Topic 4

1. Arrays
2. Common array algorithms
3. Arrays / functions
4. <u>Problem solving: adapting algorithms</u>
5. Problem solving: discovering algorithms
6. 2D arrays
7. Vectors
8. Chapter Summary

Problem Solving: Adapting Algorithms

Recall that you saw quite a few
(too many?)
algorithms for working with arrays.

Suppose you need to solve a problem that does not exactly fit any of those?

What to do?

No, “give up” is not an option!

You can adapt algorithms you already know to produce a new algorithm.

Problem Example: Summing Quiz Scores

Consider this problem:

Compute the final quiz score from a set of quiz scores,

but be nice:

drop the lowest score.

Adapting Algorithms: Three that We Know

Calculate the sum:

```
double total = 0;
for (int i = 0; i < size of values; i++)
{
   total = total + values[i];
}
```

Find the minimum:

```
double smallest = values[0];
for (int i = 1; i < size of values; i++)
{
   if (values[i] < smallest)
   {
      smallest = values[i];
   }
}
```

Remove an element:

```
values[pos] = values[current_size - 1];
current_size--;
```

Adapting Algorithms: A Glitch in Combining Those Three

```
  values[pos] = values[current_size - 1];
  current_size--;
```

This algorithm removes by knowing *the position* of the element to remove… …but…

```
 double smallest = values[0];
  for (int i = 1; i < size of values; i++)
  {
     if (values[i] < smallest)
     {
        smallest = values[i];
     }
  }
```

That’s not the *position* of the smallest – it IS the smallest.

Algorithm to Find the Position

Here’s another algorithm I know that *does* find the position:

```
int pos = 0;
bool found = false;
while (pos < size of values && !found)
{
    if (values[pos] == 100) // looking for 100
    {
        found = true;
    }
    else
    {
        pos++;
    }
}
```

Adapting the Minimum Algorithm to Report the Position

Combining the minimum value algorithm with the position-finder:

```
int smallest_position = 0;
for (int i = 1; i < size of values; i++)
{
    if (values[i] < values[smallest_position])
    {
        smallest_position = i;
    }
}
```

Final Answer for Adapting Algorithms

Aha! Here is the algorithm:

1. *Find the* ***position*** *of the minimum*
2. *Remove it from the array*
3. *Calculate the sum*

(will be without the lowest score)

4. *Calculate the final score*

Topic 5

1. Arrays
2. Common array algorithms
3. Arrays / functions
4. Problem solving: adapting algorithms
5. Problem solving: discovering algorithms
6. 2D arrays
7. Vectors
8. Chapter Summary

Discovering Algorithms by Manipulating Physical Objects

What if you come across a problem
for which you cannot find an algorithm you know
and you cannot figure out how to adapt any algorithms?

you can use a technique called:

MANIPULATING PHYSICAL OBJECTS

better know as:

playing around with things.

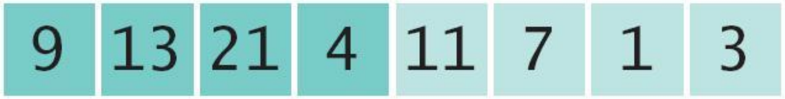
Big C++ by Cay Horstmann

Manipulating Physical Objects: Example Problem

Here is a problem:

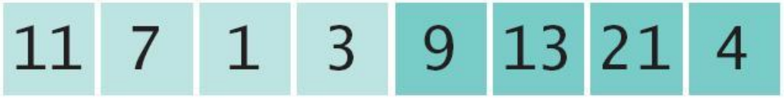
You are given an array whose size is an even number.
You are to switch the first and the second half.

Before:



9	13	21	4	11	7	1	3

After:

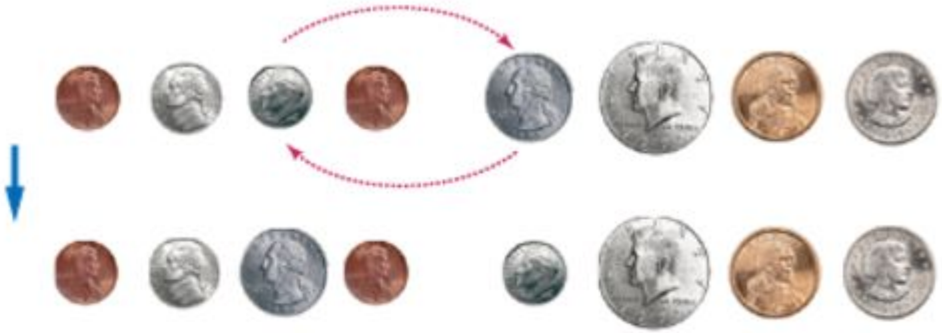


11	7	1	3	9	13	21	4

Manipulating Physical Objects: Coins

We'll use 8 coins as a model for our 8-elements of the array

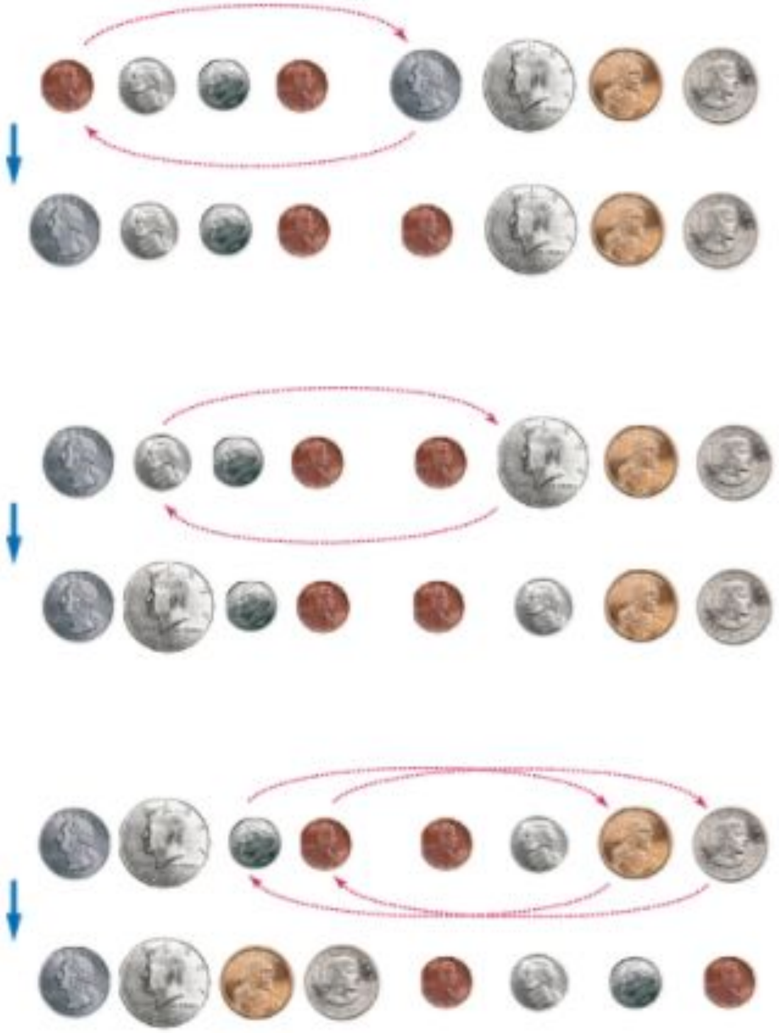
We can swap coins like we'd swap array elements:



Big C++ by Cay Horstmann

Swapping Coins: the Algorithm

- We find that by swapping the
  - 1st and 4th coins, and
  - 2nd and 5th
  - 3rd and 6th
  - And 4th and 8th
  - We have swapped the first half of the 8 with the last



Translating the Manipulations to Code

Pseudocode:

```
i = 0
j = size / 2
While i < size / 2
    Swap elements at positions i and j.
    i++
    j++
```

Translating to C++ is left as a Programming Exercise at the end of the chapter

Self Check: Practice Manipulating Objects

Using physical objects such as coins to represent array elements, determine the purpose of the function below:

```
void transform(int array[], int length)
{
   int position = 0;
   for (int k = 1; k < length; k++)
   {
      if (array[k] < array[position])
      {
         position = k;
      }
   }
   int temp = array[position];
   while (position > 0)
   {
      array[position] = array[position - 1];
      position--;
   }
   array[0] = temp;
}
//ANSWER: copies the smallest value to the first array
location and shifts other elements so no values are lost
```