Linked List:

Insertion 1

```
        if (current->value>=value) {
                // the value of the current
element is bigger
                // so insert the new value
just before

                List *elem = new List;
                elem->value=value;

                elem->next=current;
                current=elem;
        } else {
                // go on recursively

                Insert(current->next, value);
        }
}

void printList(List* list) {
        if (list!=0) {
                cout << list->value << endl;
                printList(list->next);
        }
}

int main(int argc, char** argv) {
        List* list=0;
        Insert(list, 5);
        Insert(list, 4);
        Insert(list, 2);
        Insert(list, 9);
        Insert(list, 342);
        Insert(list, 45);
        Insert(list, 1);

        printList(list);
}
```

Insertion 2

```
#include <iostream>

using namespace std;

struct List {
        List* next;
        int value;

        List() {
                value=-1;
                next=0;
        }
};

void Insert(List *&current, int value) {

        if (current==0) {
                current = new List;
                current->value=value;
                return;
        }

        // handle where the head is the
biggest element
        if (current->value>value) {
                List *elem = new List;
                elem->value=value;

                elem->next=current;
                current=elem;
                return;
        }

        // handle the other case
        List *list = current;
        List *next = current->next;

        while (next!=0 && next->value <=
value) {
                list=list->next;
                next=next->next;
        }

        List *elem = new List;
        elem->value=value;

        if (next==0) {
                list->next=elem;
        } else {
                elem->next=list->next;
                list->next=elem;
        }
}

void printList(List* list) {
        if (list!=0) {
                cout << list->value << endl;
                printList(list->next);
        }
}

int main(int argc, char** argv) {
        List* list=0;
        Insert(list, 5);
        Insert(list, 4);
        Insert(list, 2);
        Insert(list, 9);
        Insert(list, 342);
        Insert(list, 45);
        Insert(list, 1);

        printList(list);
}
```

Delete

```
#include <iostream>

using namespace std;

struct List {
        List* next;
        int value;

        List() {
                value=-1;
                next=0;
        }
};

void Insert(List *&current, int value) {
        if (current==0) {
                // we are at the tail of the list
                // so just insert the value

                current = new List;
                current->value=value;

                return;
        }

        if (current->value>=value) {
                // the value of the current
element is bigger
                // so insert the new value
just before

                List *elem = new List;
                elem->value=value;

                elem->next=current;
                current=elem;
        } else {
                // go on recursively

                Insert(current->next, value);
        }
}

void Delete(List *&list, int value) {

        // handle empty list
        if (list==0) return;

        // handle the head of the list
        if (list->value==value) {
                List *tmp=list;
                list=list->next;

                delete tmp;
                return;
        }

        List *cur=list;
        while (cur->next != 0) {
                if (cur->next->value ==
value) {
                        List *tmp = cur-
>next;
                        cur->next=cur-
>next->next;
                        delete tmp;

                        return;
                }

                cur=cur->next;
        }
}

void printList(List* list) {
        if (list!=0) {
                cout << list->value << " ";
                printList(list->next);
        }
}

int main(int argc, char** argv) {
        List* list=0;
        Insert(list, 5);
        Insert(list, 4);
```

```
	Insert(list, 2);
	Insert(list, 9);
	Insert(list, 342);
	Insert(list, 45);
	Insert(list, 1);

	printList(list);
	cout << endl << endl;

	Delete(list, 1);
	printList(list);
	cout << endl << endl;

	Delete(list, 9);
	printList(list);
	cout << endl << endl;

	Delete(list, 342);
	printList(list);
	cout << endl;
}
```

Contains

```
#include <iostream>

using namespace std;

struct List {
	List* next;
	int value;

	List() {
		value=-1;
		next=0;
	}
};

void Insert(List *&current, int value) {
	if (current==0) {
		// we are at the tail of the list
		// so just insert the value

		current = new List;
		current->value=value;

		return;
	}

	if (current->value>=value) {
		// the value of the current
element is bigger
		// so insert the new value
just before

		List *elem = new List;
		elem->value=value;

		elem->next=current;
		current=elem;
	} else {
		// go on recursively

		Insert(current->next, value);
	}
}

void printList(List* list) {
	if (list!=0) {
		cout << list->value << endl;
		printList(list->next);
	}
}

bool contains(List* l1, List* l2) {
	if (l1==0 || l2==0) {
		return false;
	}

	List* t1=l1;
	List* t2=l2;

	while(t1!=0 && t2!=0 && t1-
>value==t2->value) {
		t1=t1->next;
		t2=t2->next;
	}

	if (t2==0) {
		return true;
	}

	return contains(l1->next, l2);
}

void evaluate(bool b) {
	if (b) {
		cout << "true" << endl;
	} else {
		cout << "false" << endl;
	}
}
```

```
int main(int argc, char** argv) {
	List* list1=0;
	Insert(list1, 5);
	Insert(list1, 4);
	Insert(list1, 2);
	Insert(list1, 9);
	Insert(list1, 342);
	Insert(list1, 45);
	Insert(list1, 1);

	List* list2=0;
	Insert(list2, 5);
	Insert(list2, 1);
	Insert(list2, 2);

	printList(list1);
	printList(list2);

	evaluate(contains(list1, list2));

	Insert(list2, 4);

	evaluate(contains(list1, list2));
}
```

Array Search

```
#include <iostream>

using namespace std;

void search(int *cur, int *end, int elem) {
	if (cur != end) {
		if (*cur == elem) {
			cout << "element
found " << endl;
		} else {
			search(++cur, end,
elem);
		}
	} else {
		cout << "element " << elem
<< " not found" << endl;
	}
}

int main(int argc, char **argv) {
	int arr[] =
{ 1,2,4,3,6,7,8,9,12,24,534,234,65,435,432
,12 };

	search(arr, &arr[16], 65);
	search(arr, &arr[16], 43);
}
```

Sort

```
#include <iostream>

using namespace std;

void print(int* a, unsigned N) {
	for (int i=0; i<N; i++) {
		cout << a[i] << endl;
	}
}

void selection(int* a, unsigned N) {
	int index;
	int temp;
	for (int i=0; i<N; i++) {
		index=i;
		// search the smallest
element
		for (int j=i; j<N; j++) {
			if (a[j]<a[index])
index=j;
		}
		// swap
		temp=a[index];
		a[index]=a[i];
		a[i]=temp;
	}
}

void bubble(int* a, unsigned N) {
	int temp;
	for (int i=1; i<N; i++) {
		for (int j=i; j>=1; j--) {
			if (a[j]<a[j-1]) {
				// swap the
elements
				temp=a[j];
				a[j]=a[j-1];
				a[j-1]=temp;
			}
		}
	}
}

int main(int argc, char**) {
	const unsigned N=12;
	int
a[]={6,5,3,7,2,1,796,54,24,526,74,16};
	int b[N];

	memcpy(b, a, N*sizeof(int));

	bubble(a, N);
	selection(b, N);

	print(a, N);
	print(b, N);
	system("pause");

}
```

Trees

```
#include <iostream>

using namespace std;

struct Tree {
	Tree *left, *right;
	int key;

	Tree() {
		left=right=0;
		key=-1;
	}
};

void Insert(Tree* &tree, int value) {
	if (tree==0) {
		tree = new Tree;
		tree->key=value;
		return;
	}

	if (tree->key>value) {
		Insert(tree->left, value);
	} else {
		Insert(tree->right, value);
	}
}

int SearchMinimum(Tree* tree) {
	// assumend all values bigger than 0
	if (tree==0) return 0;

	if (tree->left!=0) {
		return
SearchMinimum(tree->left);
	}
	return tree->key;
}

int SearchMaximum(Tree* tree) {
	// assumend all values bigger than 0
	if (tree==0) return 0;

	if (tree->right!=0) {
		return
SearchMaximum(tree->right);
	}
	return tree->key;
}
```

```
int Sum(Tree* tree) {
	if (tree==0) return 0;

	return tree->key + Sum(tree->left)
+ Sum(tree->right);
}

void print(Tree* tree) {
	if (tree!=0) {
		cout << tree->key << endl;
		print(tree->left);
		print(tree->right);
	}
}

int main(int argc, char** argv) {

	Tree *root = 0;
	Insert(root, 3);
	Insert(root, 2);
	Insert(root, 5);
	Insert(root, 8);
	Insert(root, 1);
	Insert(root, 9);
	Insert(root, 7);
	Insert(root, 4);

	print(root);
	cout << endl << endl;

	cout << SearchMinimum(root) <<
endl;
	cout << SearchMaximum(root) <<
endl;

	cout << Sum(root) << endl;
}
```

Delete Trees

```
// shall show the implementation of a
deletion from a
// binary tree

#include <iostream>
using namespace std;

struct Tree {
	Tree *left, *right;
	int key;

	Tree() {
		left=right=0;
		key=-1;
	}
};

void Insert(Tree* &tree, int value) {
	if (tree==0) {
		tree = new Tree;
		tree->key=value;
		return;
	}

	if (tree->key>value) {
		Insert(tree->left, value);
	} else {
		Insert(tree->right, value);
	}
}

void Delete(Tree* &tree, int value) {
	if (tree==0) return;
	if (tree->key == value) {

		// test if one of the leaves
		// is equal to null
		if ((tree->left==0) || (tree-
>right==0)) {
			// store the element
to delete in a temporary
			Tree* elem = tree;
			if (tree->left==0) {
				tree=tree-
>right;
			} else {
				tree=tree-
>left;
			}

			delete elem;
			return;
		}

		// second case where both
subtrees are
		// not empty.
		// search the smallest
element in the
		// right subtree

		Tree **p = &tree->right;
		while ((*p)->left != 0) {
			// get the address of
the left tree
			p=&((*p)->left);
		}

		// store the minimum in a
temporary pointer
		Tree *tmp = (*p);
		// store the element to delete
in a temporary
		Tree *elem = tree;

		// remove the minimum out
of the tree by
		// draging up the right
subtree of the minimum
		(*p)=(*p)->right;
		// link the minimum into the
tree where we
		// wanted to delete
		tmp->left=tree->left;
		tmp->right=tree->right;
		tree=tmp;

		// delete the node from the
tree
		delete elem;

	} else {
		// delete recursively in the
tree
		if (tree->key>value) {
			Delete(tree->left,
value);
		} else {
			Delete(tree->right,
value);
		}
	}
}

void print(Tree* tree) {
	if (tree!=0) {
		cout << tree->key << endl;
```

```
                print(tree->left);
                print(tree->right);
        }
}

int main(int argc, char** argv) {

        Tree *root = 0;
        Insert(root, 3);
        Insert(root, 2);
        Insert(root, 5);
        Insert(root, 8);
        Insert(root, 1);
        Insert(root, 9);
        Insert(root, 7);
        Insert(root, 4);

        print(root);
        cout << endl << endl;

        Delete(root, 2);
        print(root);
        cout << endl << endl;
        cout << endl << endl;

        Delete(root, 9);
        print(root);
        cout << endl << endl;
        cout << endl << endl;

        Delete(root, 3);
        print(root);
        cout << endl << endl;
        cout << endl << endl;
}
/*
3
2
1
5
4
8
7
9


3
1
5
4
8
7
9


3
1
5
4
8
7


4
1
5
8
7

*/
```