balanceHeight(root) > -1)
		return true;
	else
		return false;

}
```

TASK5

Given N points on a 2D plane, find the maximum number of points that lie on the same straight line.
Use following data structure and function signature:

```
struct Point
{
	int x;
	int y;
	Point() :x(0), y(0){}
	Point(int a, int b) :x(a), y(b){}
};

int task5(vector<Point>& points);
```

Solution:

```
bool operator < (const Point& a, const Point& b)
{
	return a.x < b.x || a.x == b.x && a.y < b.y ;
}
bool operator == (const Point& a, const Point& b)
{
	return a.x == b.x && a.y == b.y;
}
int task5(vector<Point>& points)
{
	sort(points.begin(), points.end());
	points.resize(unique(points.begin(), points.end()) - points.begin());

	if (points.size() == 1)
	{
		cout << "All points are the same!" << endl;
		cout << "Answer:";
		return 0;
	}

	vector<float> k(points.size());

	cout << "Tangens alpha:" << endl;
	for (unsigned int i = 0; i < points.size(); i++)
	{
		if (points[i].x == 0)
			k[i] = numeric_limits<float>::infinity();
		else
			k[i] = (float)(points[i].y) / (float)(points[i].x);
```

```
            cout << k[i] << endl;
        }

        int counter = 1;

        int BcountOX = 0;
        int BcountOY = 0;
        int temp = 0;

        int result = 2;
        for (unsigned int z = 0; z < k.size(); z++)
        {
            if (BcountOX>temp || BcountOY>temp)
                temp = max(BcountOX, BcountOY);


            counter = 1;
            BcountOX = 1;
            BcountOY = 1;

            for (unsigned int j = z+1; j < k.size(); j++)
            {
                if (points[z].x == points[j].x)
                    BcountOX++;
                if (points[z].y == points[j].y)
                    BcountOY++;
                if (k[z] == k[j])
                    counter++;


            }
            if (counter > result)
                result = counter;
        }

        if (result > temp)
            return result;
        else
            return temp;
}
```