1. Sortarea prin interschimbare (bubble sort)

```
#include <iostream>

void read(int n, int A[]) {
  for (int i = 0; i < n; i++) {
    std::cout << "A[" << i << "] = ";
    std::cin » A[i];
  }
}

void print(int n, int A[]) {
  for (int i = 0; i < n; i++)
    std::cout << A[i] << " ";
  std::cout << std::endl;
}

void swap(int &a, int &b) {
  int aux = a;
  a = b;
  b = aux;
}

void bubbleSort(int n, int A[]) {
  for (int i = 1; i < n; i++) {
    for (int j = n-1; j >= i; j--) {
      if (A[j-1] > A[j]) {
        swap(A[j-1], A[j]);
      }
    }
  }
}

int main ( ) {
  int n = 0;
  std::cout << "n = " ;
  std::cin » n;
  int A[n];
  read(n, A);
  bubbleSort(n, A);
  print(n, A);
  return 0;
}
```

2. Sortarea prin interschimbare (bubble sort) îmbunătăţită

```
const int UNDEFINED = -1;

void bubbleSort2(int n, int A[]) {
  int p = UNDEFINED;
  for (int i = 1; i < n; i++) {
    for (int j = n-1; j >= i; j--) {
      if (A[j-1] > A[j]) {
        swap(A[j-1], A[j]);
        p = j;
      }
    }
    if (p == UNDEFINED) break ;
    else {
      i = p;
      p = UNDEFINED;
    }
  }
}
```

3. Sortarea prin inserţie

```
void insertionSort(int n, int A[]) {
  for (int i = 1; i < n; i++) {
    int cheie = A[i];
    int j = i - 1;
    while (j > -1 && A[j] > cheie) {
      A[j+1] = A[j];
      j = j - 1;
    }
    A[j+1] = cheie;
  }
}
```

4. Sortarea prin selecţie

```
void minSelectionSort(int n, int A[]) {
  for (int i = 0; i < n-1; i++) {
    int minPos = i;
    for (int j = i + 1; j < n; j++) {
      if (A[j] < A[minPos]) {
        minPos = j;       }
    }
    swap(A[i], A[minPos]);
  }
}

void maxSelectionSort(int n, int A[]) {
  for (int i = n-1; i > 0; i--) {
    int maxPos = i;
    for (int j = i-1; j >= 0; j--) {
      if (A[j] > A[maxPos]) {
        maxPos = j;
      }
    }
    swap(A[i], A[maxPos]);
  }
}
```

5. Căutare liniară

```
void linearSearch(int n, int A[], int val, int
&loc) {
  loc = UNDEFINED;
  int i = 0;
  while (i < n && A[i] != val)
    i++;
  if (i < n)
    loc = i;
}

int main ( ) {
  int n = 0;
  std::cout << "n = " ;
  std::cin » n;
  int A[n];
  read(n, A);
  print(n, A);
  std::cout << "Introduceti valoarea
cautata\nval = " ;
  int val = 0;
  std::cin » val;
  int loc = UNDEFINED;
  linearSearch(n, A, val, loc);
  if (loc != UNDEFINED)
    std::cout << "Valoarea cautata este A["
<< loc << "].\n" ;
  else
    std::cout << "Valoarea cautata nu se afla
in A. \n" ;
  return 0;
}
```

6. Căutare liniară cu componentă marcaj

```
 void linearSearch2(int n, int A[], int val,
int &loc) {
  A[n] = val;
  loc = 0;
  while (A[loc] != val)
    loc++;
  if (loc == n)
    loc = UNDEFINED;
}

int main ( ) {
  int n = 0;
  std::cout << "n = " ;
  std::cin » n;
  int A[n + 1];
  read(n, A);
  print(n, A);
  std::cout << "Introduceti valoarea
cautata\nval = " ;
   int val = 0;
  std::cin » val;
  int loc = UNDEFINED;
  linearSearch2(n, A, val, loc);
  if (loc != UNDEFINED)
    std::cout << "Valoarea cautata este A["
<< loc << "].\n" ;
  else
    std::cout << "Valoarea cautata nu se afla
in A. \n" ;
  return 0;
}
```

7. Căutare binară

```
 void binarySearch(int n, int A[], int val,
int &loc) {
  int left = 0;
  int right = n - 1;
  int mid = (left + right) / 2;
  while (left <= right && val != A[mid]) {
    if (val < A[mid]) right = mid - 1;
    else left = mid + 1;
    mid = (left + right) / 2;
  }
  if (A[mid] == val) loc = mid;
  else loc = UNDEFINED;
}

int main ( ) {
  int n = 0;
  std::cout << "n = " ;
  std::cin » n;
  int A[n];
  read(n, A);
  insertionSort(n, A);
  print(n, A);
  std::cout << "Introduceti valoarea
cautata\nval = " ;
  int val = 0;
  std::cin » val;
  int loc = UNDEFINED;
  binarySearch(n, A, val, loc);
  if (loc != UNDEFINED)
    std::cout << "Valoarea cautata este A["
<< loc << "].\n" ;
  else
    std::cout << "Valoarea cautata nu se afla
in A. \n" ;
  return 0;
}
```