What we have learned so far

- Straight forward statements
- Conditional statements (branching)
- Repeated statements (loops)
- Grouping statements in a subprogram (functions)

Adding Comments

- **Why is it important to write comments?**
  - **Some programmers are not very smart and write ugly codes!!**

Two types of Comments

- **Single line comment**

```
int a=5; //initialization
```

- **Multi-line comments**

```
/*
   Addition Of Two Numbers
   By Bill Gates
   © Microsoft Corporation
*/
int a=(b*c +  b*d)/b;
```

```
int a = c +  d;
```

However…..


REAL PROGRAMMERS
DON'T COMMENT THEIR CODE.
IF IT WAS HARD TO WRITE,
IT SHOULD BE HARD TO
UNDERSTAND.

Arrays

One variable many data

Problem:
Read 10 numbers from the keyboard and store them

100

Problem:

Read 10 numbers from the keyboard and store them

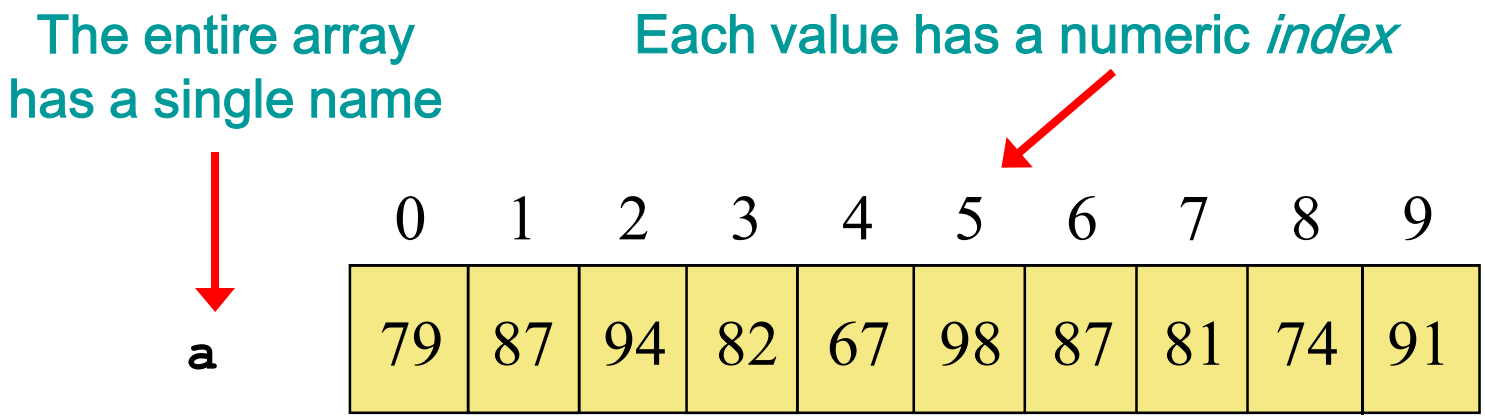
```
// solution #1
int a, b, c, d, e, f, g, h, i, j;

printf("Enter a number: ");
scanf(" %d", &a);

printf("Enter a number: ");
scanf(" %d", &b);

//…

printf("Enter a number: ");
scanf(" %d", &j);
```

Many variables

Many lines of code

Arrays

- **An *array* is an ordered list of values of similar type**



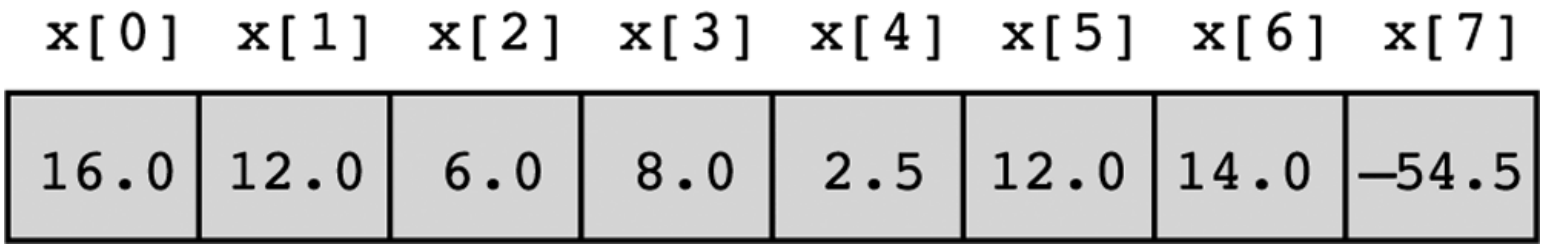
An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

An array with 8 elements of type double

```
double x[8];
```

Array x



Arrays

- **The values held in an array are called *array elements***
- **An array stores multiple values of the same type – the *element type***
- **The element type can be a primitive type**
- **Therefore, we can create an array of ints, an array of floats, an array of `doubles` , an array of `chars` .**

Declaring Arrays

```
data_type array_name[size];
```

For example:

```
int a[10];
```

a is an array of 10 integers.

```
float prices[3];
```

prices is an array of 3 floats.

```
char c[6];
```

c is an array of 6 characters.

How to assign values?

There are 3 ways.

How to assign values?

First way

- It is possible to initialize an array when it is declared:

```
float prices[3] = {1.0, 2.1, 2.0};
```

- Elements with missing values will be initialized to 0

```
float prices[9] = {1.0, 2.1, 2.0,2.3};
```

How to assign values?

First way (Continue)

- **Declaring an array of characters of size 3:**

```
char letters[3] = {'a', 'b', 'c'};
```

- **Or we can skip the 3 and leave it to the compiler to estimate the size of the array:**

```
char letters[] = {'a', 'b', 'c'};
```

How to assign values?

Second way:

- **Use assignment operator**

```
int a[6];

a[0]=3;

a[1]=6;
```

How to assign values?

Third way:

- **Use scanf to input in the array:**

```
int a[6];
scanf("%d", &a[0]);
scanf("%d", &a[1]);
    ……..
scanf("%d", &a[5]);
```

Array index could be constant, integer variable or expressions that generate integers

How to assign values?

Third way (continue):

- **Use scanf to input in the array:**

```
int a[6];

for(i= 0; i < 6; i++){

      scanf("%d", &a[i]);

}
```

Arrays: Some easy examples

- **Example 1: Suppose an array has 5 students' marks. Find average mark.**

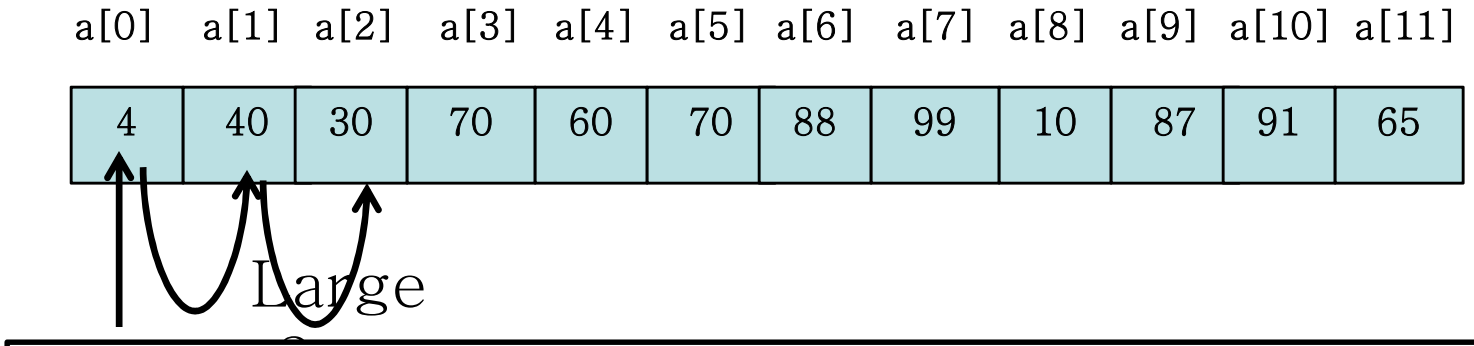
  How to accommodate N students' where N will be input to your program?

- **Example 2: Suppose an array has N students' marks. Find grade of each student.**

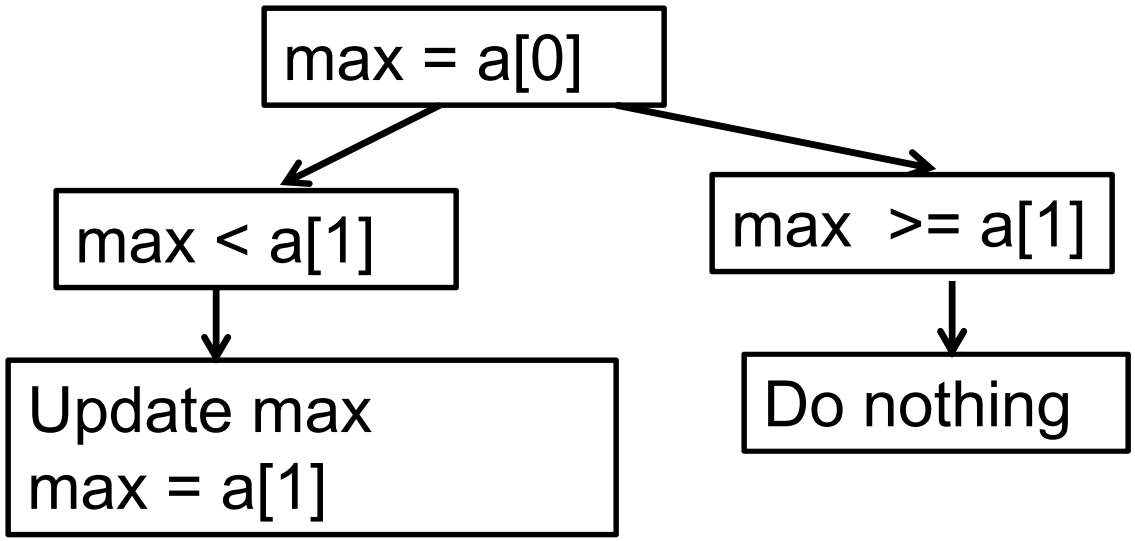
- **Example 3: Take N numbers as input and store them in an array. Print all odd numbers in the array.**

Example 4:
Find the maximum number in an array of unsorted integers

Finding maximum



Initially assume first element is the maximum



find_maximum.c

```
#include <stdio.h>
#include <stdlib.h>

#define N 12
int main()
{
   int a[N] = { 14, 21, 36, 14, 12, 9, 8, 22, 7, 81, 77, 10};
   int i;

   // Find The Maximum Element

   int max=a[0]; // pick the first number as the current maximum
   for(i=1; i< N; i++)
   {
        if(max < a[i])
        {
             max=a[i];
        }
   }

   printf("The maxiumum value in the array is %d.\n\n", max);
}
```

Example 5:
Find the maximum number (and its index) in an array of unsorted integers

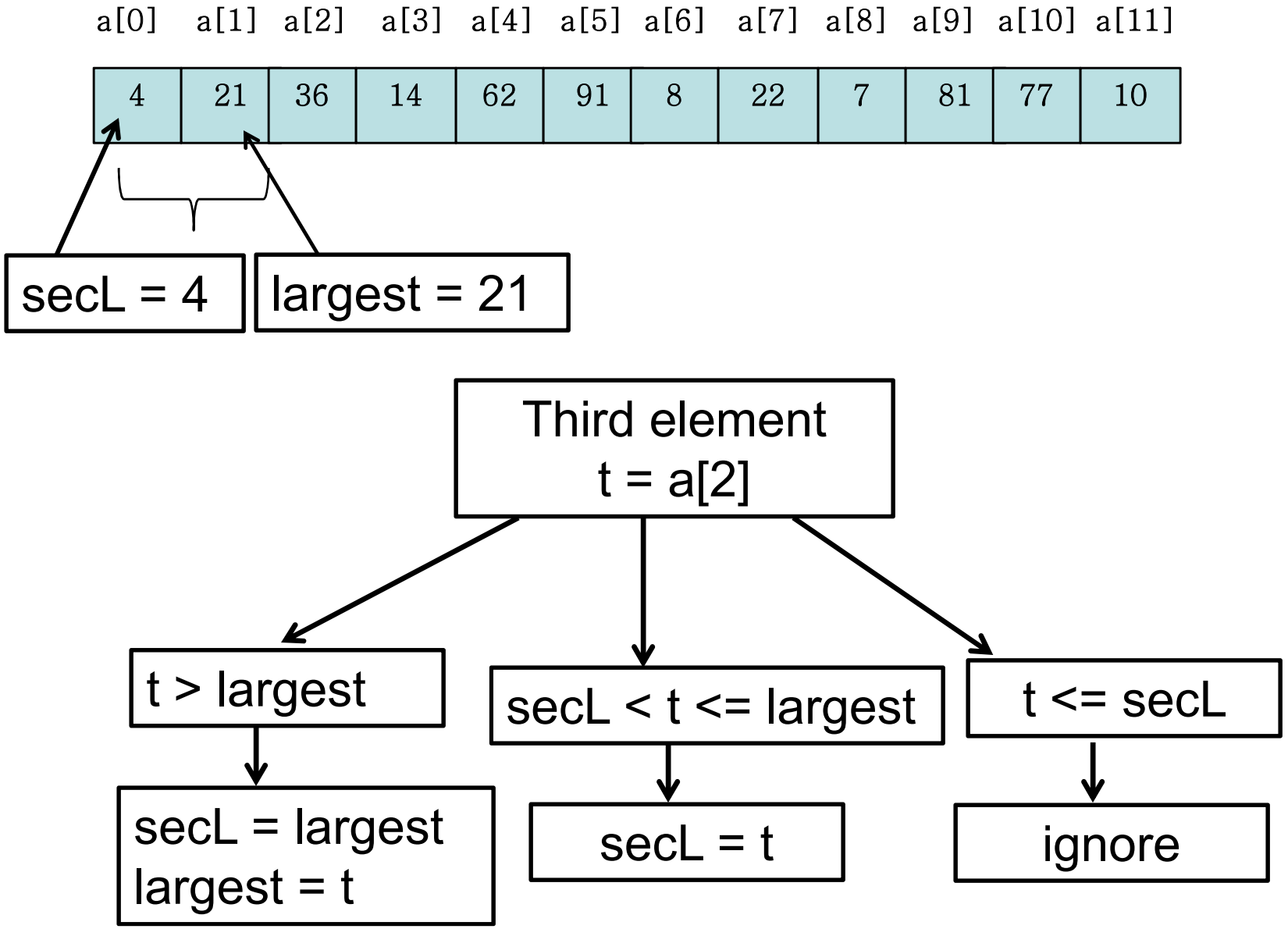
find_maximum_and_index.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N] = { 14, 21, 36, 14, 12, 9, 8, 22, 7, 81, 77, 10};
   int i, max;

   // Find The Maximum Element and it index
   max= a[0];  // initial guess: a[0] is the maximum value
   int idx=0;  // initial guess: the maximum value is at index 0

   for(i=0; i< N; i++)
   {
        if(max  < a[i])
        {
                max=a[i];
                idx=i;
        }
   }
   printf("The maximum value in the array is %d.\n\n", max);
   printf("It is located at index: %d \n\n",  idx);
}
```

Some Harder Examples

- **Print largest and second largest element of an array. Assume that the array has at least two elements.**

Largest and Second largest



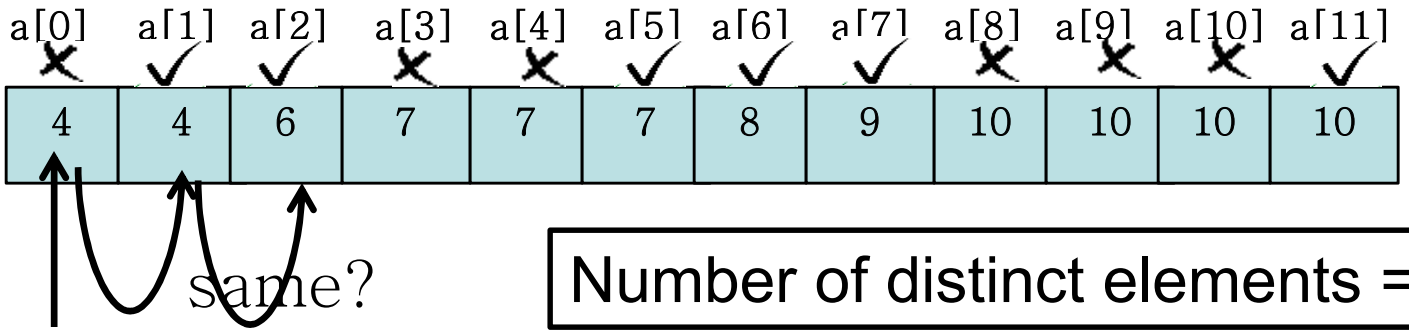
Code Snippet

```
   if(a[0] > a[1]){
        largest = a[0];
        secL = a[1];
   }
   else{
        largest = a[1];
        secL = a[0];
   }

   for(i=2; i< N; i++){
        t = a[i];
        if(t >= largest){
                secL = largest;
                largest = t;
        }
        else if (t > secL) secL = t;
   }

   printf("The largest: %d second largest: %d", largest, secL);
}
```

Some Harder Examples

- **Print the number of distinct elements in an array which is already sorted in ascending order**

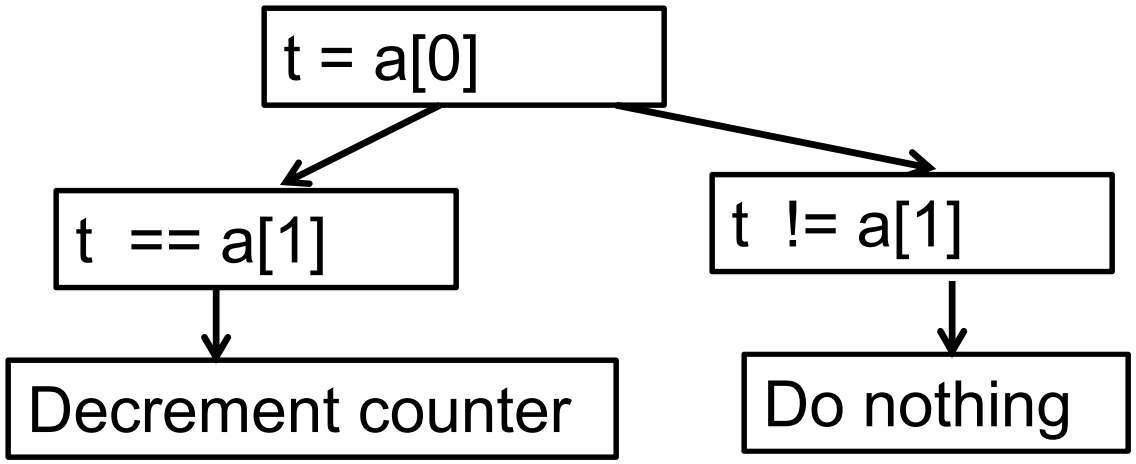
Number of distinct elements in a sorted array



Number of distinct elements = 6

Initially assume all elements are distinct

Number of distinct elements = total number of elements
= 12



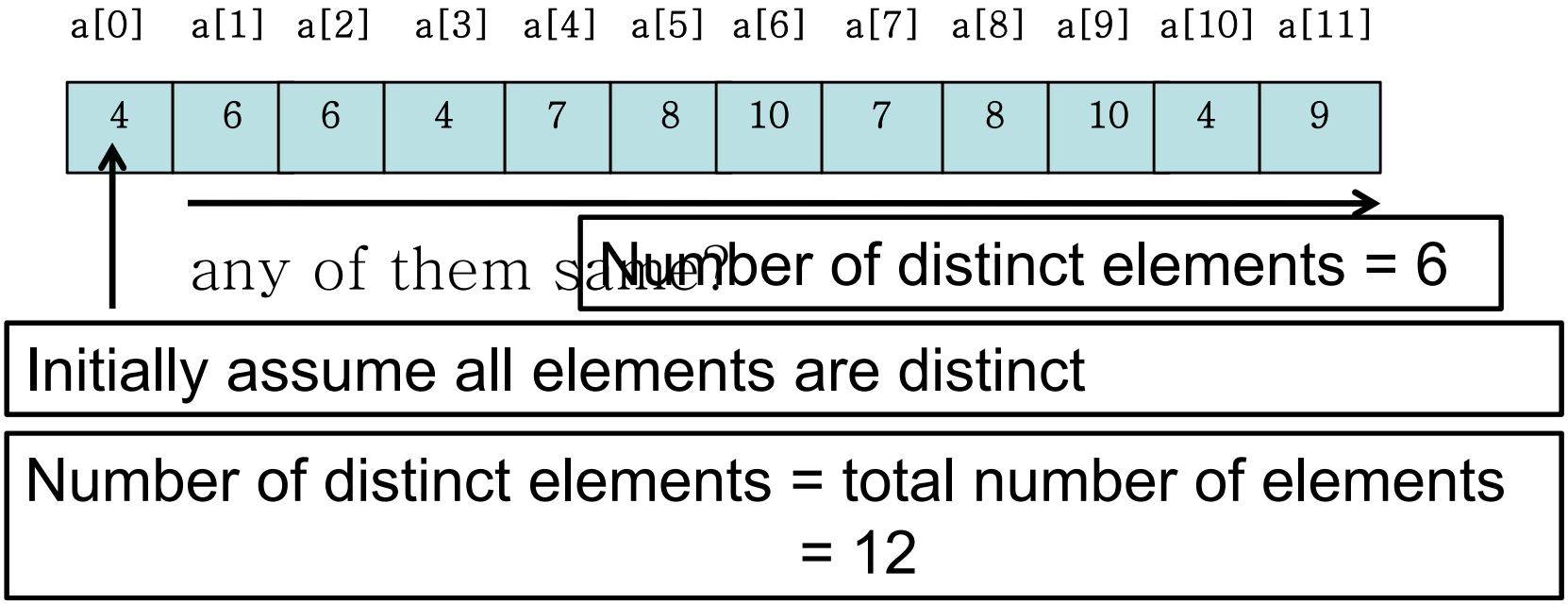
Code snippet

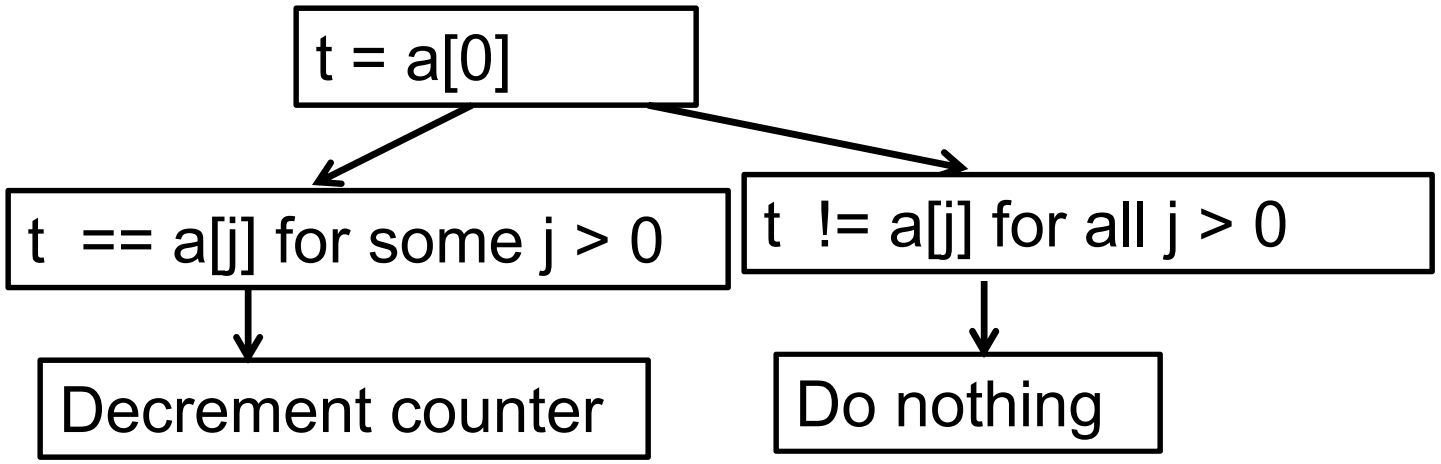
```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N] = { 4, 4, 6, 6, 7, 7, 7, 8, 9, 10, 10, 10, 10};
   int i, counter;

   counter = N;
   for(i=0; i< N-1; i++){
        if(a[i] == a[i+1])
                counter--;
   }
   printf("The number of distinct elements in the array is
   %d.\n\n", counter);
}
```

Some Harder Examples

- **Print number of distinct elements in an unsorted array**

Number of distinct elements in an unsorted array





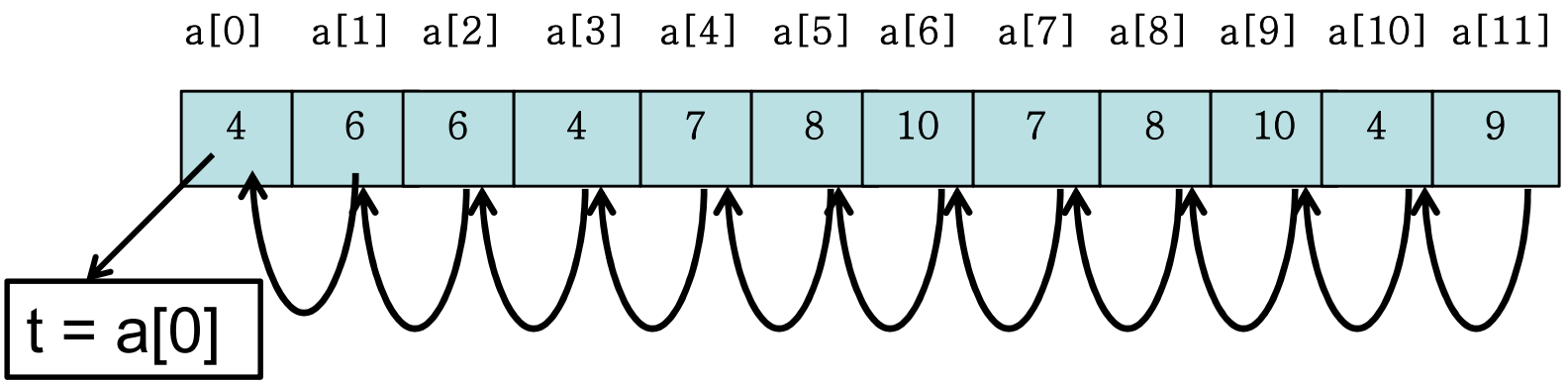
Code snippet

```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N] = { 4, 6, 6, 4, 7, 8, 10, 7, 8, 10, 4, 9};
   int i, j, counter;

   counter = N;
   for(i=0; i< N-1; i++){
        for (j = i+1; j < N; j++){
                if(a[i] == a[j]){
                        counter--;
                        break;
                }
        }
   }
   printf("The number of distinct elements in the array is
   %d.\n\n", counter);
}
```

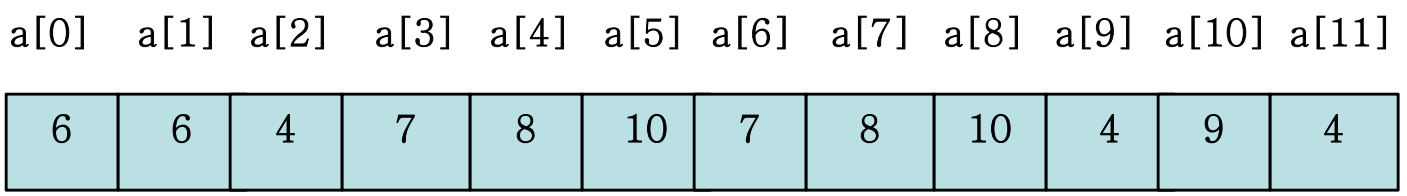
Some Harder Examples

- **Left rotate all elements of an array**

Left rotate all the elements in an array



Desired output:

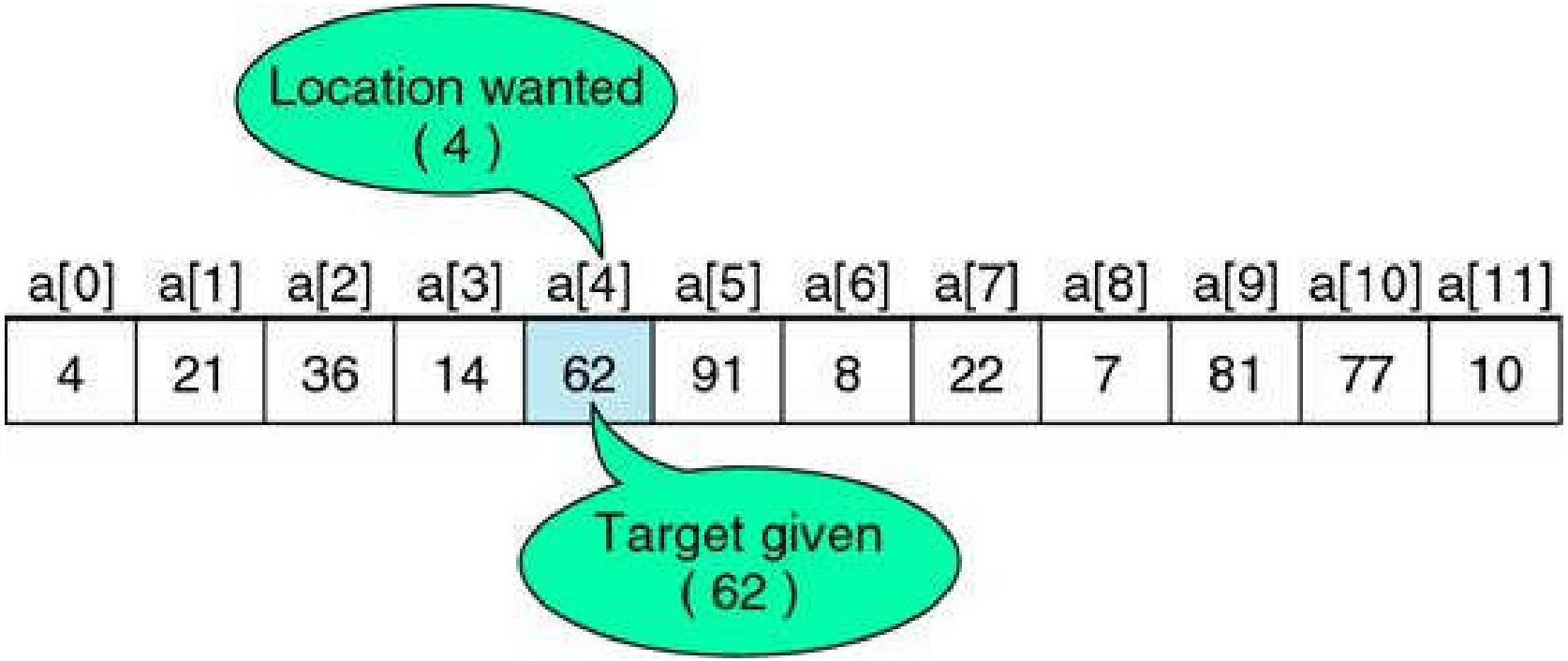


Code snippet

```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N] = { 4, 6, 6, 4, 7, 8, 10, 7, 8, 10, 4, 9};
   int i, t;

   t = a[0];
   for(i=0; i < N-1; i++)
        a[i] = a[i+1];

   a[N-1] = t;
   printf(" Array elements after left rotation……..\n");

   for(i = 0; i < N; i++)
        printf("%d\n", a[i]);
}
```

Searching

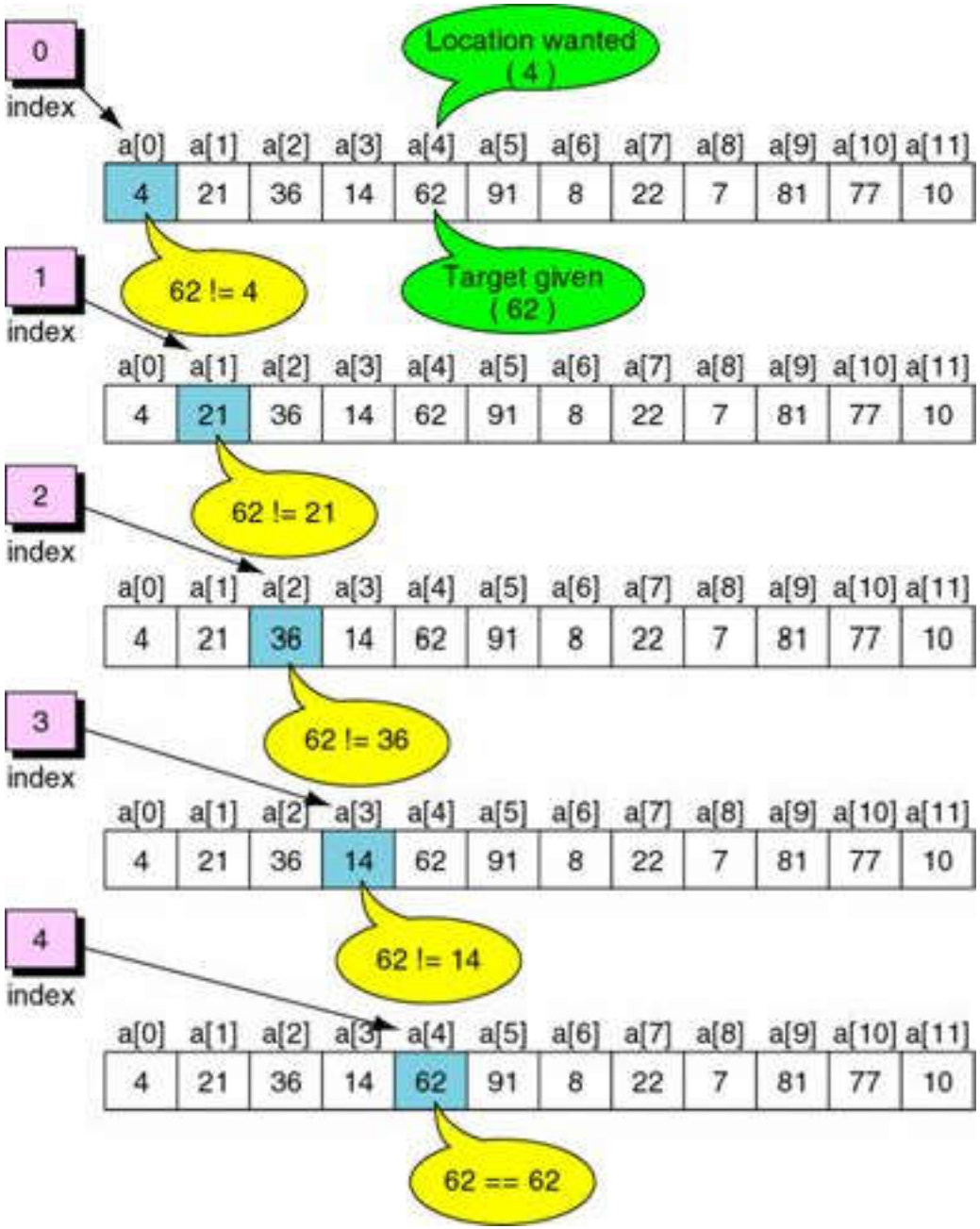
Search


Location wanted
(4)
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
Target given
(62)

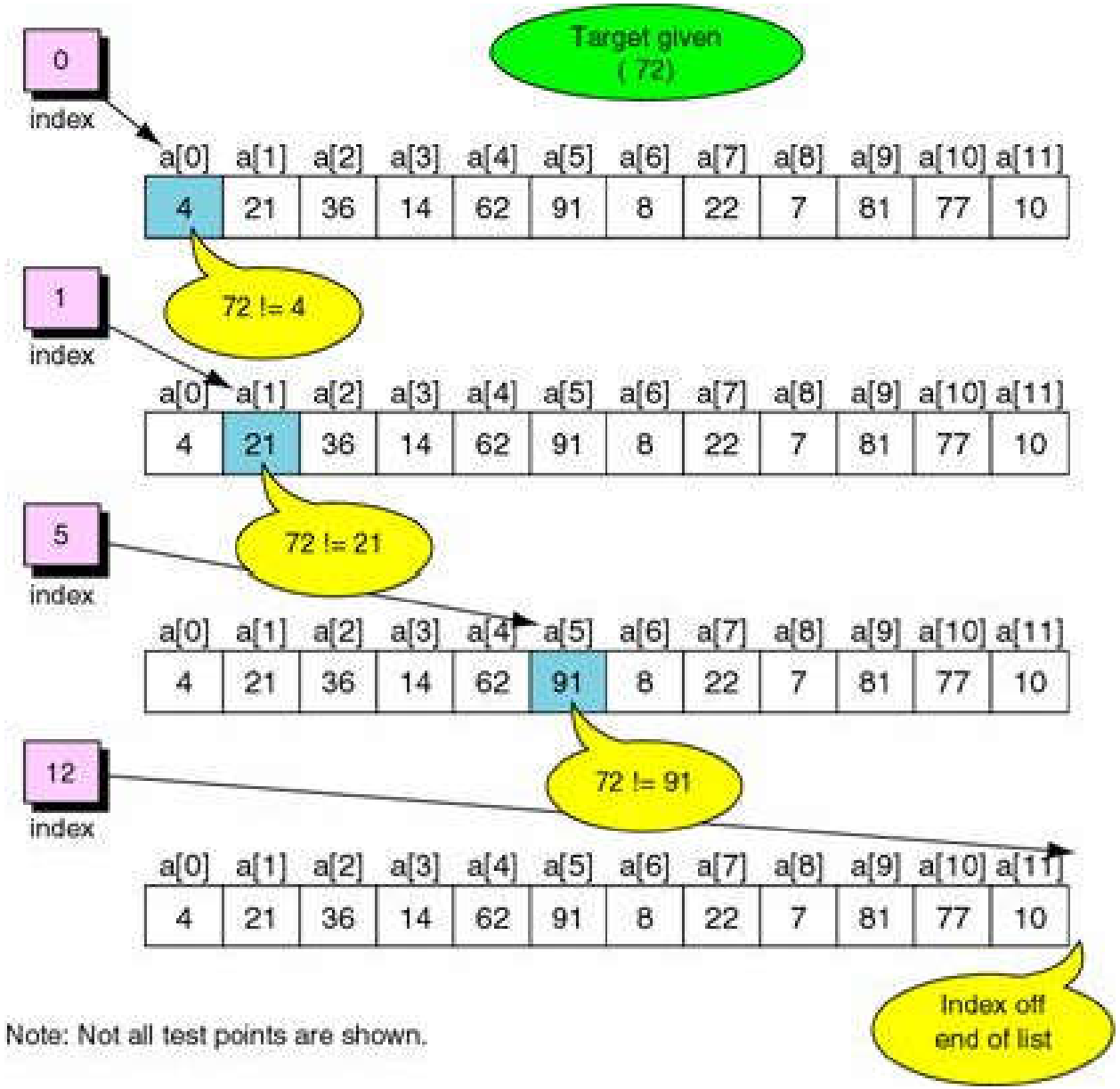
Linear Search

- **The most basic**
- **Very easy to implement**
- **The array DOESN'T have to be sorted**
- **All array elements must be visited if the search fails**
- **Could be very slow**

Example: Successful Linear Search



Example: Failed Linear Search



Note: Not all test points are shown.

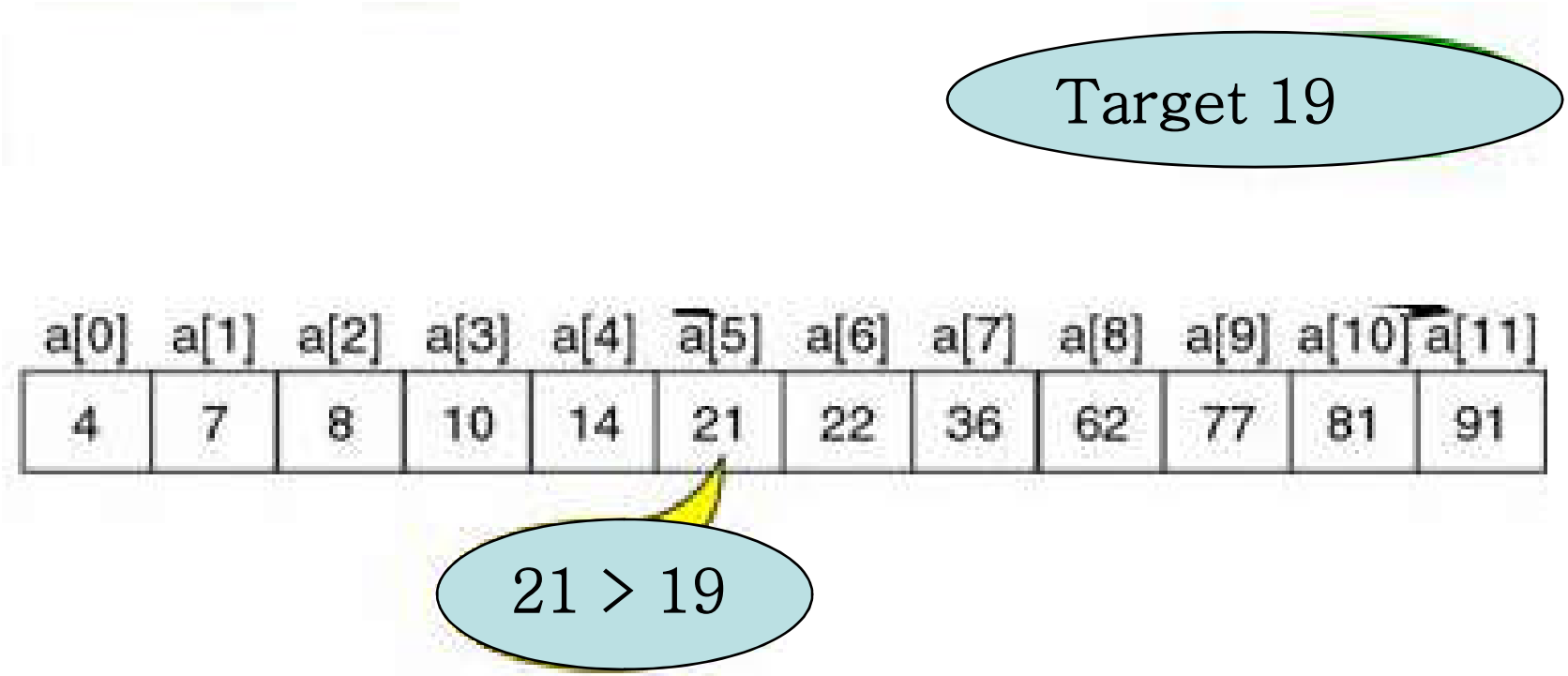
Problem:
Find the index of a number in an unsorted array of integers

linear_search.c

Linear_Search.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N] = { 4, 21, 36, 14, 62, 91, 8, 22, 7, 81, 77, 10};
   int i;

   int target = 62;  //int target = 72; // Try this next
   int idx=-1;
   for(i=0; i< N; i++)
   {
        printf(".\n");
        if(a[i] == target)
        {
                idx=i;
                break;
        }
   }
   if(idx == -1)
        printf("Target not found.\n\n");
   else
        printf("Target found at index: %d \n\n", idx);
}
```

Linear Search in a Sorted Array



Problem:
Find the index of a number in a sorted array of integers

LinearSearch_InSortedArray.c

LinearSearch_InSortedArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
   int a[N]= { 4, 7, 8, 10, 14, 21, 22, 36, 62, 77, 81, 91};

   int target = 62;    //int target = 72;// Try this target next
   int i, idx=-1;
   for(i=0; i< N; i++)
   {
        if(a[i] == target)
        {
                idx=i;
                break;
        }
        else if(a[i]>target)
                break; // we can stop here
   }
   if(idx == -1)
        printf("Target not found.\n\n");
   else
        printf("Target found at index: %d. \n\n", idx);
}
```

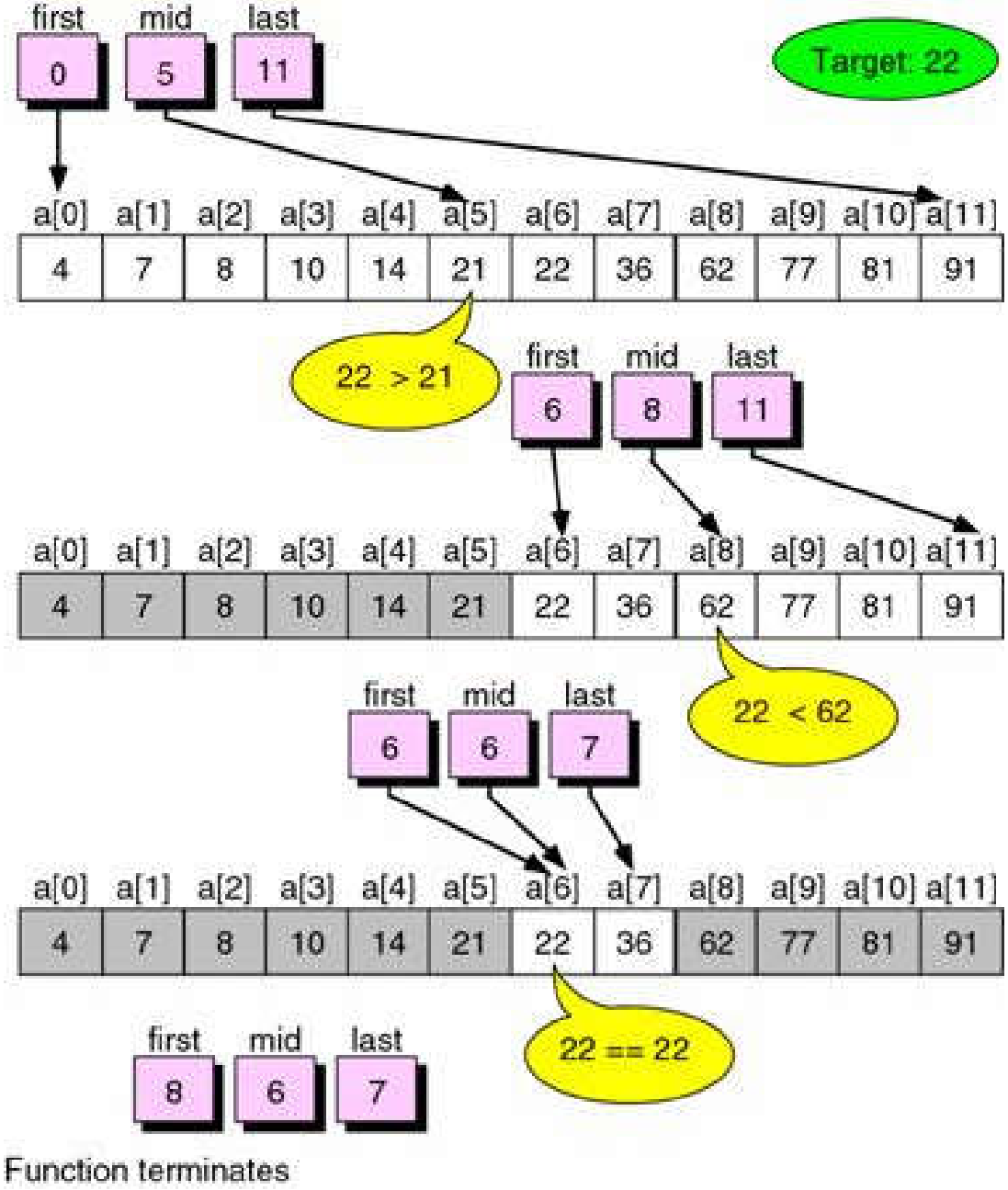
Analysis

- **If the list is unsorted we have to search all numbers before we declare that the target is not present in the array.**
- **Because the list is sorted we can stop as soon as we reach a number that is greater than our target**
- **Can we do even better?**

Binary Search

- At each step it splits the remaining array elements into two groups
- Therefore, it is faster than the linear search
- Works only on an already SORTED array
- Thus, there is a performance penalty for sorting the array

Example: Successful Binary Search



Example: BinarySearch.c

Binary_Search.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 12

int main()
{
     int a[N]= { 4, 7, 8, 10, 14, 21, 22, 36, 62, 77, 81, 91}; //sorted in increasing order
     int i;
     int target = 22;      //int target = 72; // Try this target next
     int idx=-1;           // if the target is found its index is stored here

     int first=0;          // initial values for the three search varaibles
     int last= N-1;
     int mid= (first + last)/2;

     while(last >= first)
     {
             if( a[mid] == target)
             {
                          idx=mid;  // Found it!
                          break;    // exit the while loop
             }
             else if(a[mid] > target)
             {
                          // don't search in a[mid] ... a[last]
                          last = mid-1;
             }
             else
             {
                          // don't search in a[first] ... a[mid]
                          first = mid +1;
             }

             // recalculate mid for the next iteration
             mid = (first + last)/2; // integer division!

     } // end of while loop

     if(idx == -1)
             printf("Target not found.\n\n");
     else
             printf("Target found at index: %d \n\n", idx);
}
```

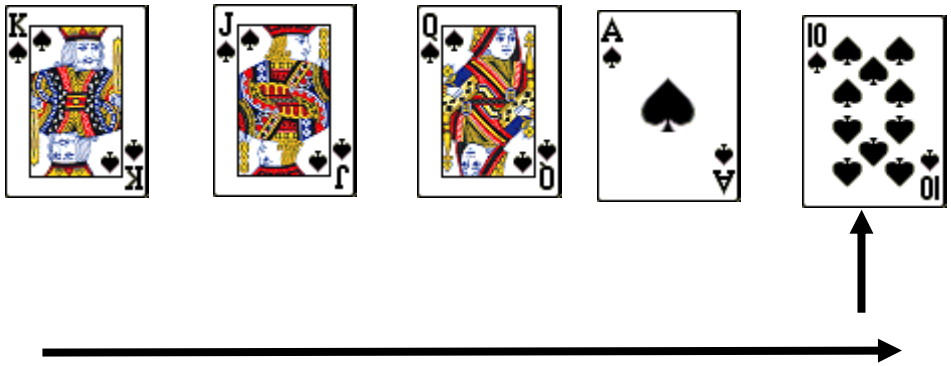
Problem:
Find the all occurrences of a number in an array and replace it with a new value.

search_and_replace.c

Linear_Search.c

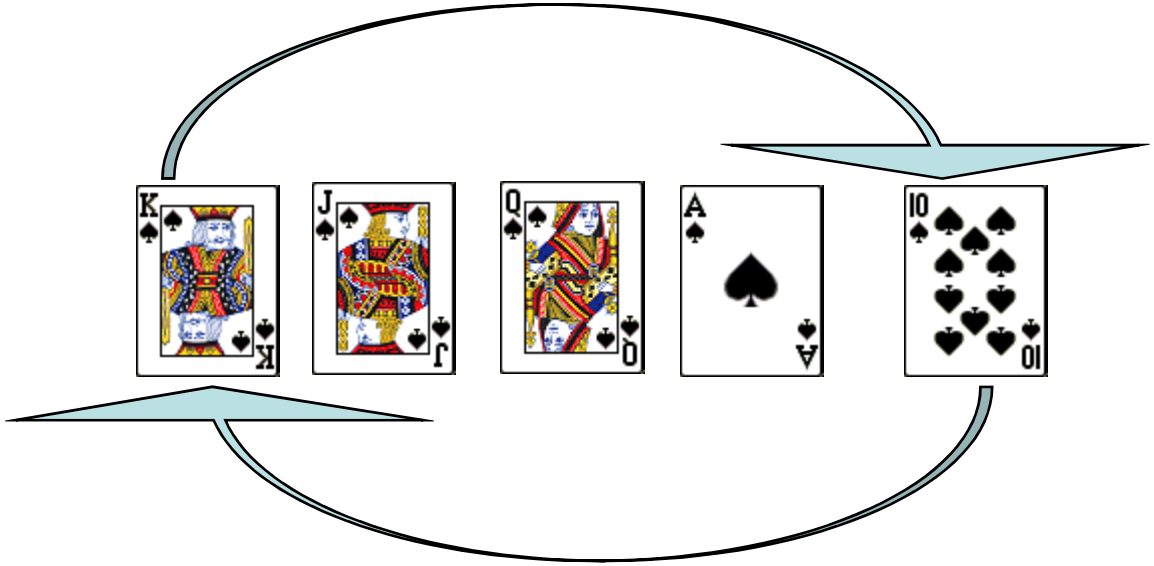
```
#include <stdio.h>
#include <stdlib.h>
#define N 12
int main()
{
    int a[N] = { 4, 21, 36, 14, 62, 91, 8, 22, 7, 81, 62, 10};
    int i;
    int target = 62;
    int newValue = 65;

    int count=0;
    int idx[5];  // a helper array that keeps the indexes of all entries == target value
    int found=0;

    for(i=0; i< N; i++)
    {
            if(a[i] == target)
            {
                        found = 1;
                        idx[count] = i;
                        count++;
            }
    }

    if(found == 0)
            printf("Not found!\n\n");
    else
    {
            printf("Found it a total of %d times.\n", count);
            for(i=0; i< count; i++)
                        printf("\t Found @ index %d \n", idx[i]);
    }
    // Now replace all found occurences with a nother number
    for(i=0; i< count; i++)
            a[ idx[i] ] = newValue;

    system("pause");
}
```

Selection Sort (Cards Example)

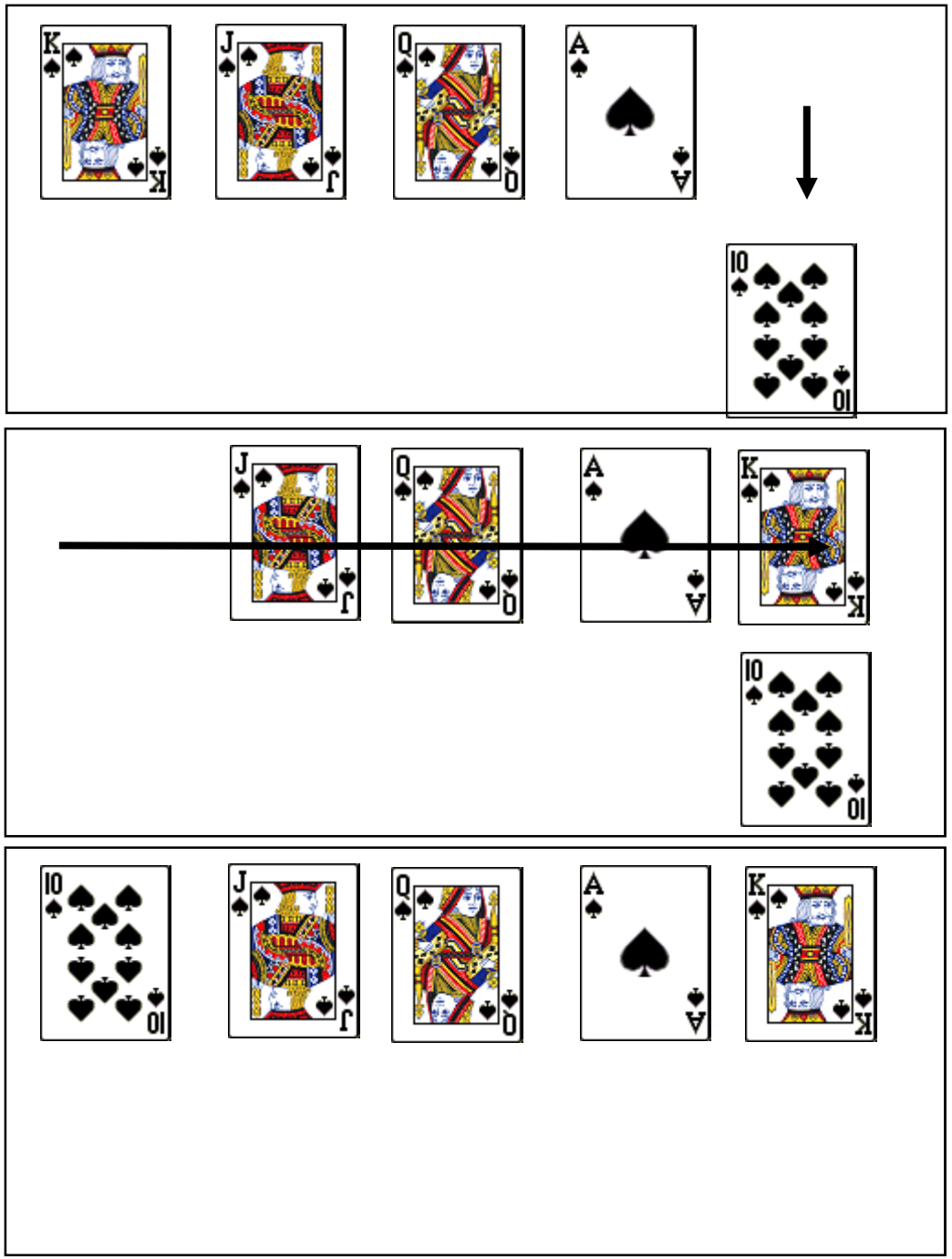


Initial Configuration

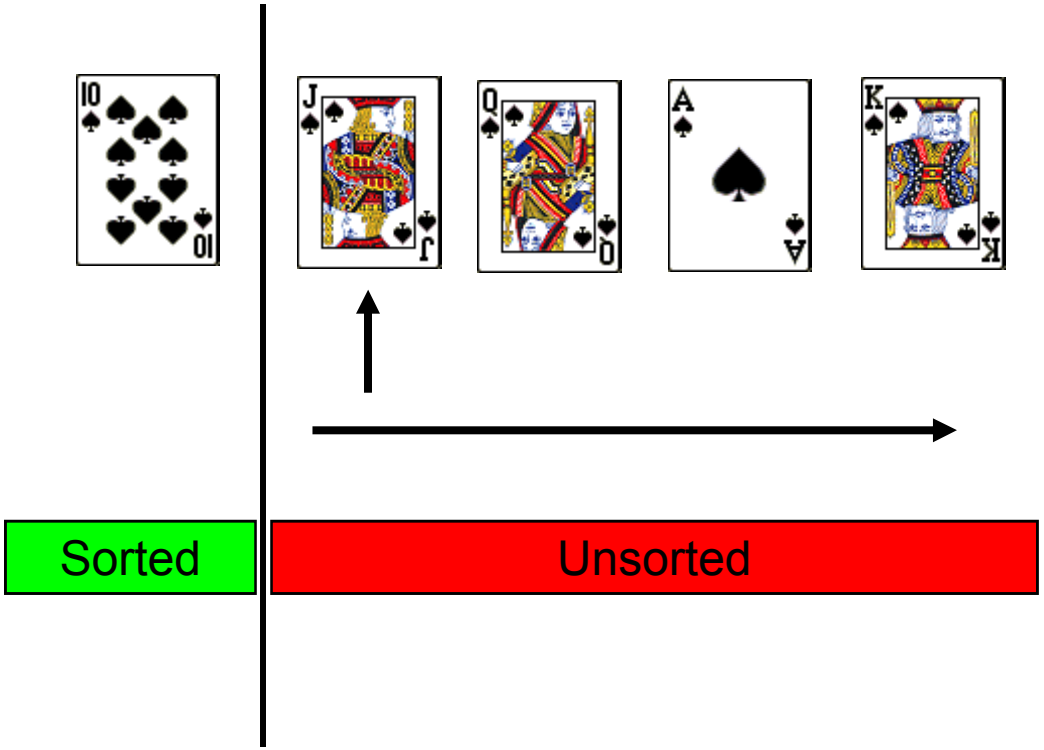
(search all cards and find the smallest)



Swap the two cards



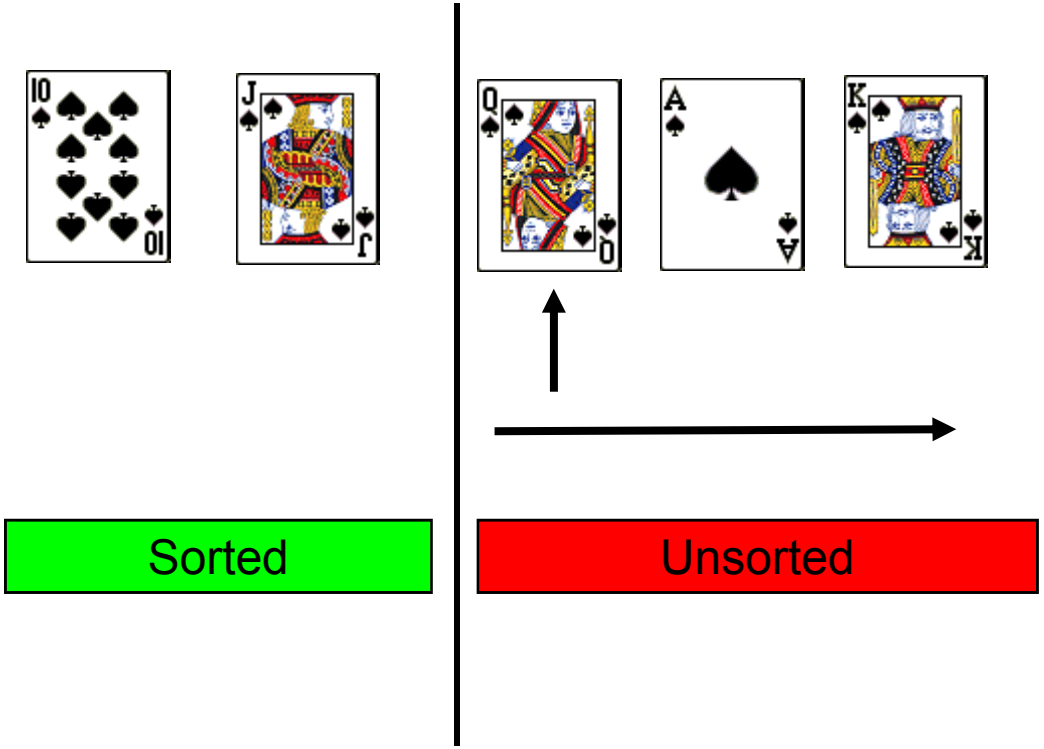
As before, the swap is performed in three steps.



Among the remaining cards
the Jack is the smallest.

It will remain in place.

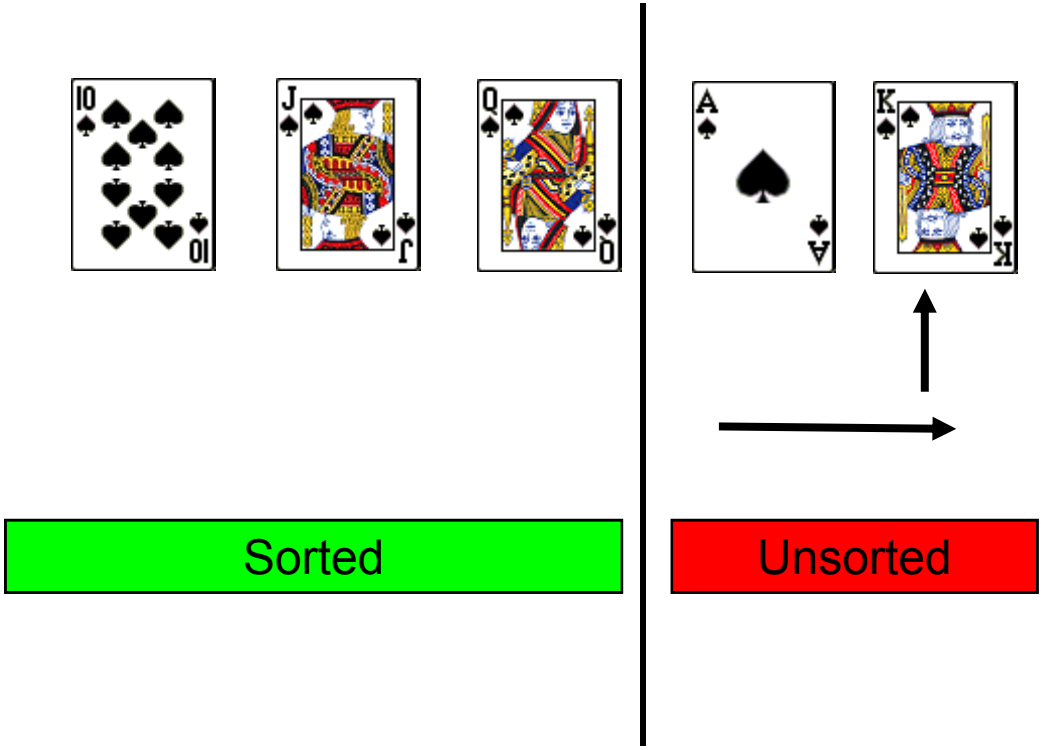
But the algorithm may perform
Some empty operations
(ie., swap it with itself in place)



Among the remaining cards
the queen is the smallest.

It will remain in place.

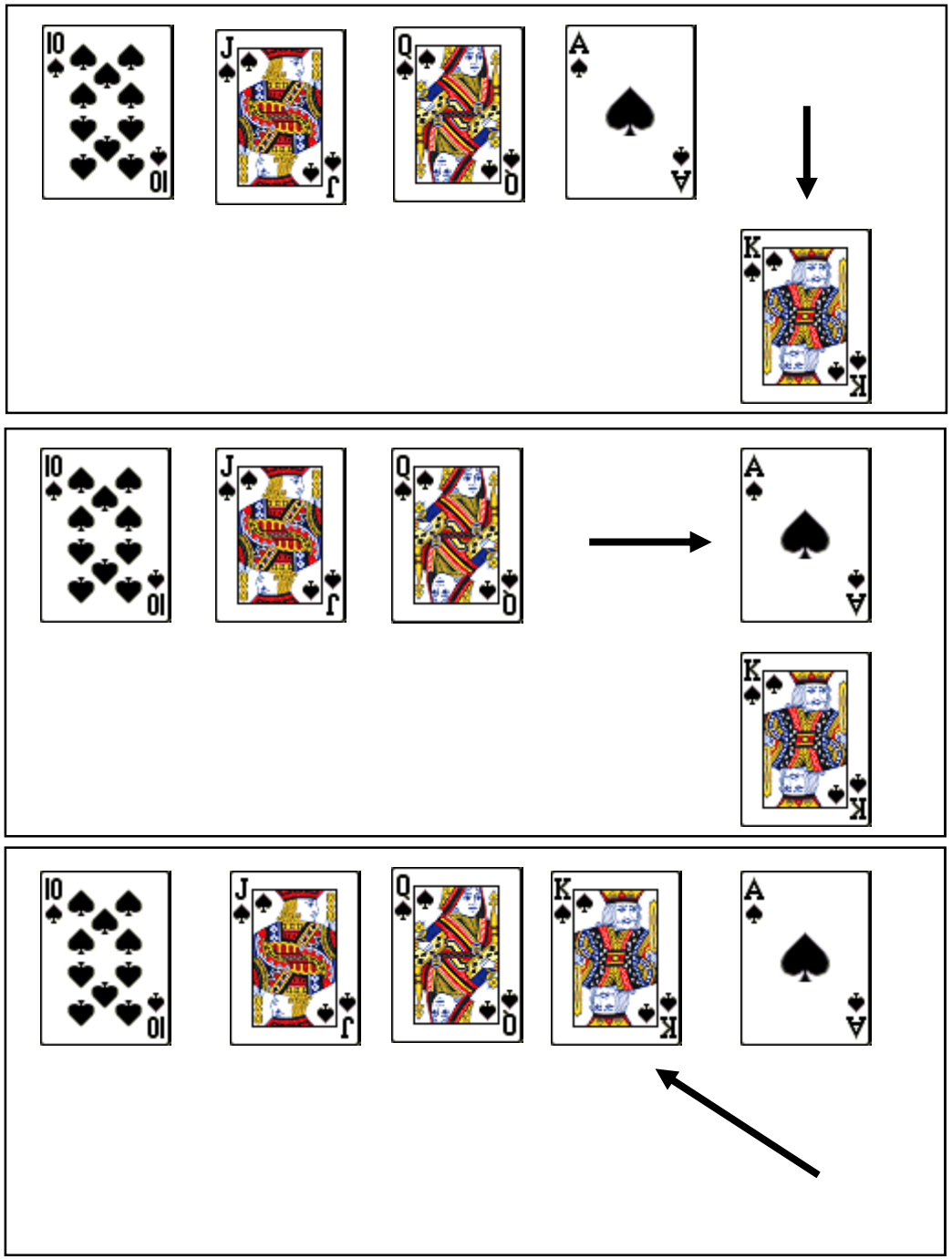
But the algorithm may perform
Some empty operations
(i.e., swap it with itself in place)



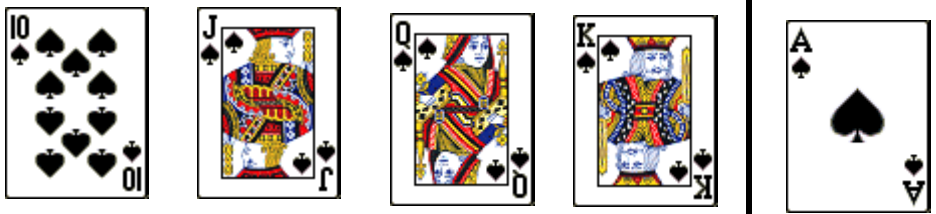
Among the remaining cards
the king is the smallest.

It will remain in place.

But the algorithm may perform
Some empty operations
(i.e., swap it with itself in place)



As before, the swap is performed in three steps.



Sorted

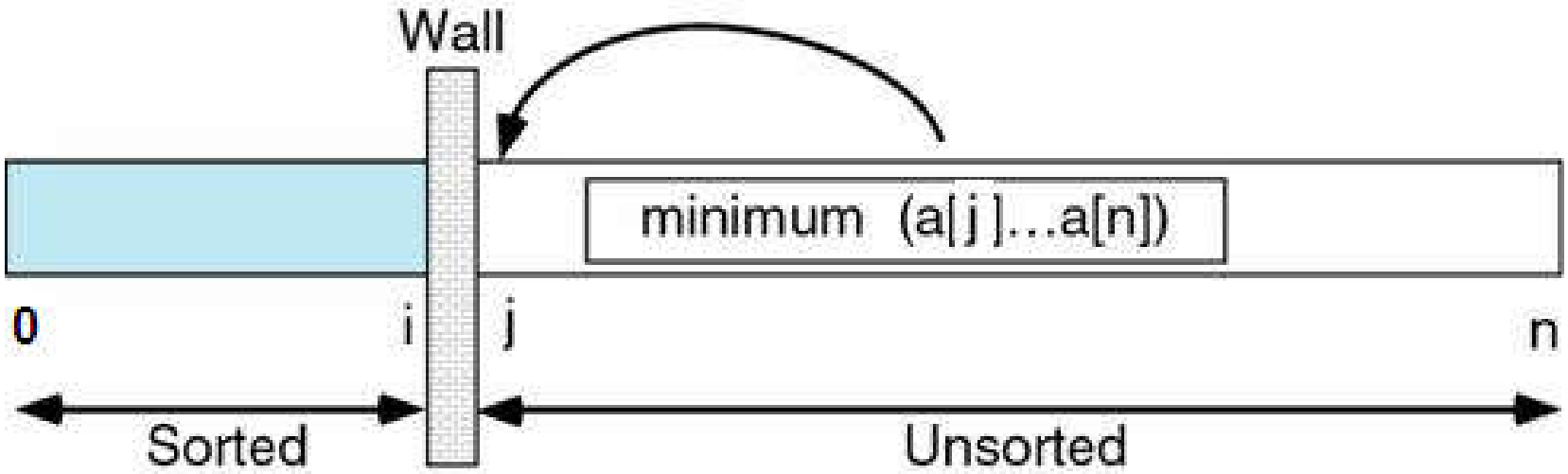
Unsorted

We are down to the last card.
Because there is only one and
Because we know that it is
Smaller than all the rest
We don’t need to do anything
Else with it. This is why the
Algorithm goes up to < N-1

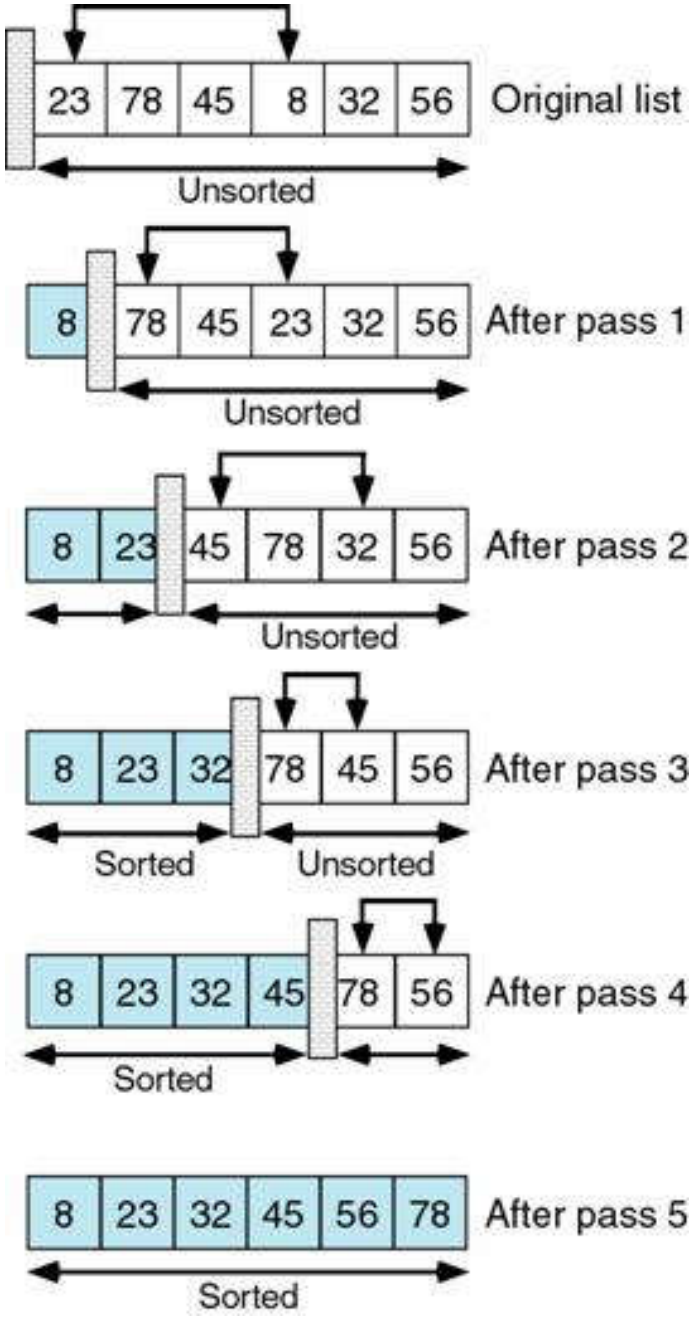
All cards are now sorted.

Sorted

Selection Sort



Example: Selection Sort



Example: SelectionSort.c

```
#include <stdio.h>
#define N 6
int main()
{
   int a[N]= { 23, 78, 45, 8, 32, 56};
   int i,j,tmp;
   // Sort the array using Selection Sort
   int idx,min;
   for(i=0; i < N-1; i++)
   {
        min=a[i];
        idx = i;
        for(j=i+1; j < N; j++)
        if(a[j] < min){
               idx = j;
               min = a[j];
             }
        tmp = a[i];
        a[i] = min;
        a[idx] = tmp;
   }
   for(i = 0; i < N; i++)
         printf("%d\n",a[i]);
}
```

Questions?