Lists: Another Application of Arrays

- What is a list ?

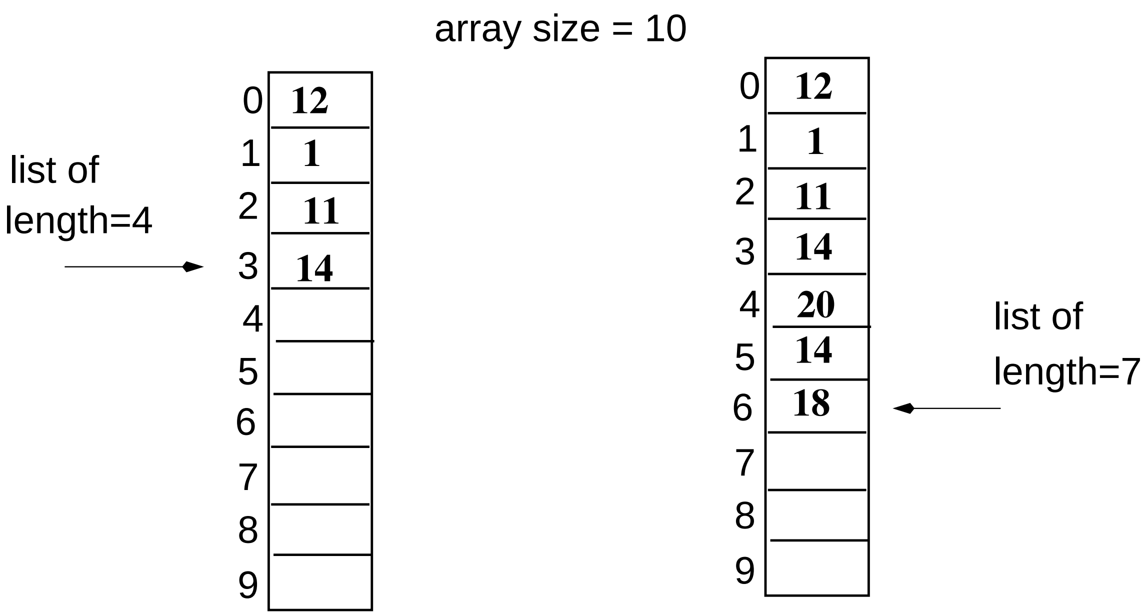
  - It is a collection of similar components (e.g., names, telephone numbers, addresses, etc.)

  - The number of elements in the list is the *length* of the list

- How can we implement a list ?

  - One way is using arrays (*size* >= *length*)

Implementing lists using arrays



- Common operations applied on lists

  - Create a list

  - Add an item to a list

  - Delete an item from a list

  - Search a list for a particular value

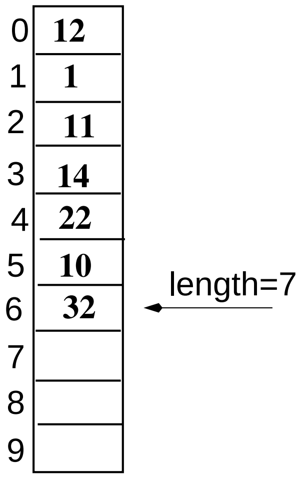
  - Sort a list (alphabetically or numerically)

- Inserting an element (unsorted list)

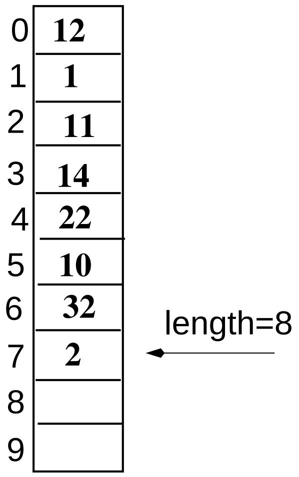
Inserting an element into a list

item = 2

Before insertion



After insertion

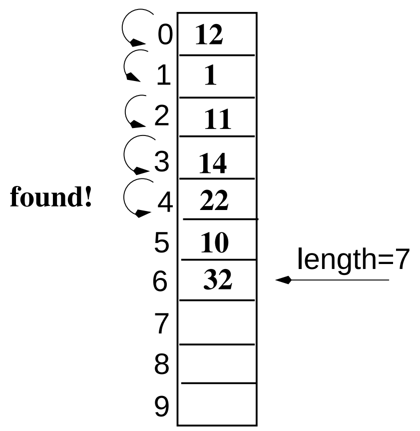



```
#include <iostream.h>
void Insert(int[], int, int&,int);
main()
{
 int value, l=7, s=10;
 int arr[10] = {12, 1, 11, 14, 22, 10, 32, 0, 0, 0};

 cout << "Enter a value";
 cin >> value;
 Insert(arr, value, l, s);
}


void Insert(int list[], int item, int& length,int size)
{
   if(length < size)
   {
       ++length;
       list[length] = item;
   }
   else  cout << "List is full !!  Insertion failed";
}
```

- Sequential search (unsorted list)

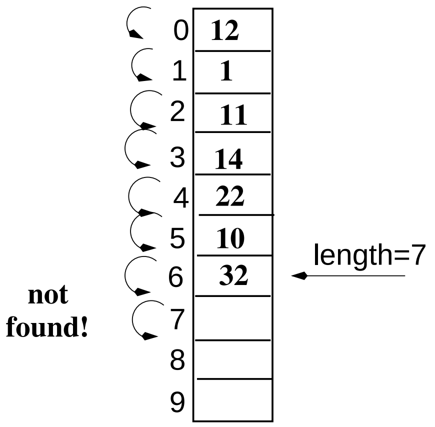
Searching for an item in a list

item = 22 (exists in the list)




```
#include <iostream.h>
void Seach(int[], int, int, int&)
 main()
 {
    int arr[10] = {12,1,11,14,22,10,32,0,0,0};
    int value, i, l=7;

    cout << "Enter a value";
    cin >> value;
    Search(arr,value,l,i);
    if (i == length)  cout << value << " does not exist in the list";
    else  cout << value << " exists at the  " << i << "-th place";
 }
```

item = 2 (does not exist in the list)

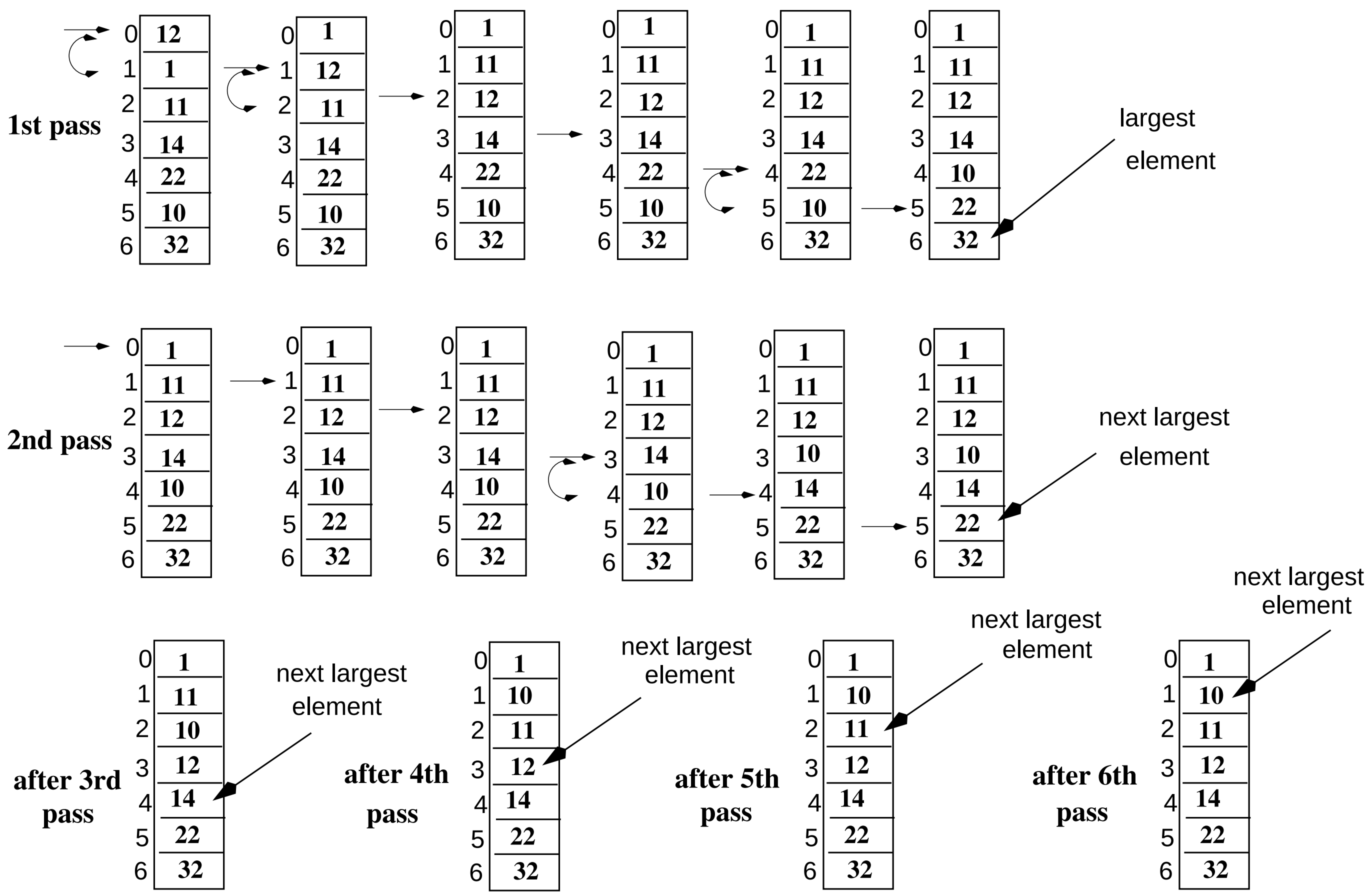



```
 void Seach(int list[], int item, int length, int& index)
 {
    int i, found;

    i=0;
    found = 0;  // Boolean variable

    while ( (i < length) && (!found) )
    {
       if (list[i] == item) {
          found = 1;
          index = i;
       }
       ++i;
    }
 }
```

- Sorting (*Bubblesort*)

  - **Idea**: Scan the list from left to right, swapping entries whenever a pair of adjacent elements is found to be out of order

  - Make *n* passes over the list (*n*: number of elements)



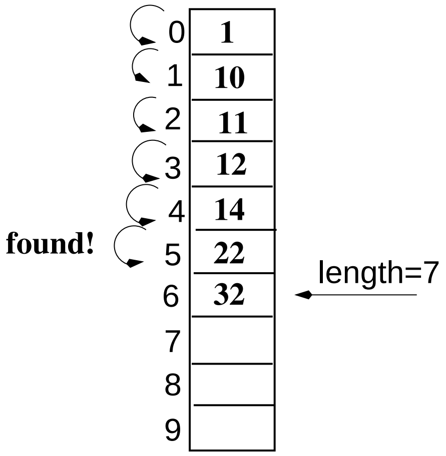

```
void Bubblesort(int list[], int length)
{
 int pass, i;
 int temp;

 for (pass = 1; pass < length; pass++)
   for(i = 0; i < length - 1; i++)
     if(list[i] > list[i+1])
     {
       temp = list[i];
       list[i] = list[i+1];
       list[i+1] = temp;
     }
}
```

- Sequential search (sorted list)

  - An item is missing if we pass its correct place in the list !!

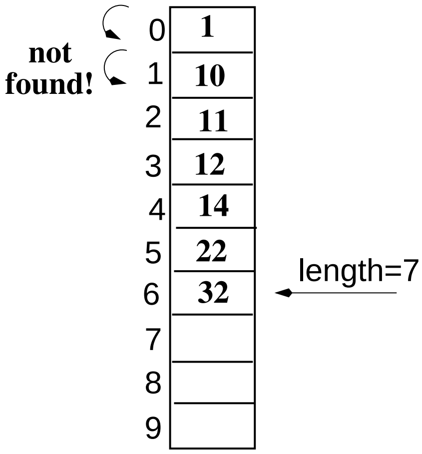
Searching for an item in a list

item = 22 (exists in the list)




```
#include <iostream.h>
void Seach(int[], int, int, int&);
main()
{
   int arr[10] = {12,1,11,14,22,10,32,0,0,0};
   int value, i, l=7;

   cout << "Enter a value";
   cin >> value;
   Search(arr,value,l,i);
    if (arr[i] != value)  cout << value << " does not exist in the l
   else  cout << value << " exists at the  " << i << "-th place";
}
```

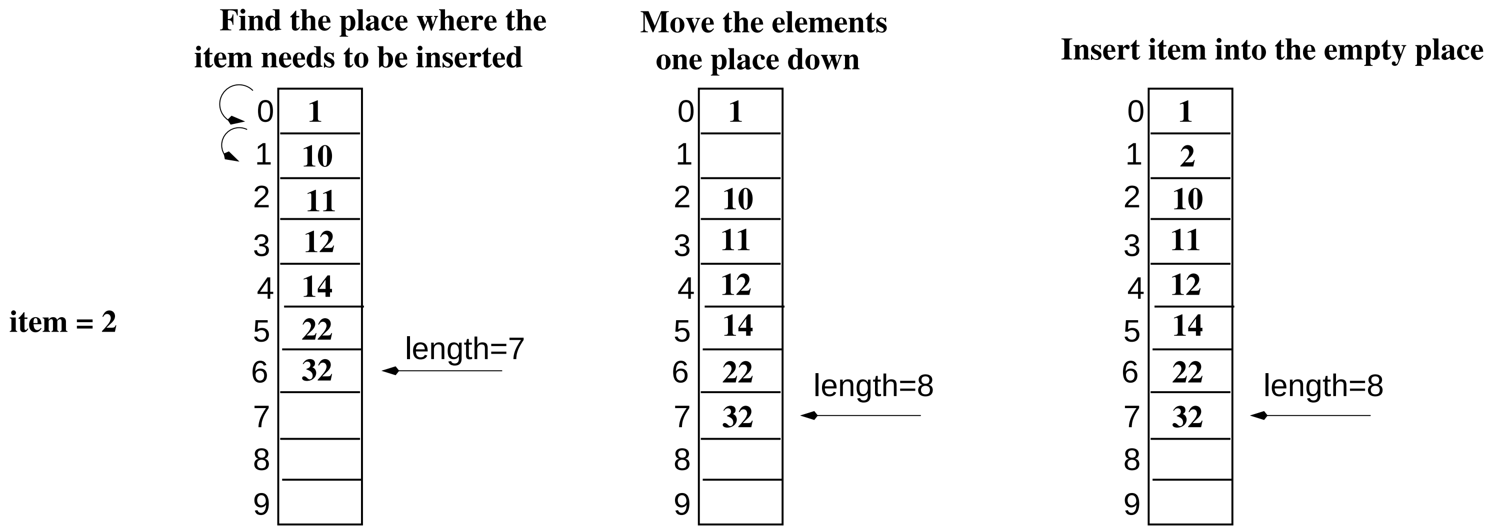
item = 2 (does not exist in the list)




```
void Seach(int list[], int item, int length, int& index)
{
   int i, found;

   i=0;
   found = 0;  // Boolean variable
   while ( (i < length) && (!found) )
    {
      if (list[i] >= item) {
         found = 1;
          index = i;
      }
      ++i
    }
  }
         ;
```

- Inserting an element (sorted list)

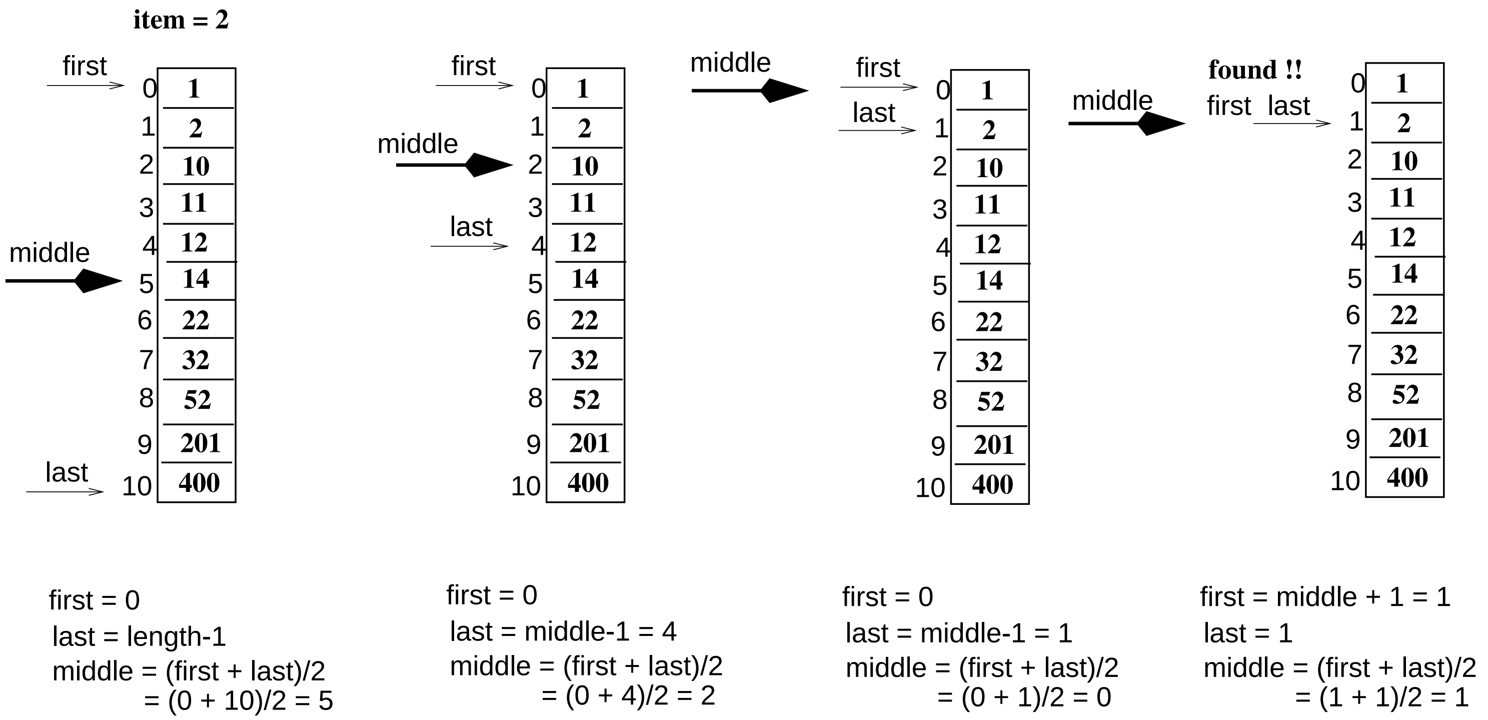
Inserting an item into a sorted list




```
void InsertSorted(int list[], int item, int length, int size)
{ int i, index;
  if (length < size) {
    Search(list, item, length, index);   // find location to insert item
    for ( i = length+1; i > index; i--)   // move elements down
       list[i] = list[i-1];
    list[index] = item;   // copy item
    ++length;
  }
  else  cout << "List is full !! Insertion failed";
}
```

- **Binary search**: an improved searching approach for sorted lists

Binary search




```
void BinarySearch(int list[], int item, int length, int& index)
{
 int first = 0;
 int last = length - 1;
 int middle, found;

 found=0;
 while( (last >= first) && (!found) )
 {
  middle = (first + last)/2; // integer division !!
  if(item < list[middle]) // reject lower part of the list
   last = middle - 1;
  else if(item > list[middle]) // reject upper part of the list
   first = middle + 1;
  else
   found = 1;

  if(found)
   index = middle;
  else
   index = length;
 }
}
```