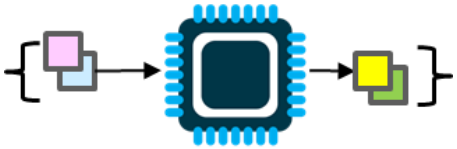
Unit 3b – Array and Loop Tasks

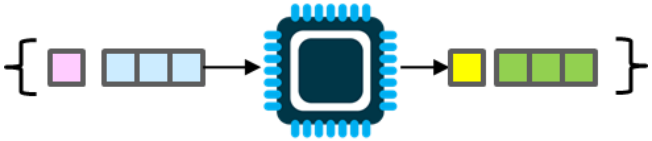
Mark Redekopp

Unit 3

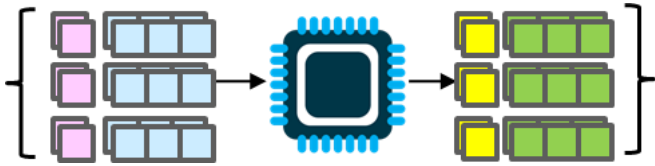
- **Unit 1**: Scalar processing
  - aka IPO=Input-Process-Output Programs



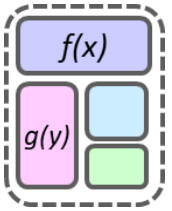
- **Unit 2**: Linear (1D) Processing



- **Unit 3**: Multidimensional Processing



- **Unit 4**: Divide & Conquer (Functional Decomposition)



Algorithmic Thinking

- Informal definition of **algorithm:**
  - **A precise way to accomplish a task or solve a problem**
- The skill we REALLY want to help you build is **algorithmic thinking** (i.e. computational problem solving)

- *We will just try to work as many example as possible, but you need to be mentally engaged and trying to solve these tasks before and while we go through them together and then reflect and extract strategies afterwards.*

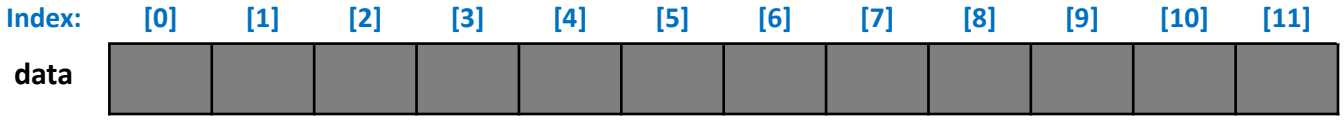
Implementation

- Algorithm Discovery:
  - Solve the problem yourself for several examples
  - Reflect on what your thought process was
  - Given a computer can only do 1 operation (on two values) at a time, what **variables** do you need to remember past results and what **loops** are necessary to perform that 1 operation many times
- Implementation / Programming mechanics
  - Can we achieve our task in one pass (loop), sequential loops, or need nested loops
  - Can we stop early? And how do we implement that (break statements, etc.)

Task 1

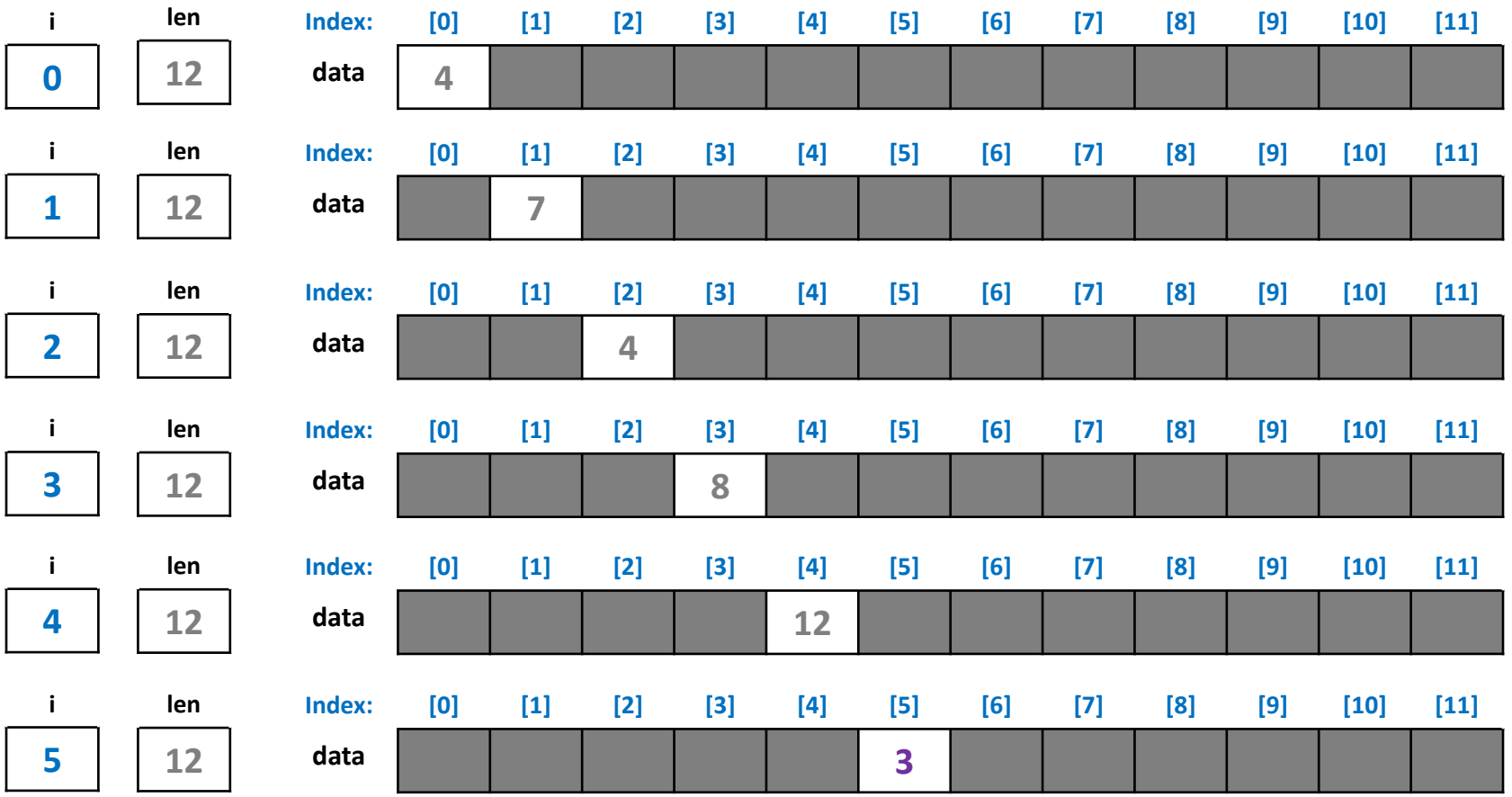
- Let the user input a **value** and **find** the first occurrence of that **value** in the array and output its index, or -1 if it does not exist
- Questions:
  - Could it be anywhere?
  - Is there any intelligent way to narrow it down?

If the user enters **3**, find **3** and return its index or -1:



Task 1

If the user enters 3, find 3 and return its index or -1:



Task 1

- What programming issues (mechanics) should you think about?
  - How would you generate the appropriate indexes?
  - When can you stop?
  - When would you be ready to return -1?

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 2

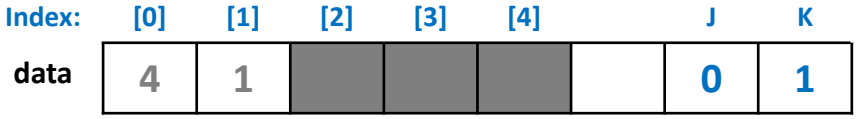
- Output all possible pairs of numbers `(a,b)`
  - Order does NOT matter [Don't output `(b,a)` after outputting `(a,b)`]
- Questions:
  - How many pairs are there?
    - How many `(4,x)` pairs? `(1,x)` pairs? `(7,x)` pairs? `(8,x)` pairs? `(12,x)` pairs?
  - Can we do this in 1 pass?

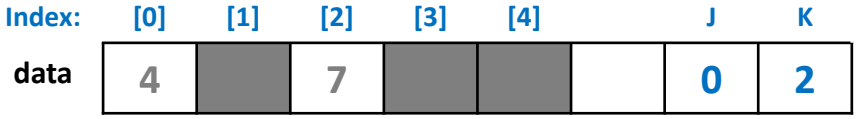
Output all pairs of values from the array:

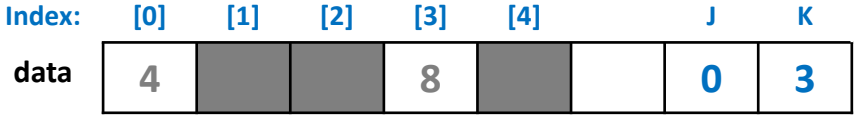
Index:	[0]	[1]	[2]	[3]	[4]
data	4	1	7	8	12

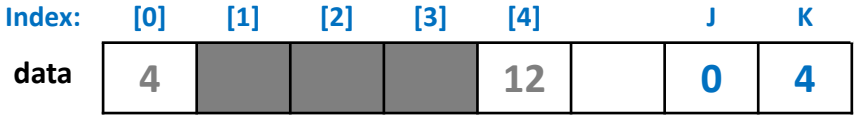
Task 2

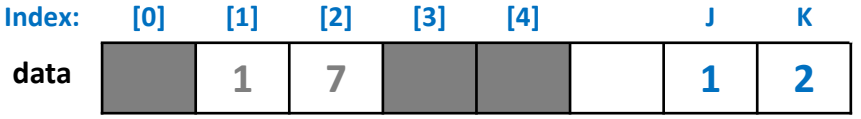
Output all pairs of values from the array:

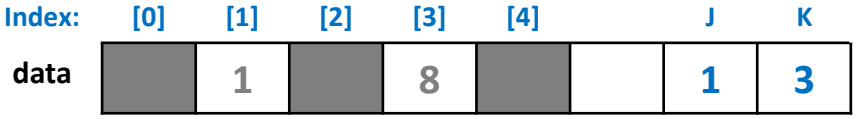


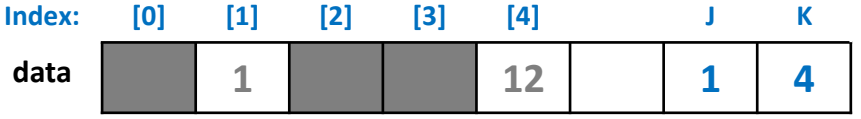


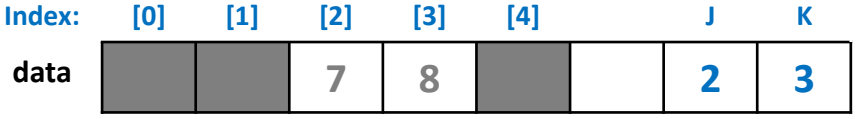


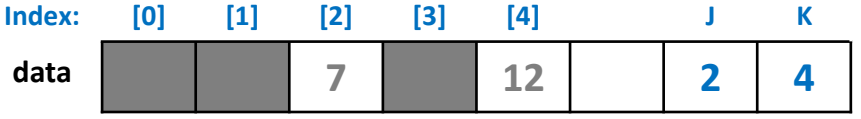


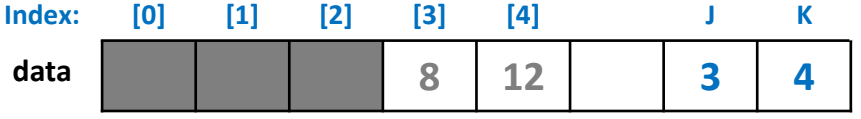












Task 2

- What programming issues (mechanics) should you think about?
  - How would you generate the appropriate indexes?
  - Are the indexes independent or is one dependent on another?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 3

- Check if all the numbers in an array are unique
- Seems easy enough for a human on the examples below

Check if all number are unique: NO

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	4	1	7	8	12	7	6

Check if all number are unique: YES

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	4	1	10	8	12	7	6

What if we can only see 1 or 2 numbers at a time.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data							

Task 3

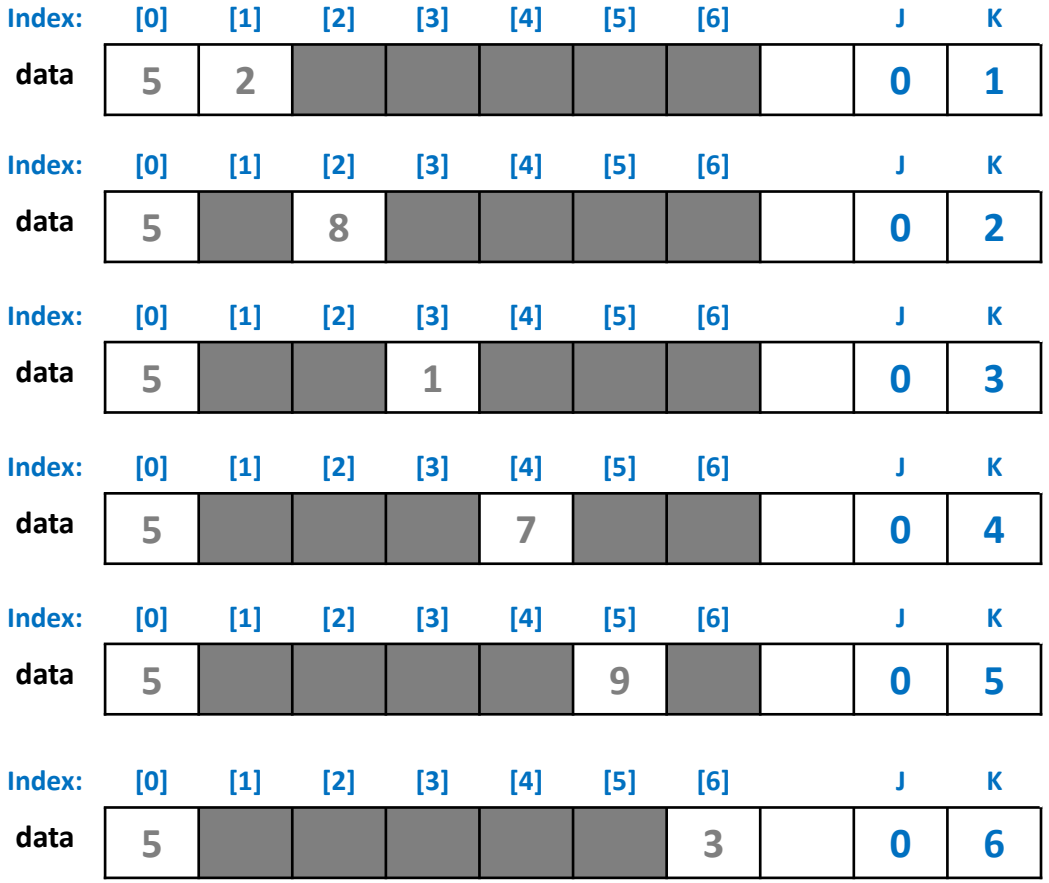
- Check if all the numbers in an array are unique
- It may not be as easy even for a human when we increase the amount of numbers?
- Questions:
  - What process would you use as a human? Can we do this in 1 pass?
  - If we can only see 1 thing or perform 1 operation at a time, what other variables do we need?
  - Do we always have to do the same amount of work, or might we find an answer "early"?

Sample Data:

```
6 40 3 96 44 94 74 9 23 22 56 64 12 7 51 31 24 80 88 10
91 27 38 30 78 60 37 69 26 11 39 50 68 21 41 48 66 46
20 25 82 98 76 34 55 70 4 54 90 28 14 71 73 85 81 65 77
59 57 43 33 49 87 19 17 16 1 2 15 72 45 93 86 92 36
```

Task 3

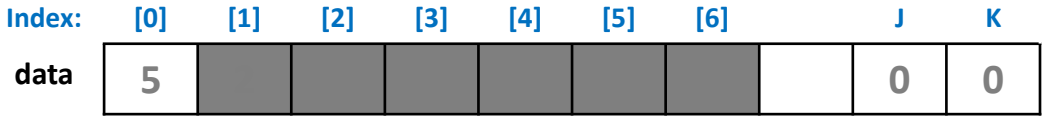
Check if all number are unique:



We've seen all the numbers now? Can we state they are all unique? What are you "remembering" to answer that question? Can the computer do that?

Task 3

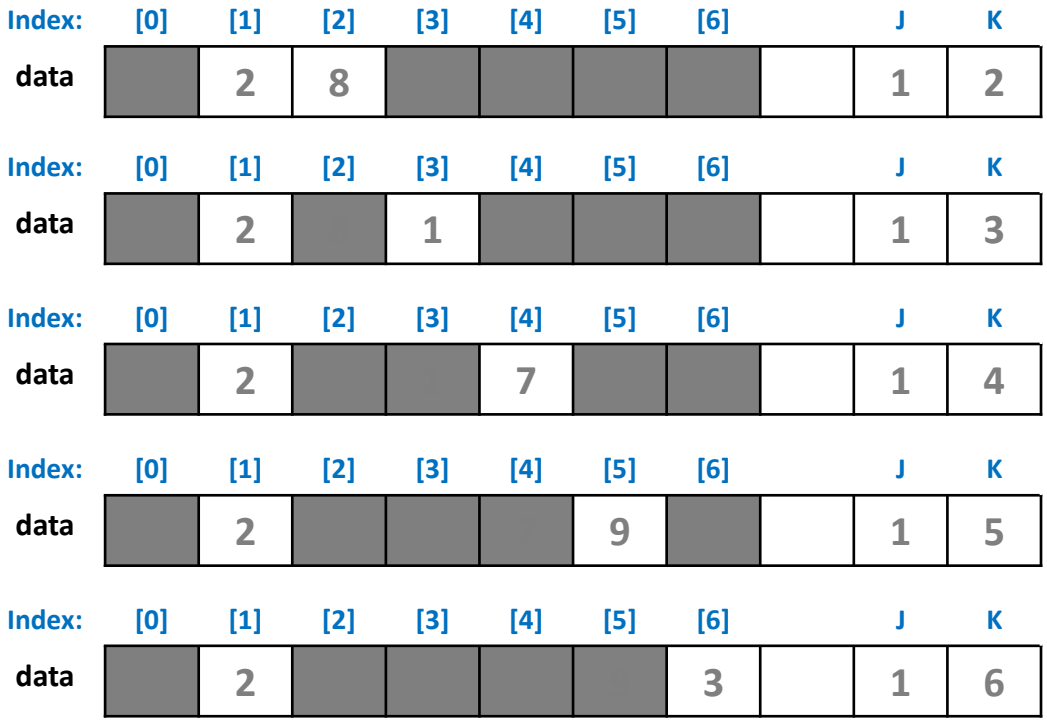
- Why not start k at 0 as well?
  - We never want to compare an element with itself




```
data[j] == data[k]
```

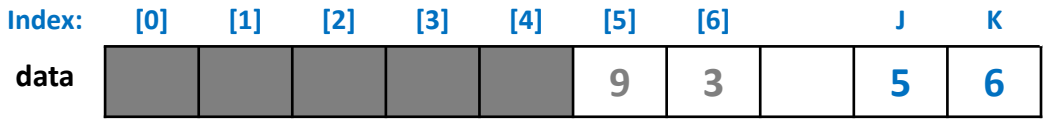
Task 3

Check if all number are unique:



Task 3

Check if all number are unique:



Task 3

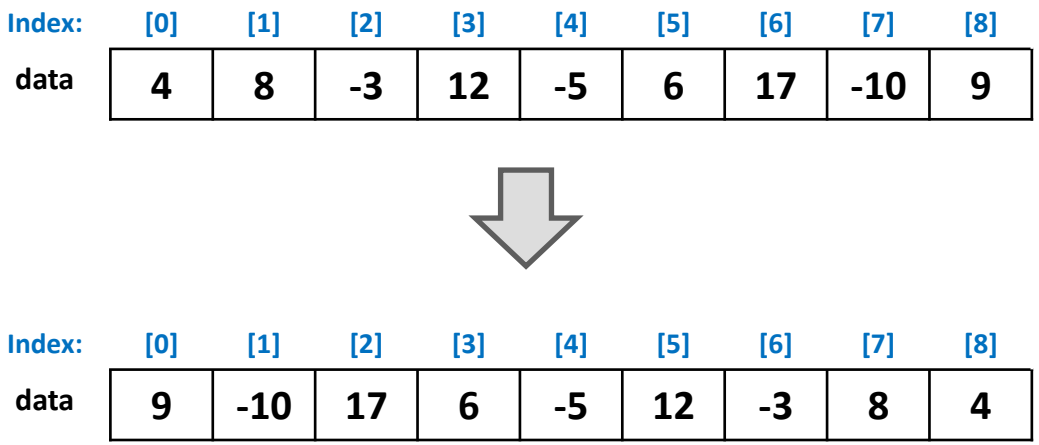
- What programming issues (mechanics) should you think about?
  - Are the indexes independent or is one dependent on another?
  - How and when are we ready to print our answer?
  - How do we stop (one or both loops)?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 4

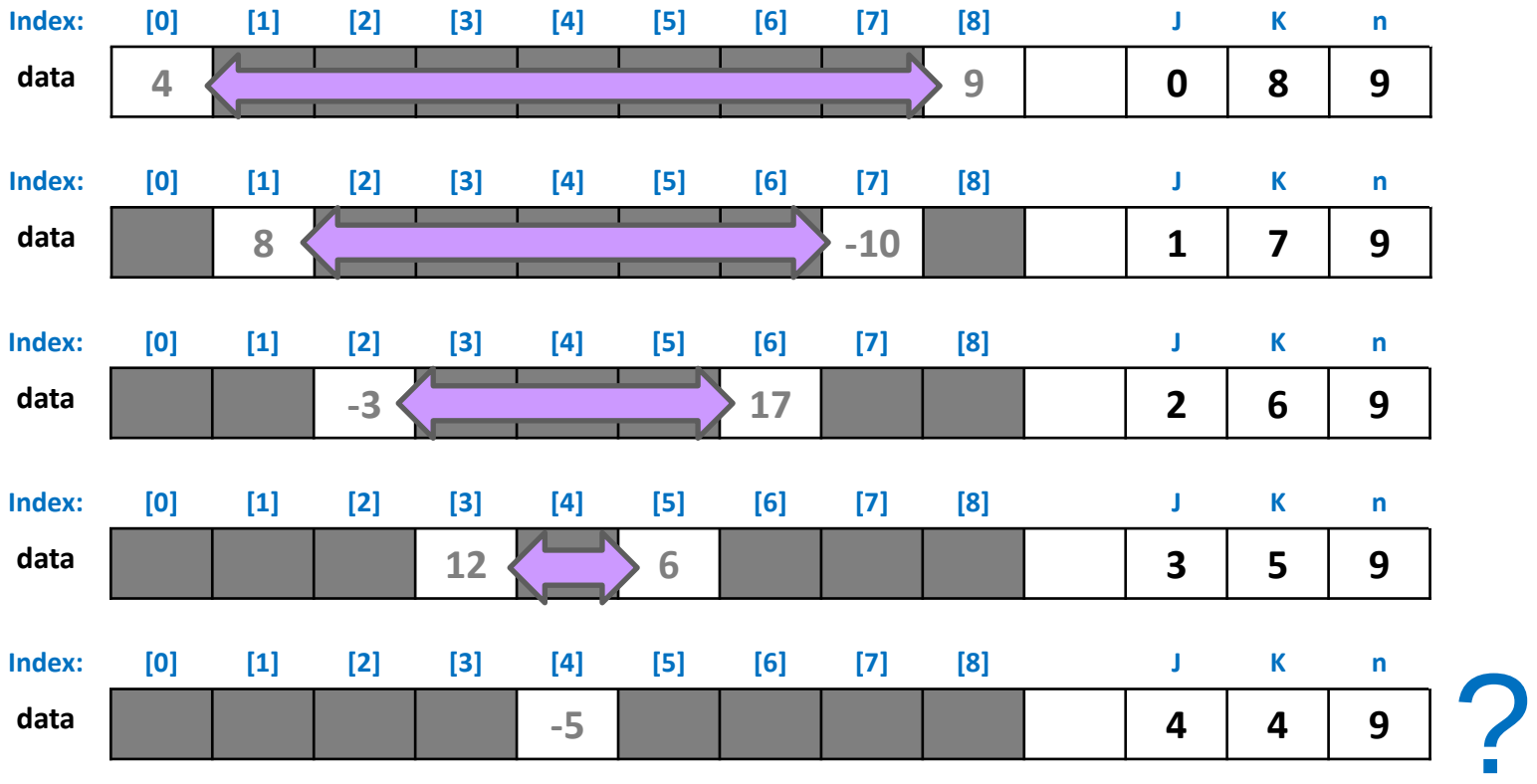
- Reverse an array of integers
  - Question: Can we do this in 1 pass or do we need a nested loop where we examine "pairs" ?

Reverse an array's contents:



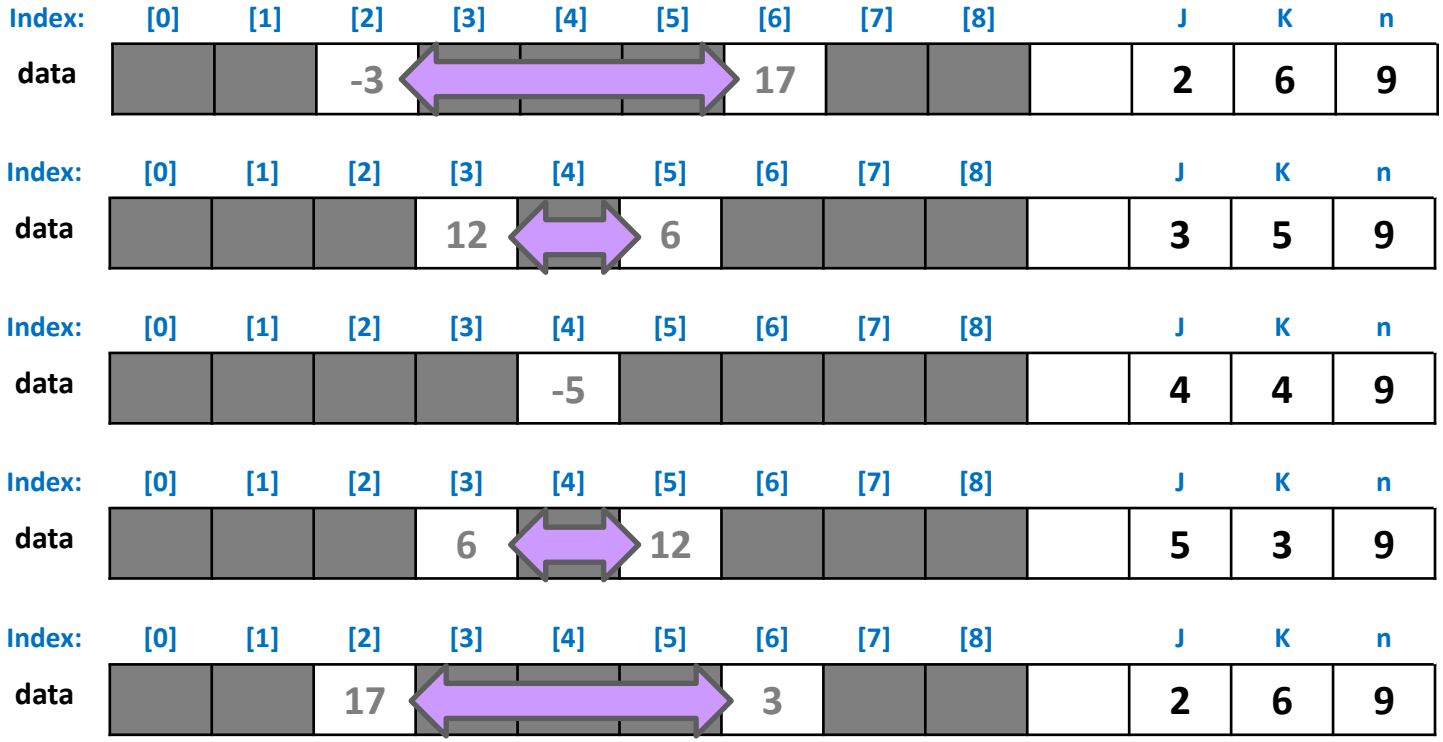
Task 4

Reverse an array's contents:



Task 4

Should we keep going?



No! We must stop at the halfway point to avoid "undoing" the swaps we've just done.

Task 4

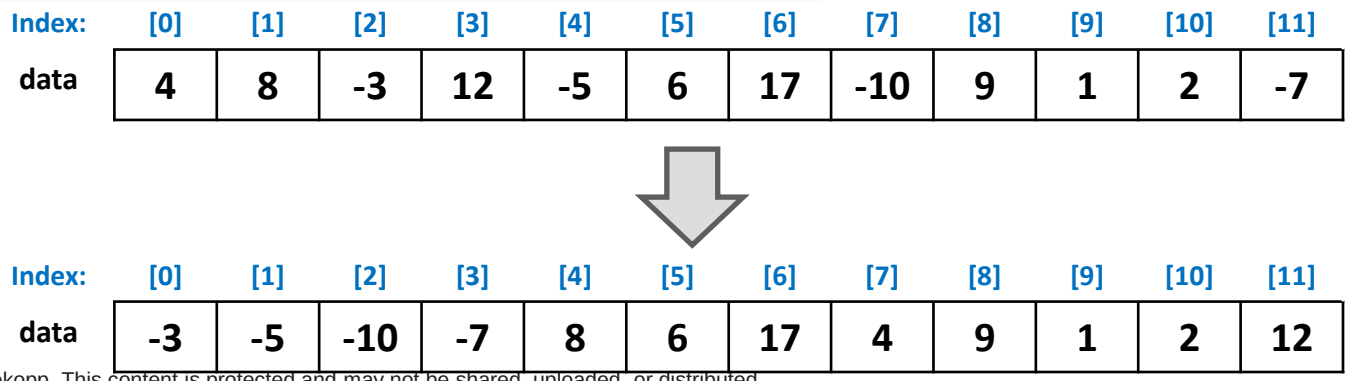
- What programming issues (mechanics) should you think about?
  - Does an even or odd length array need to be handled separately?
  - Can we do this in 1 pass or do we need a nested loop where we examine "pairs" ?
  - Are the indexes we need to generate independent or is one dependent on another?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 5

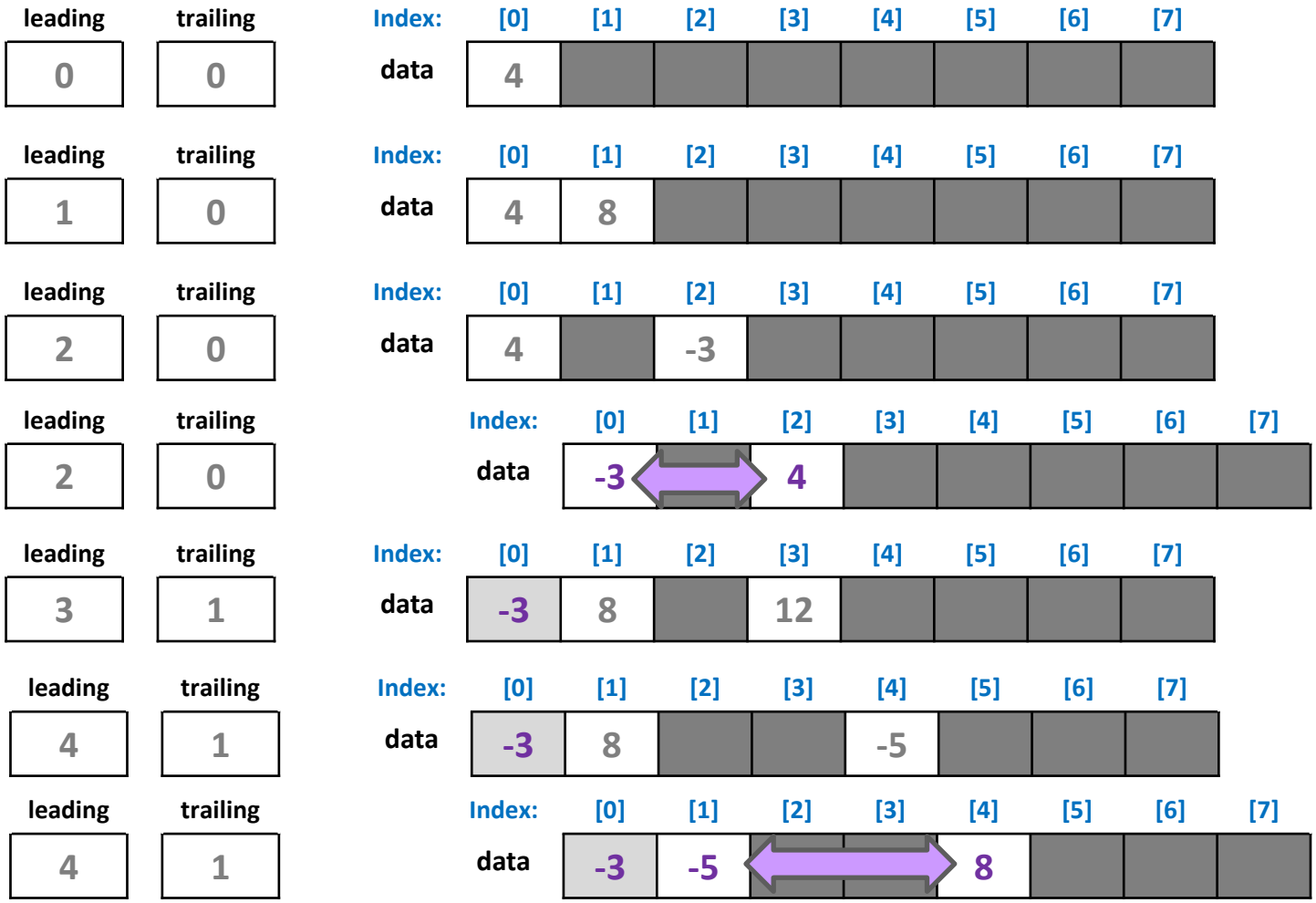
- Move all negative numbers to the front of the array, preserving order of negative numbers (but not necessarily positive numbers)
  - Question: Can we do this in 1 pass or do we need a nested loop where we examine "pairs"
  - Can we identify the items to move as we perform 1 pass?
  - If we need to move it, would we know where to place it?
  - When we move it, do we risk overwriting something we should not?

Move all negative numbers to the front of the array:



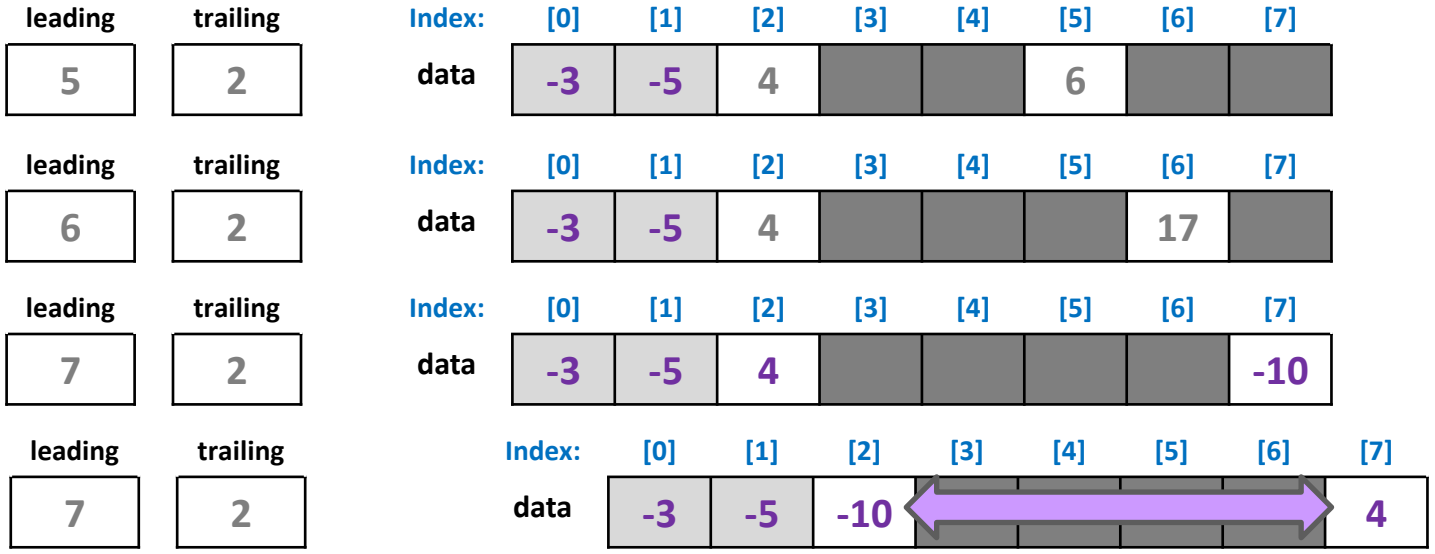
Task 5

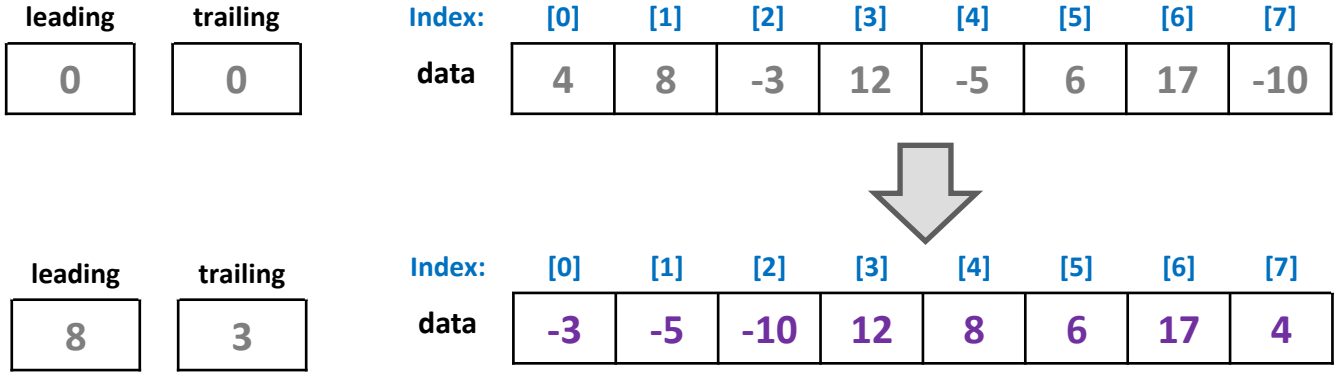
Move all negative numbers to the front of the array:



Task 5

Move all negative numbers to the front of the array:





Task 5

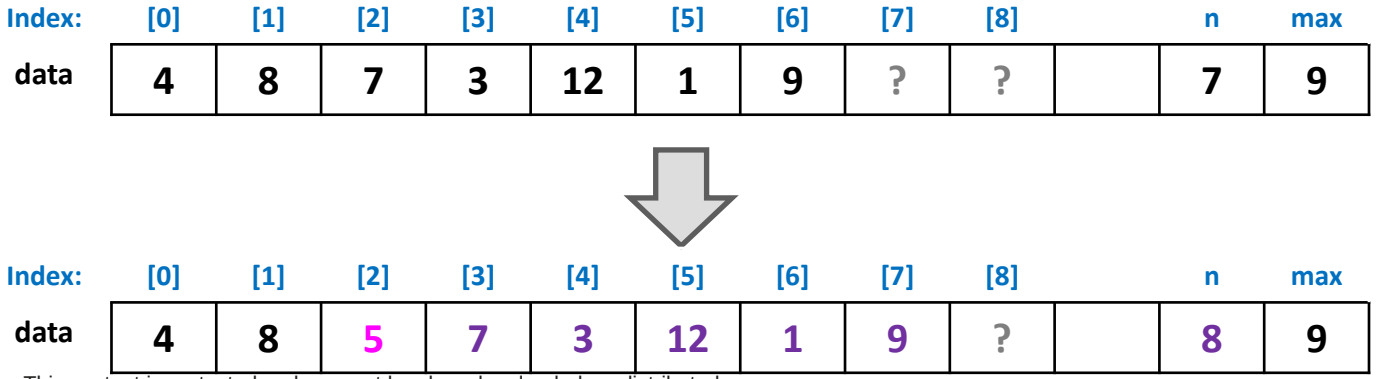
- What programming issues (mechanics) should you think about?
  - When do we increment leading?
  - When do we increment trailing?
- Invariants:
  - All values behind trailing are negative
  - All values between leading and trailing are positive

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 6

- Assuming an array of size, max, but only n occupied elements, insert a new value, v, at location, loc, shifting others back
  - Question: Can we do this in 1 pass or do we need a nested loop where we examine "pairs" ?
  - Do we insert then shift? Or shift then insert?
  - In what order should we shift the needed values?

Insert 5 at location 2 into the array of 7 elements and 9 locations.



Task 6

Insert 5 at location 2 into the array of 7 elements and 9 locations.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data	4	8	7	3	12	1	9	?	?		7	9	2	2

`data[k+1] = data[k]`

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data	4	8	7	7	12	1	9	?	?		7	9	2	2

`data[k+1] = data[k]`

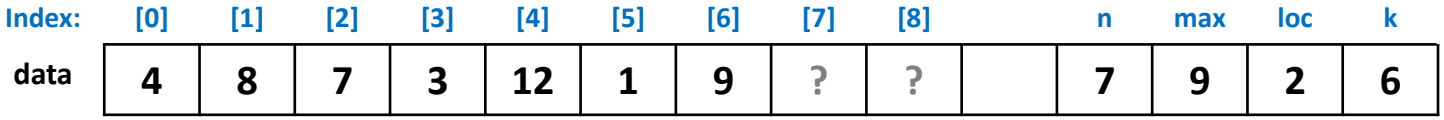
Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data	4	8	7	7	7	1	9	?	?		7	9	2	3

`data[k+1] = data[k]`

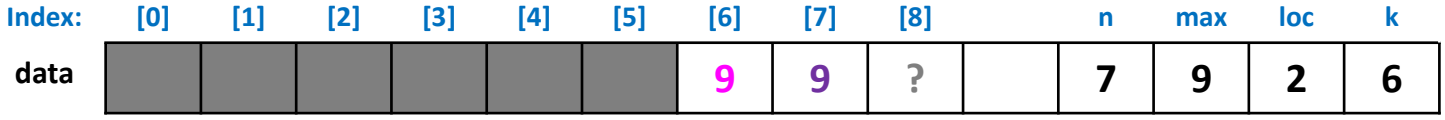
Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data	4	8	7	7	7	7	9	?	?		7	9	2	4

Task 6

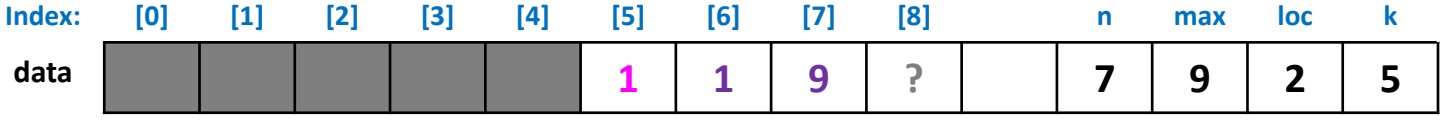
Insert 5 at location 2 into the array of 7 elements and 9 locations.



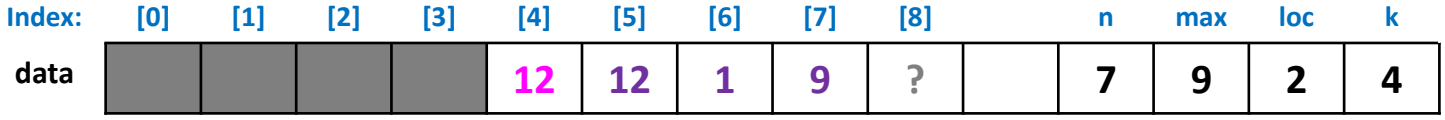
data[k+1] = data[k]



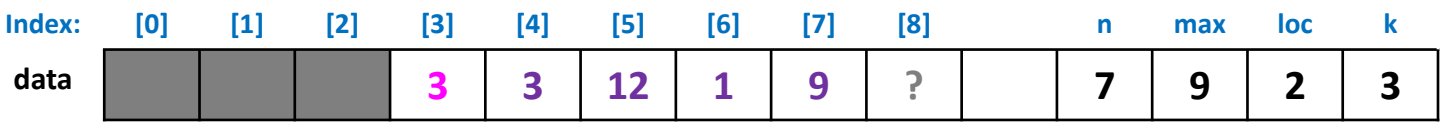
data[k+1] = data[k]



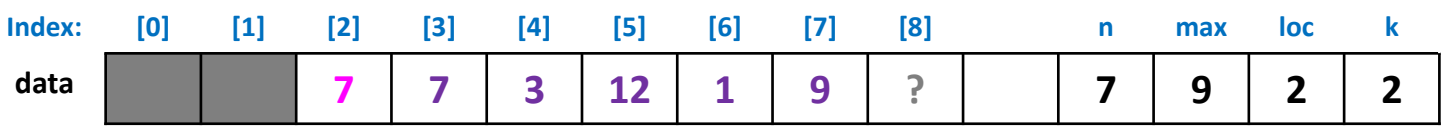
data[k+1] = data[k]



data[k+1] = data[k]



data[k+1] = data[k]



Task 6

Insert 5 at location 2 into the array of 7 elements and 9 locations.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data			7	7	3	12	1	9	?		7	9	2	2

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data			5	7	3	12	1	9	?		8	9	2	2

```
data[loc] = val;
n++;
```

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]		n	max	loc	k
data	4	8	5	7	3	12	1	9	?		8	9	2	2

Task 6

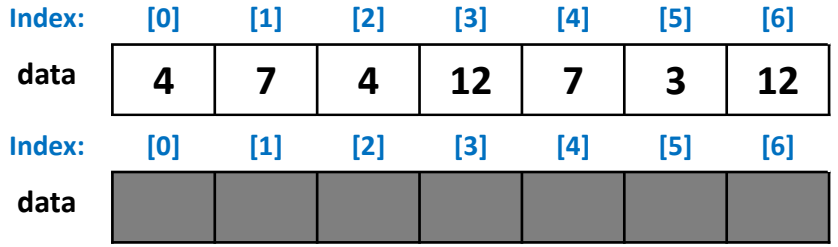
- What programming issues (mechanics) should you think about?
  - Do we want to move from k to k+1 OR k-1 to k?
  - Based on the above where should we start and stop our loop?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

Task 7

- Given there is 1 number that does NOT have a duplicate (pair), find and output that unique number.
  - Questions: Is this generating all pairs again?
  - Might we be able to answer "early"?

Find the unique number:

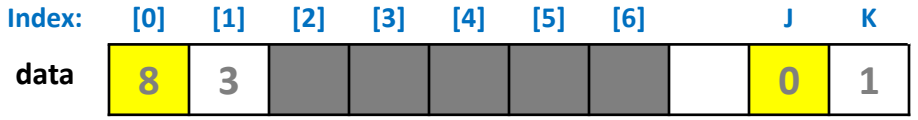


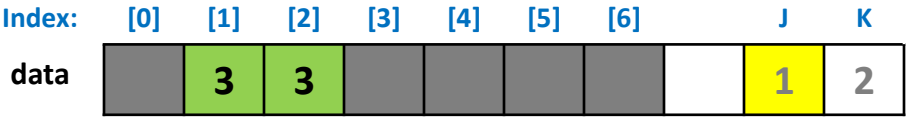
Sample Data:

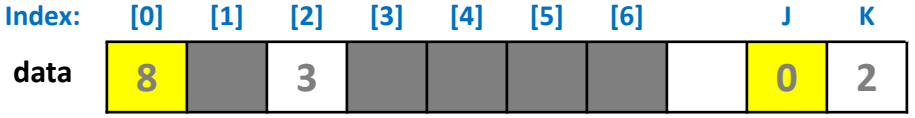
```
33 13 23 72 20 6 61 66 46 54 33 1 17 47 29 73 18 1 50
73 89 46 5 98 13 32 70 32 10 10 87 53 99 12 5 61 12 18
76 96 87 60 96 82 47 52 29 76 93 70 71 6 7 39 48 48 17
99 36 82 72 60 71 89 36 98 54 93 7 66 39 52 53 50 20
```

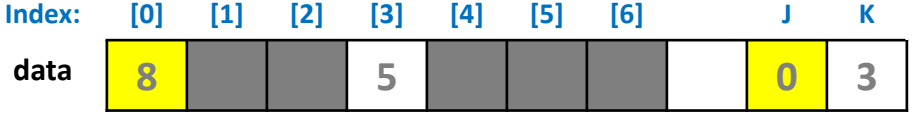
Task 7

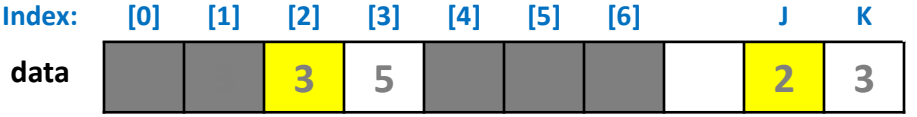
Find the unique number finding all pairs (with only one ordering rather than both for each pair):

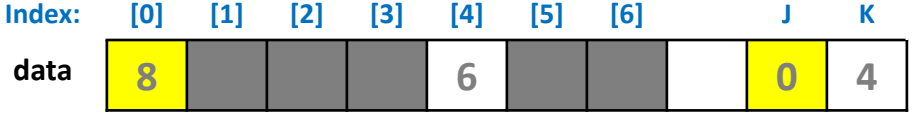


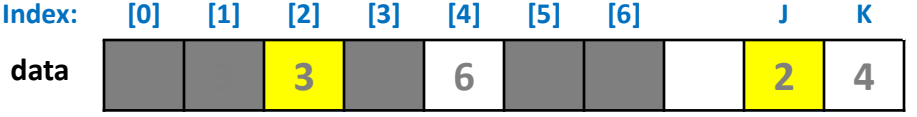


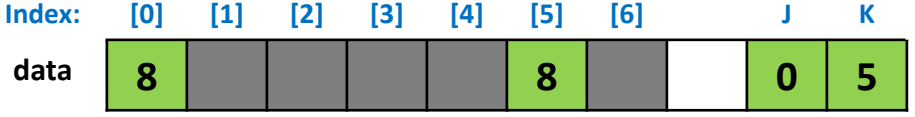


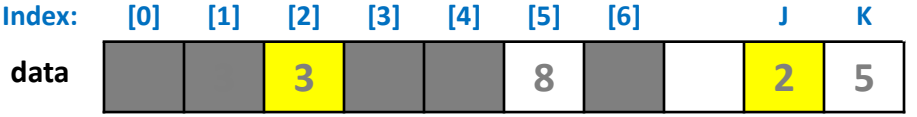


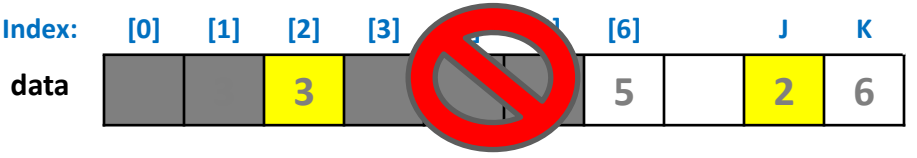








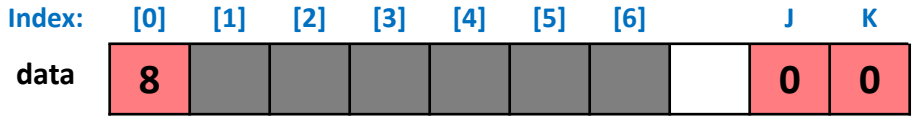


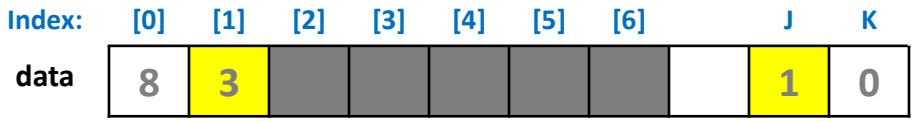


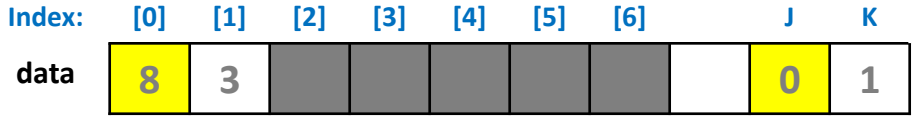
We didn't find a match for the 2nd item in a pair that DID exist!

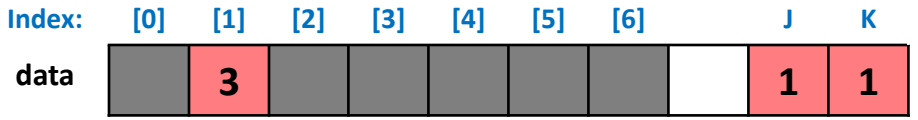
Task 7

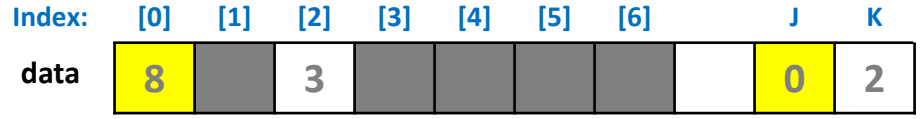
Find the unique number with all pairs (in both orders):

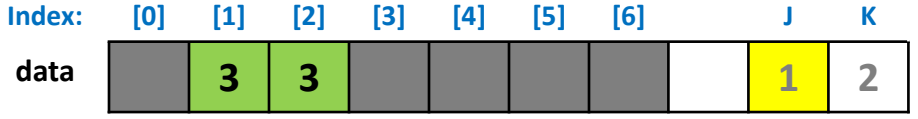


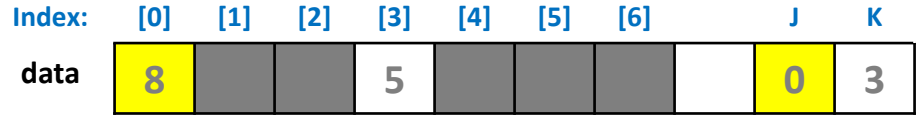


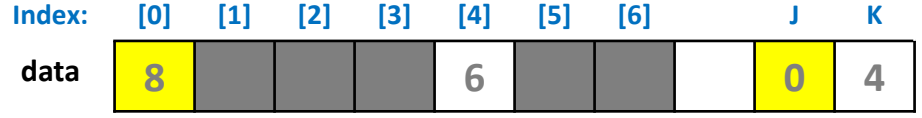


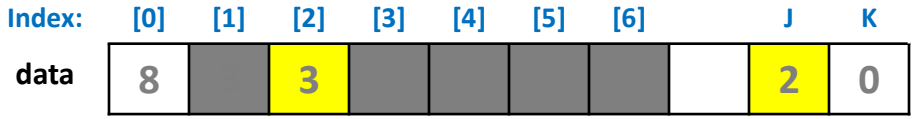


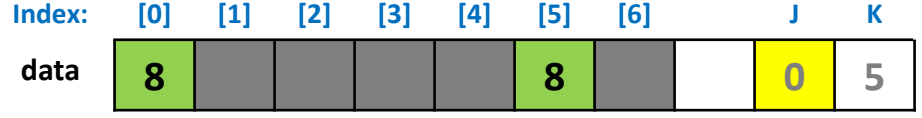


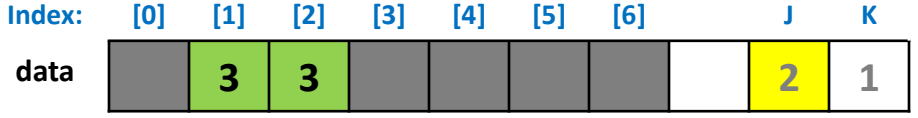






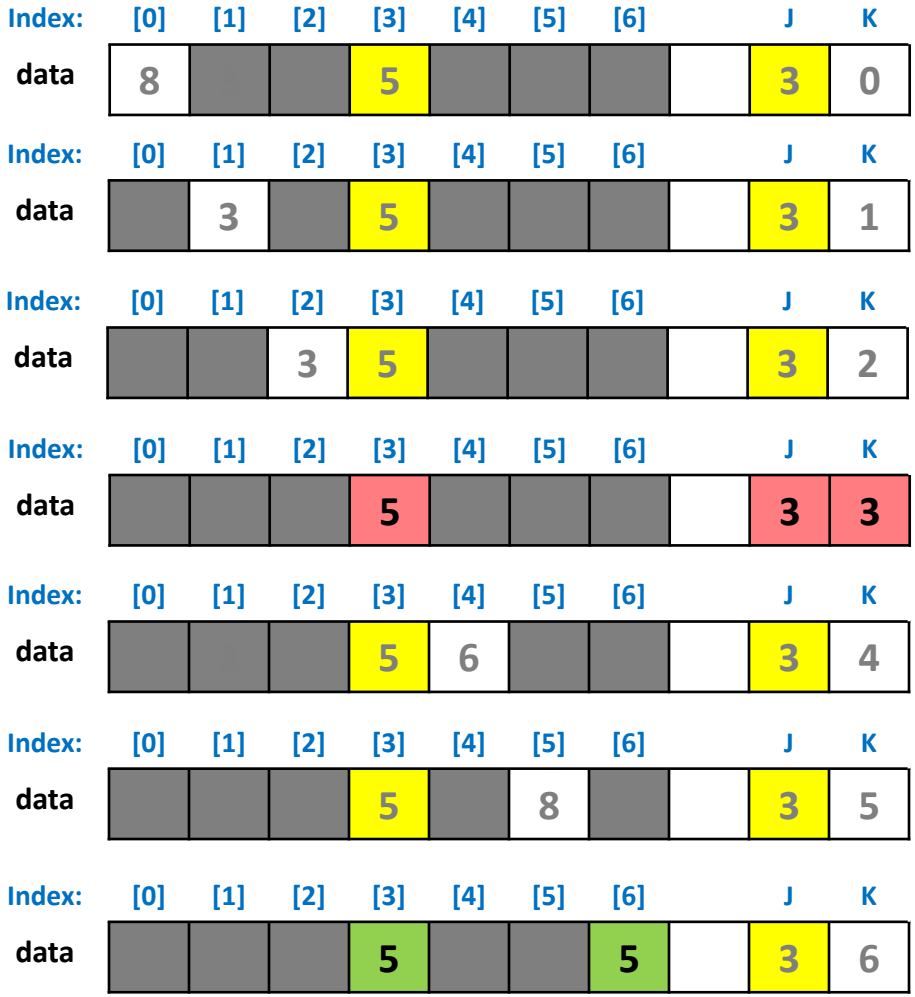


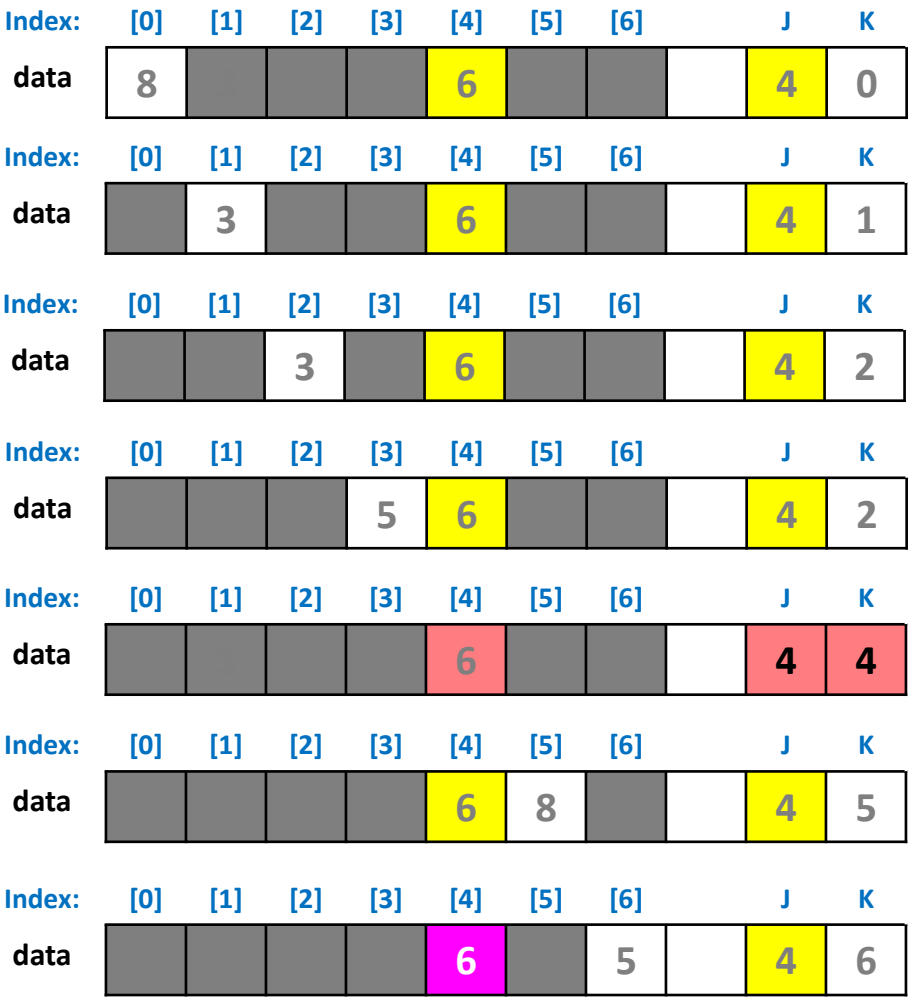




Task 7

Find the unique number:





6 is the unique number!

Task 7

- What programming issues (mechanics) should you think about?
  - How do we avoid matching ourself?
  - How do we know we've found a unique item?
  - When and how can we stop early?
- Variations
  - Allow 0 or more unique values and output the unique values OR output "All have a pair" if each number as a pair

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  return 0;
}
```

SOLUTIONS

Task 1 - Solution

- What programming issues (mechanics) should you think about?
  - How would you generate the appropriate indexes?
  - When can you stop?
  - When would you be ready to return -1?

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  cin >> val;
  bool found = false;
  for(int i=0; i < n; i++) {
    if(val == data[i]){
       cout << i << endl;
       found = true;
       break;
    }
  }
  if(!found) { cout << -1 << endl; }
  return 0;
}
```

Task 2 - Solution

- What programming issues (mechanics) should you think about?
  - How would you generate the appropriate indexes?
  - Are the indexes independent or is one dependent on another?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  for(int j=0; j < n; j++) {
    for(int k=j+1; k < n; k++) {
      cout << data[j] << "," << data[k] << " ";
    }
  }
  cout << endl;
  return 0;
}
```

Task 3 - Solution

- What programming issues (mechanics) should you think about?
  - Are the indexes independent or is one dependent on another?
  - How and when are we ready to print our answer?
  - How do we stop (one or both loops)?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  bool allUnique = true;
  for(int j=0; j < n; j++) {
    for(int k=j+1; k < n; k++) {
      if(data[j] == data[k]){
        allUnique = false;
        break;
      }
    }
    if(!allUnique) break;
  }
  if(allUnique)
   { cout << "All unique" << endl; }
  else
   { cout << "Not all unique" << endl; }
  return 0;
}
```

Task 4 - Solution

- What programming issues (mechanics) should you think about?
  - Does an even or odd length array need to be handled separately?
  - Can we do this in 1 pass or do we need a nested loop where we examine "pairs" ?
  - Are the indexes we need to generate independent or is one dependent on another?

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  for(int j=0; j < n/2; j++) {
    // swap data[j] and data[k] (k=n-j-1)
    int temp = data[j];
    data[j] = data[n-j-1];
    data[n-j-1] = temp;
  }
  return 0;
}
```

Task 5 - Solution

- What programming issues (mechanics) should you think about?
  - When do we increment leading?
  - When do we increment trailing?
- Invariants:
  - All values behind trailing are negative
  - All values between leading and trailing are positive

```
int main() {
  // setup array with data
  int n, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  int lead, trail = 0;
  for(lead=0; lead < n; lead++) {
    if(data[lead] < 0) {
      // swap leading and trailing
      int temp = data[lead];
      data[lead] = data[trail];
      data[trail] = temp;
      // only increment if we move
      trail++;
    }
  }
  return 0;
}
```

Task 6 - Solution

- What programming issues (mechanics) should you think about?
  - Do we want to move from k to k+1 OR k-1 to k?
  - Based on the above where should we start and stop our loop?

```
int main() {
  // setup array with data
  int n, data[20];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  int loc, val;
  cin >> loc >> val;
  if(n < 20 && loc >= 0 && loc <= n){
    for(int k=n-1; k >= loc; k--) {
      data[k+1] = data[k];
    }
    data[loc] = val;
    n++;
  }
  else {
    cout << "Invalid" << endl;
  }

  cout << endl;
  return 0;
}
```

Task 7 - Solution

- What programming issues (mechanics) should you think about?
  - How do we avoid matching ourself?
  - How do we know we've found a unique item?
  - When and how can we stop early?
- Variations
  - Allow 0 or more unique values and output the unique values OR output "All have a pair" if each number as a pair

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  for(int j=0; j < n; j++) {
    bool unique = true;
    for(int k=0; k < n; k++) {
      if(j != k){
        if(data[j] == data[k]){
          unique = false;
          break;
        }
      }
    }
    if(unique){
      cout << data[j] << endl;
      break;
    }
  }
  return 0;
}
```