```
//6.10
NodeList::NodeList() {                          // constructor
   n = 0;                                       // initially empty
   header = new Node;                                   // create sentinels
   trailer = new Node;
   header->next = trailer;                              // have them point to each other
   trailer->prev = header;
 }

 int NodeList::size() const                     // list size
   { return n; }

 bool NodeList::empty() const                   // is the list empty?
   { return (n == 0); }


NodeList::Iterator NodeList::begin() const      // begin position is first item
   { return Iterator(header->next); }

 NodeList::Iterator NodeList::end() const       // end position is just beyond
last
   { return Iterator(trailer); }

//6.11
                                                // insert e before p
  void NodeList::insert(const NodeList::Iterator& p, const Elem& e) {
   Node* w = p.v;                               // p's node
   Node* u = w->prev;                           // p's predecessor
   Node* v = new Node;                          // new node to insert
   v->elem = e;
   v->next = w;  w->prev = v;                   // link in v before w
   v->prev = u;  u->next = v;                   // link in v after u
   n++;
 }

 void NodeList::insertFront(const Elem& e)      // insert at front
   { insert(begin(), e); }
```

```
void NodeList::insertBack(const Elem& e)        // insert at rear
  { insert(end(), e); }
```

//6.12

```
 void NodeList::erase(const Iterator& p) {      // remove p
  Node* v = p.v;                                // node to remove
  Node* w = v->next;                            // successor
  Node* u = v->prev;                            // predecessor
  u->next = w;  w->prev = u;                    // unlink p
  delete v;                                     // delete this node
  n--;                                    // one fewer element
}

void NodeList::eraseFront()                     // remove first
  { erase(begin()); }

void NodeList::eraseBack()                      // remove last
  { erase(--end()); }
}
```

Algorithm1.cpp

```cpp
#include <cstdlib>            // provides EXIT_SUCCESS
  #include <iostream>         // I/O definitions
  #include <vector>           // provides vector
  #include <algorithm>        // for sort, random_shuffle
 using namespace std;         // make std:: accessible
  int main () {
    int a[] = {17, 12, 33, 15, 62, 45};
    vector<int> v(a, a + 6);        // v: 17 12 33 15 62 45
    cout << v.size() << endl;       // outputs: 6
    v.pop_back();              // v: 17 12 33 15 62
    cout << v.size() << endl; // outputs: 5
    v.push_back(19);           // v: 17 12 33 15 62 19
    cout << v.front() << " " << v.back() << endl;// outputs: 17 19
    sort(v.begin(), v.begin() + 4);             // v: (12 15 17 33) 62 19
    v.erase(v.end() - 4, v.end() - 2);          // v: 12 15 62 19
    cout << v.size() << endl;       // outputs: 4

    char b[] = {'b', 'r', 'a', 'v', 'o'};
        vector<char> w(b, b + 5);         // w: b r a v o
        random_shuffle(w.begin(), w.end());     // w: o v r a b
        w.insert(w.begin(), 's');               // w: s o v r a b
    for (vector<char>::iterator p = w.begin(); p != w.end(); ++p)
          cout << *p << " ";        // outputs: s o v r a b
        cout << endl;
        return EXIT_SUCCESS;
      }
```

Output

```
6
5
17 19
4
s o r v a b
```