Data Structures

CSCI 2270-202: REC 03

Sanskar Katiyar

Logistics

Office Hours at ECAE 128 (Aerospace Lobby)

Tuesday: 12:15 pm – 2:15 pm

Friday: 1:30 pm – 3:30 pm

Today at ECAE 133
12pm - 2pm

Recitation Materials *(Notes, Slides, Code, etc.)*

sanskarkatiyar.github.io/CSCI2270

Recitation Outline

1. Dynamic Memory
2. Sorting: Bubble Sort
3. Linked Lists: Introduction
4. Exercise

Dynamic Memory

Static Memory Allocation

Memory for named variables is computed at compile time

Thus, exact type, size required

Stored in a stack (also a data structure)

Managed by processor/compiler

e.g.: `string s = “CSCI2270”;`

Dynamic Memory: Scenario

You need an array of size **N**, input by user ***at runtime***.

```
cin >> N;
int A[N];     // C++ does not like this!
```

Since A is declared as a static array, it will be stored in stack.

The stack is managed by compiler/processor: the compiler needs to know this size at compile time.

Dynamic Memory

Stack	Heap
Local variables	Pool of free memory
Static declarations	Dynamic declarations
Limited space available	Comparatively larger space
Managed by processor	Managed by developer

```
int *p = new int;
```

Dynamic Memory: new

new: allocate space on heap

Single Variable

```
int *ptr1 = new int;
```

Step 2
Store memory address

Step 1
Allocate memory on heap and return address

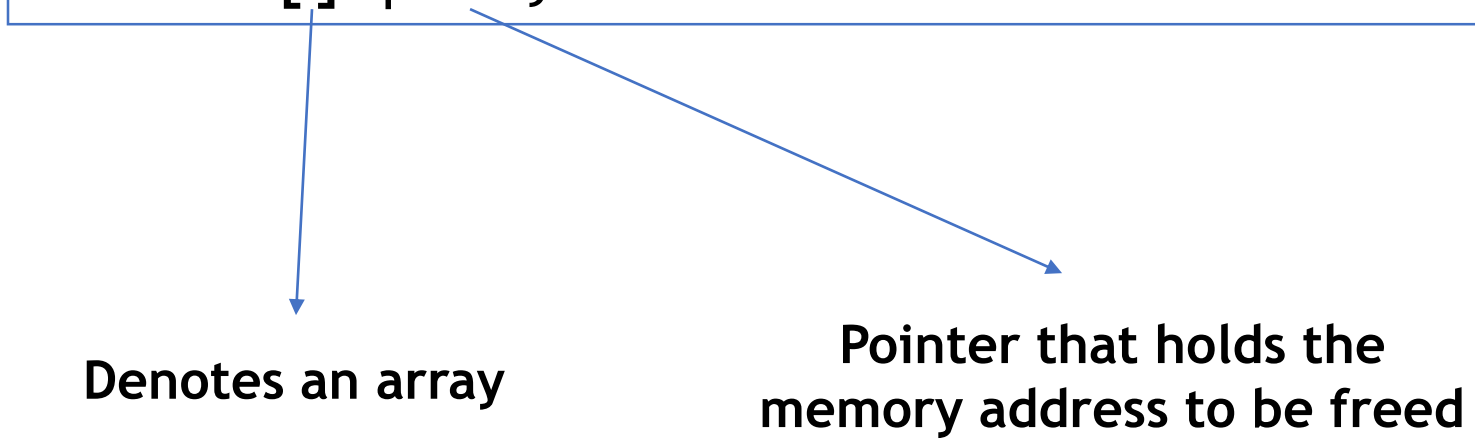
Arrays

```
int *ptr2 = new int[10];
```

Dynamic Memory: delete

`delete`: free space from heap

```
delete ptr1;     // does not delete variable ptr1

delete [] ptr2;
```



Dynamic Memory: new, delete

new: allocate space on heap

```
int *ptr1 = new int;
int *ptr2 = new int[10];
```

delete: free space from heap

```
delete ptr1;    // does not delete variable ptr1
delete [] ptr2;
```

Dynamic Memory: Careful

Memory Leak

Lost track of memory location without freeing it

Dangling Pointer

Set pointer to NULL or nullptr (when not in use)

Segmentation Faults!

Accessing a bad index in an array

Allocation, de-allocation mismatch

Accessing memory before initialization

Dynamic Memory: Array Doubling

Idea: When array is full, double the size

Applicable when input size is unknown

Algorithm

Step 1: Allocate heap memory for array (2 x N), store in pointer

Step 2: Copy contents of indices `A[0,N]` to `new_array[0,N]`

Step 3: Free the memory allocated to `A`

Step 4: `A` points to `new_array`

Sorting: Bubble Sort

Pssst! Assignment 2!

Sorting: Bubble Sort

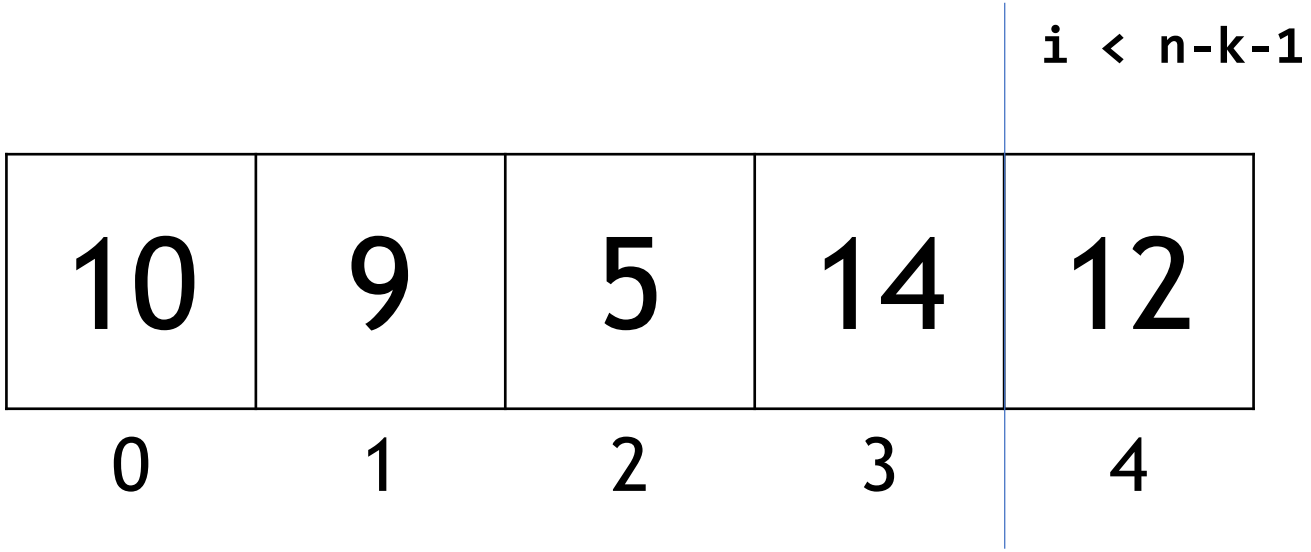
Idea: Repeatedly compare adjacent pairs of elements

Elements move up, in order, like bubbles

e.g: `int A[], length n`; sort A in ascending order

```
int temp, i, k;


for(k = 0; k < n-1; k++) {

    for(i = 0; i < n-k-1; i++) {

       if(A[i] > A[i+1])

           swap(A[i], A[i+1]);

    }

}
```

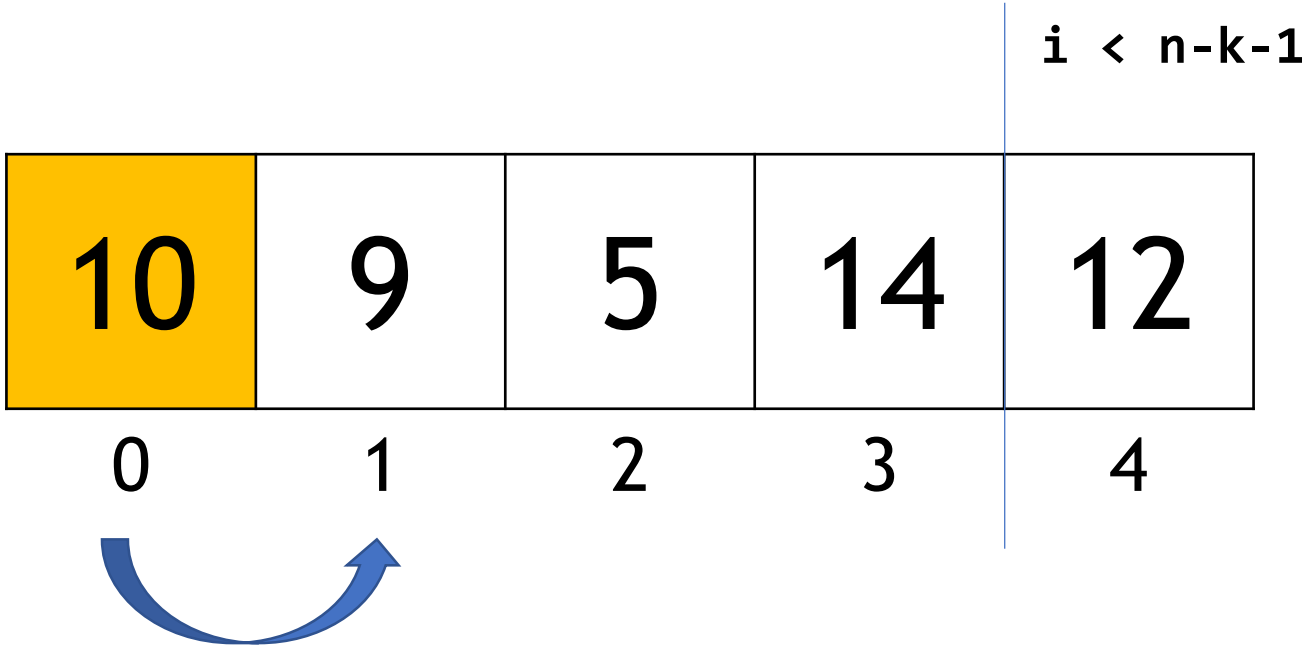
Sorting: Bubble Sort



i = 0

k = 0

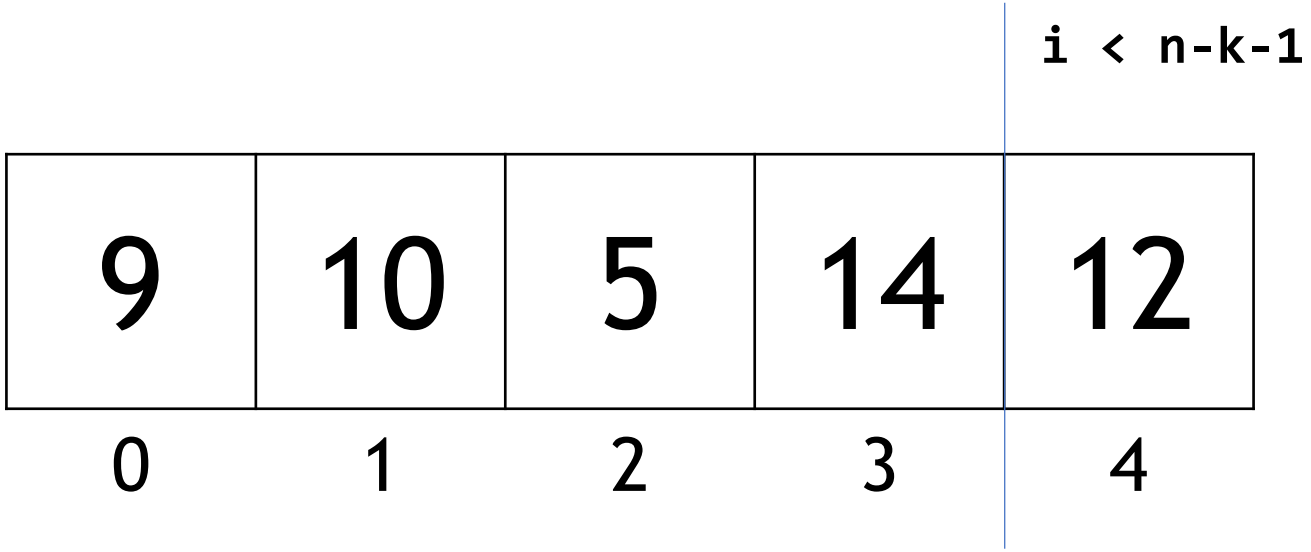
Sorting: Bubble Sort



i = 0

k = 0

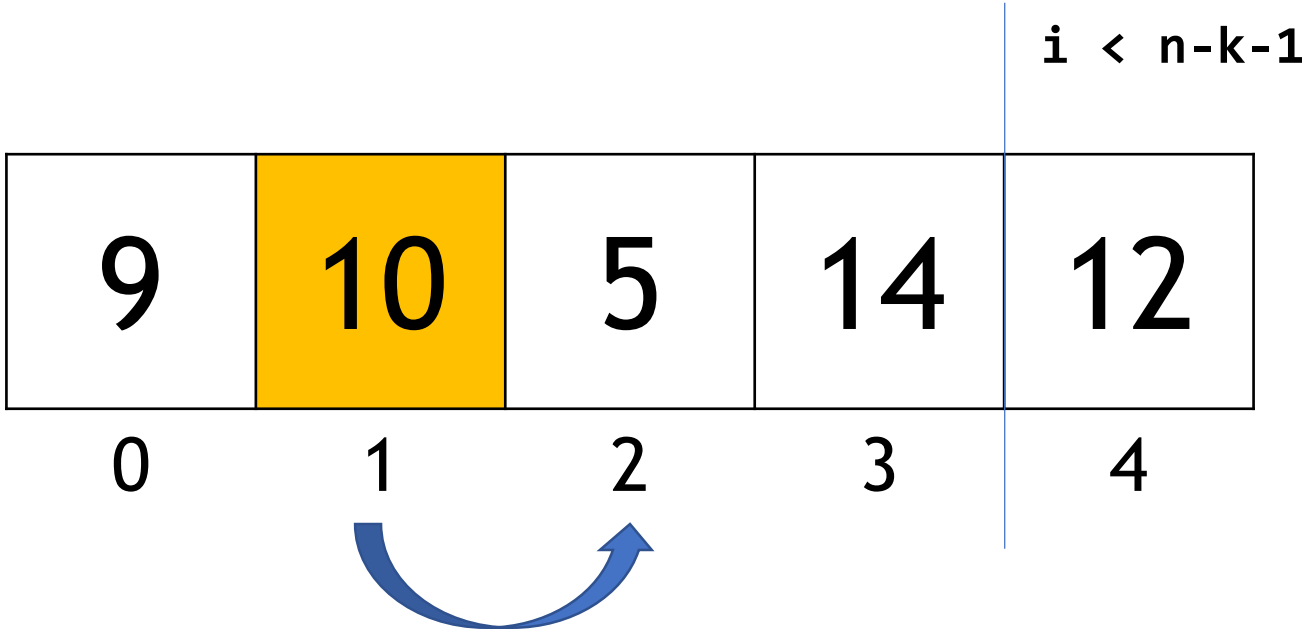
Sorting: Bubble Sort



i = 0

k = 0

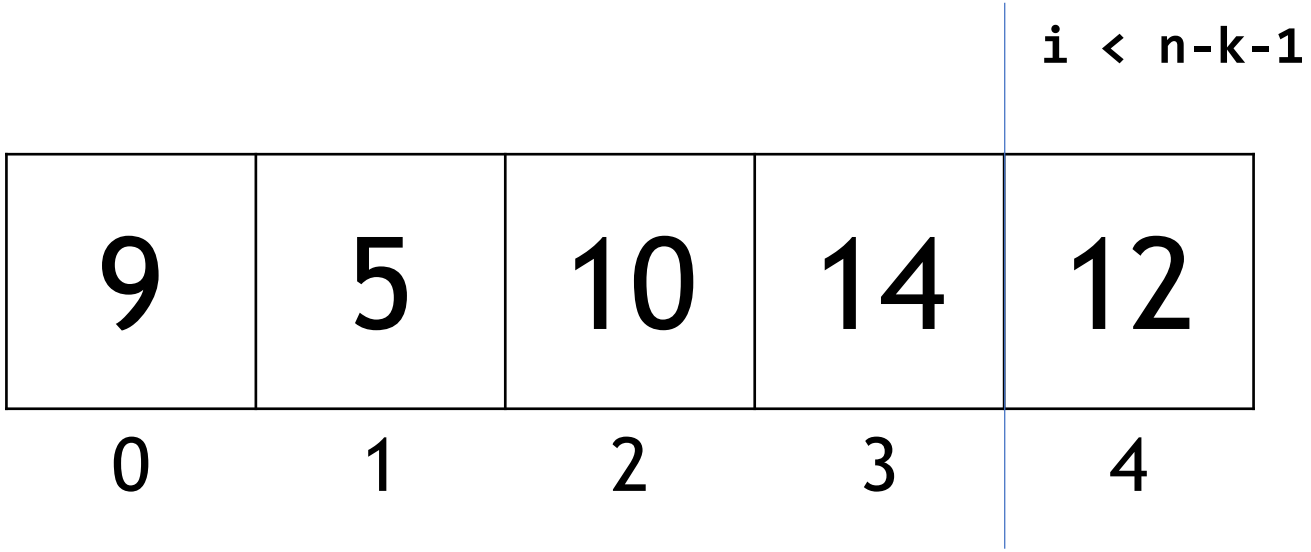
Sorting: Bubble Sort



i = 1

k = 0

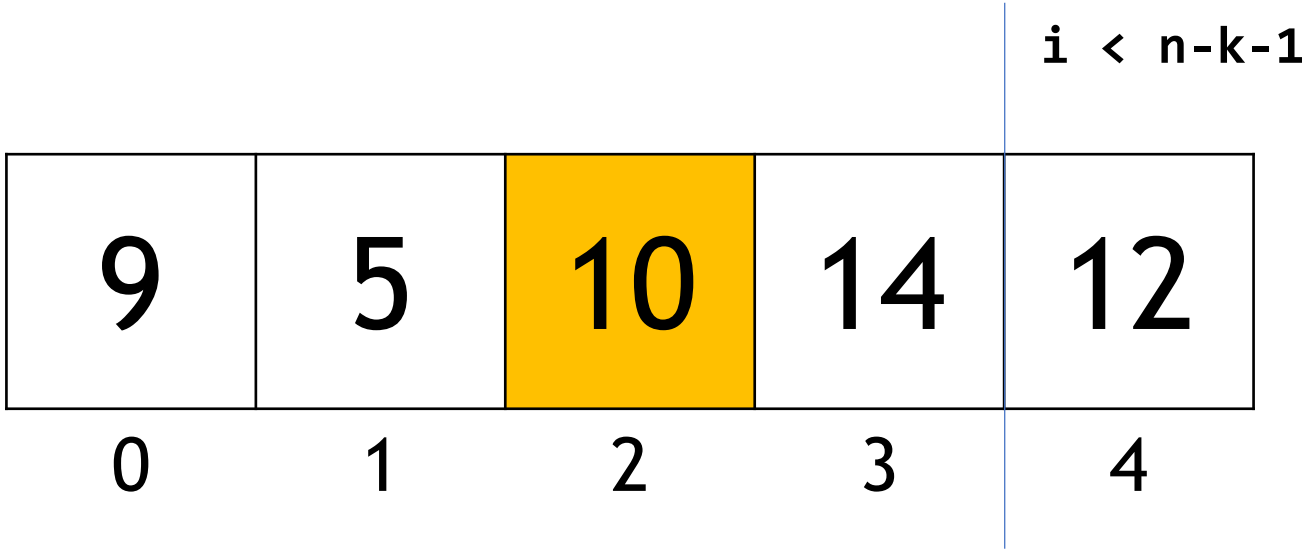
Sorting: Bubble Sort



i = 1

k = 0

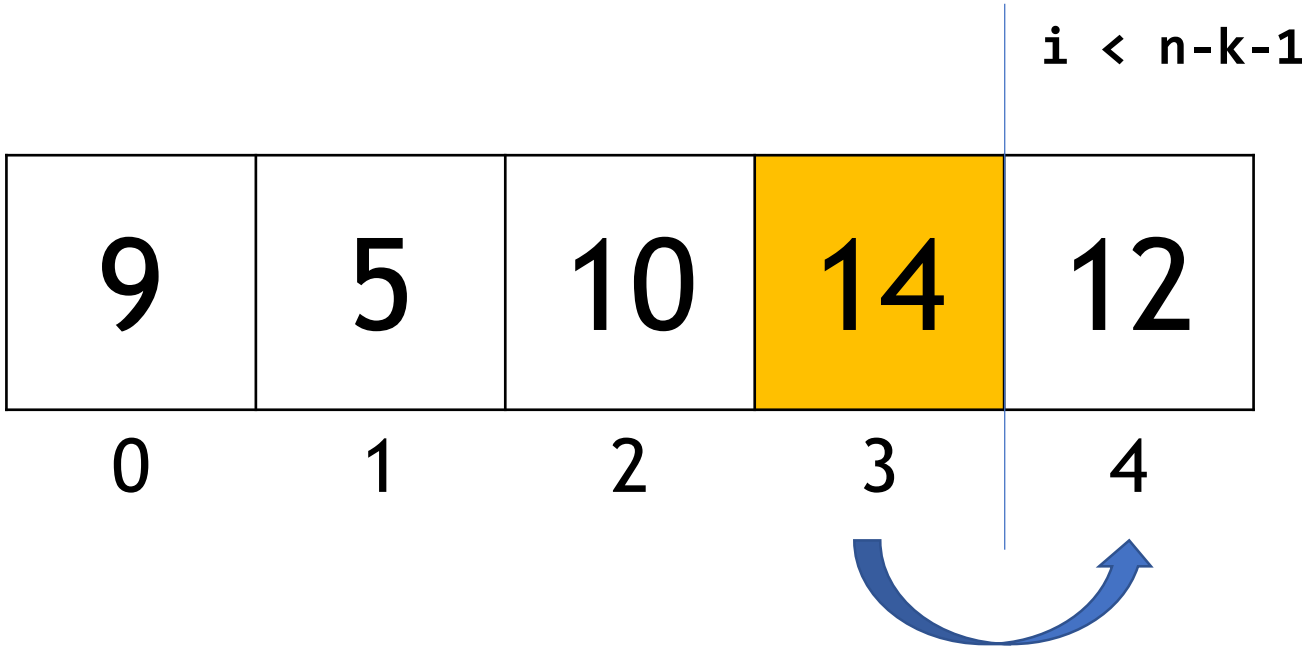
Sorting: Bubble Sort



i = 2

k = 0

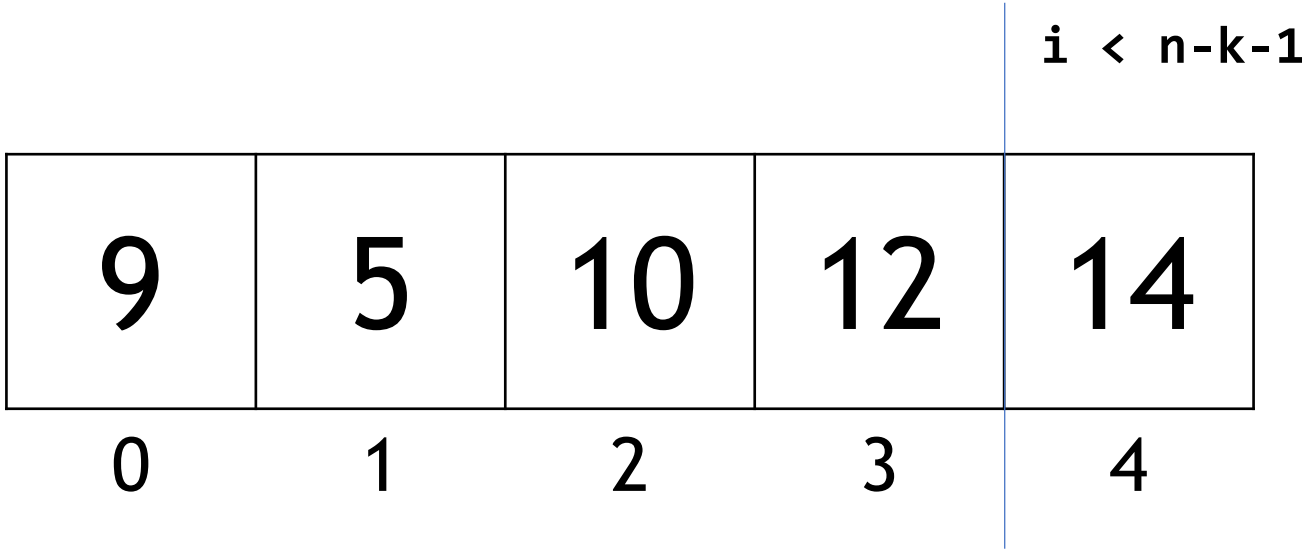
Sorting: Bubble Sort



i = 3

k = 0

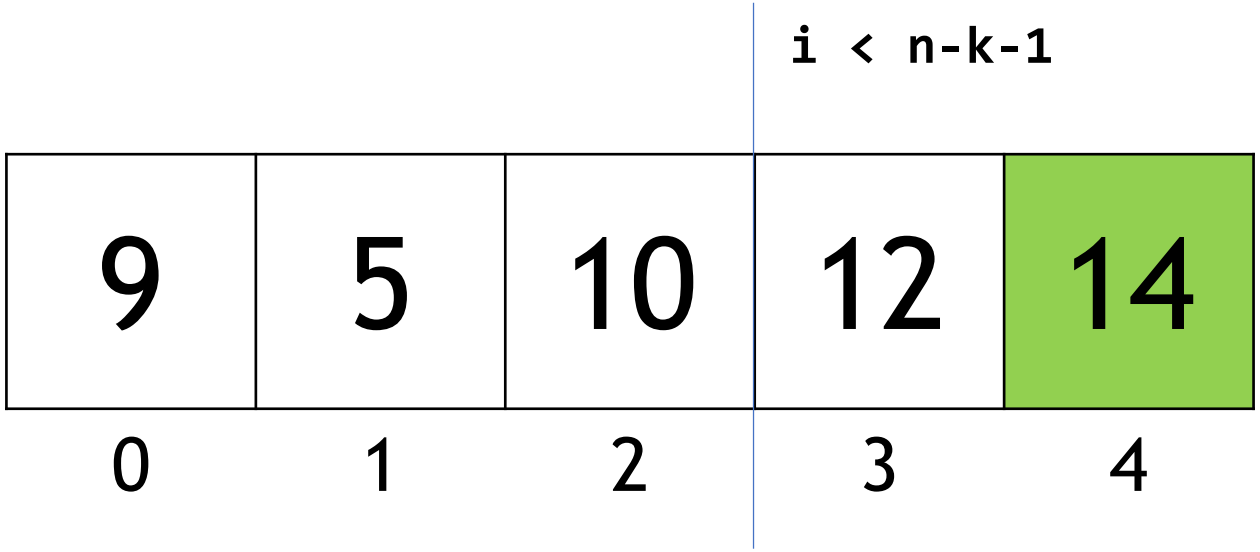
Sorting: Bubble Sort



i = 3

k = 0

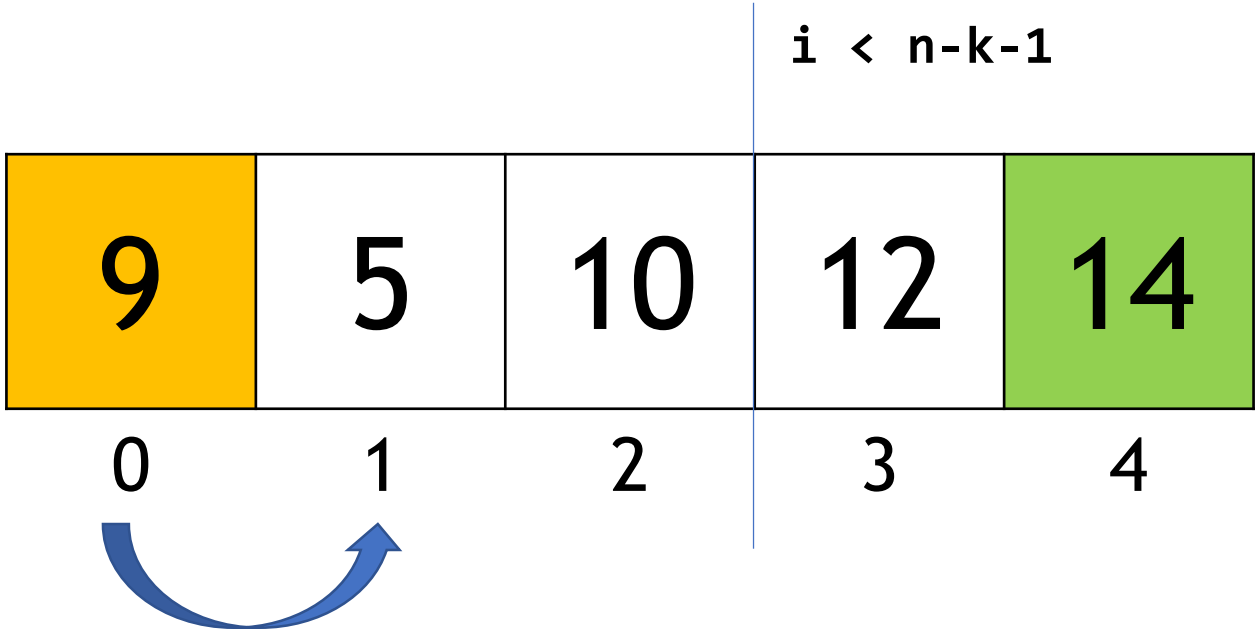
Sorting: Bubble Sort



i = N/A

k = 1

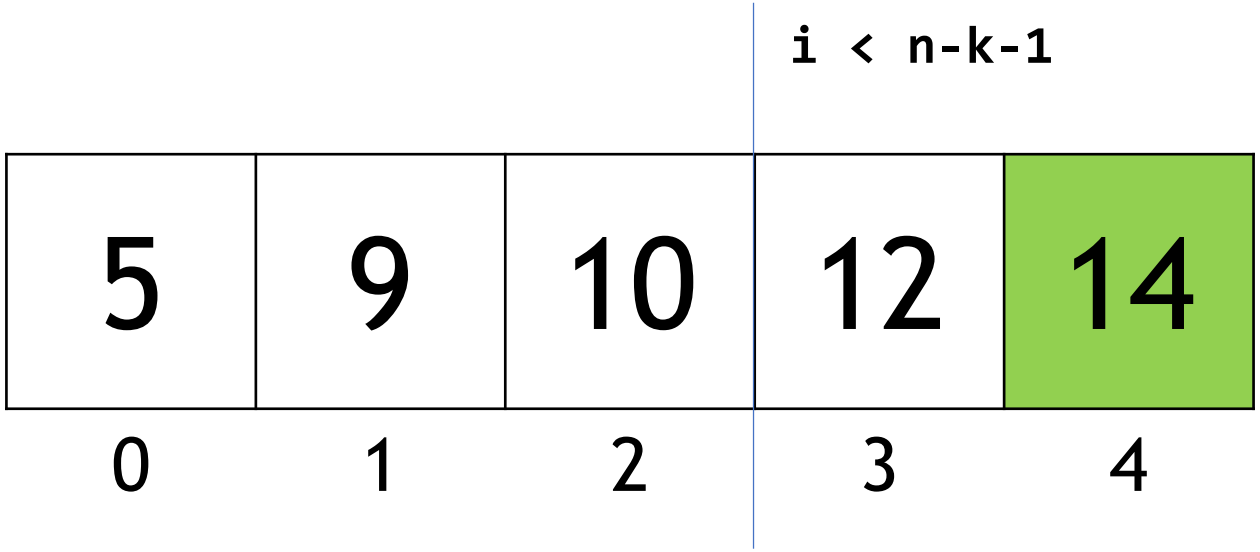
Sorting: Bubble Sort



i = 0

k = 1

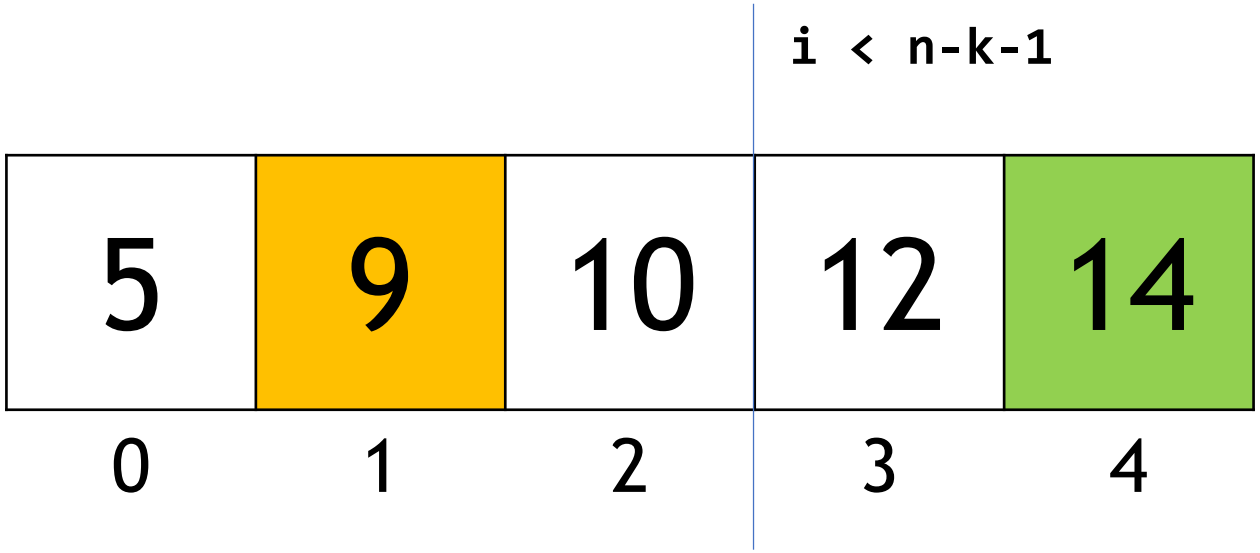
Sorting: Bubble Sort



i = 0

k = 1

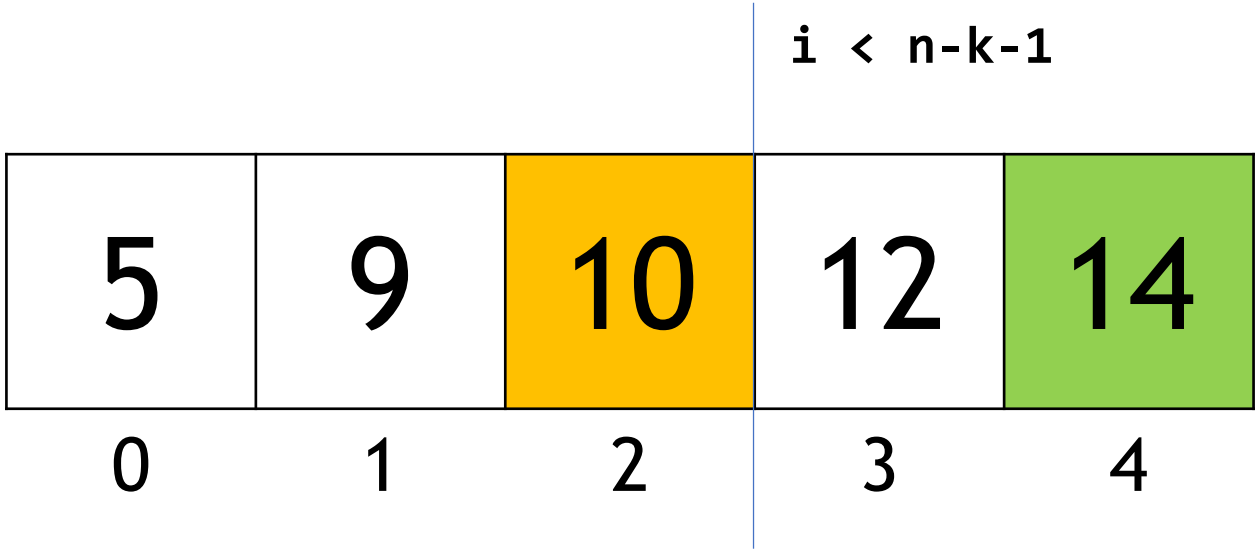
Sorting: Bubble Sort



i = 1

k = 1

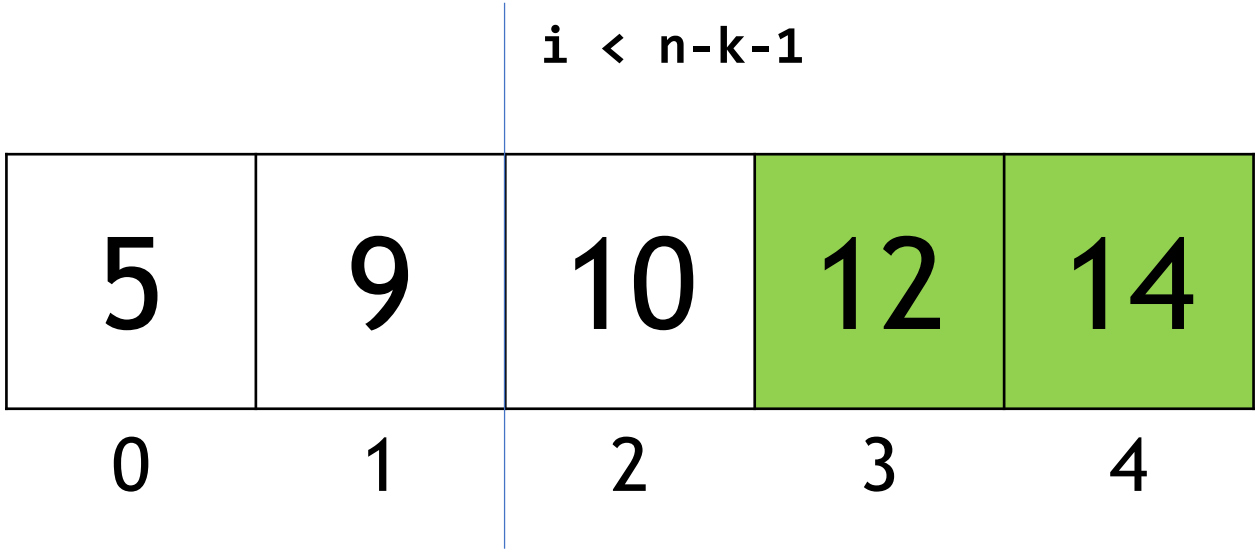
Sorting: Bubble Sort



i = 2

k = 1

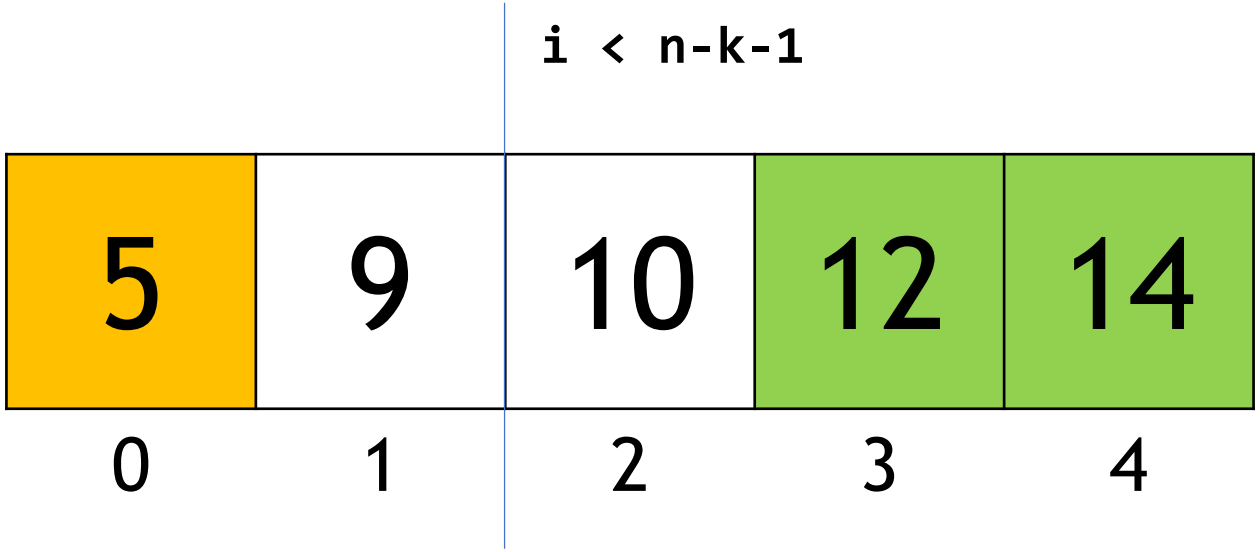
Sorting: Bubble Sort



i = N/A

k = 2

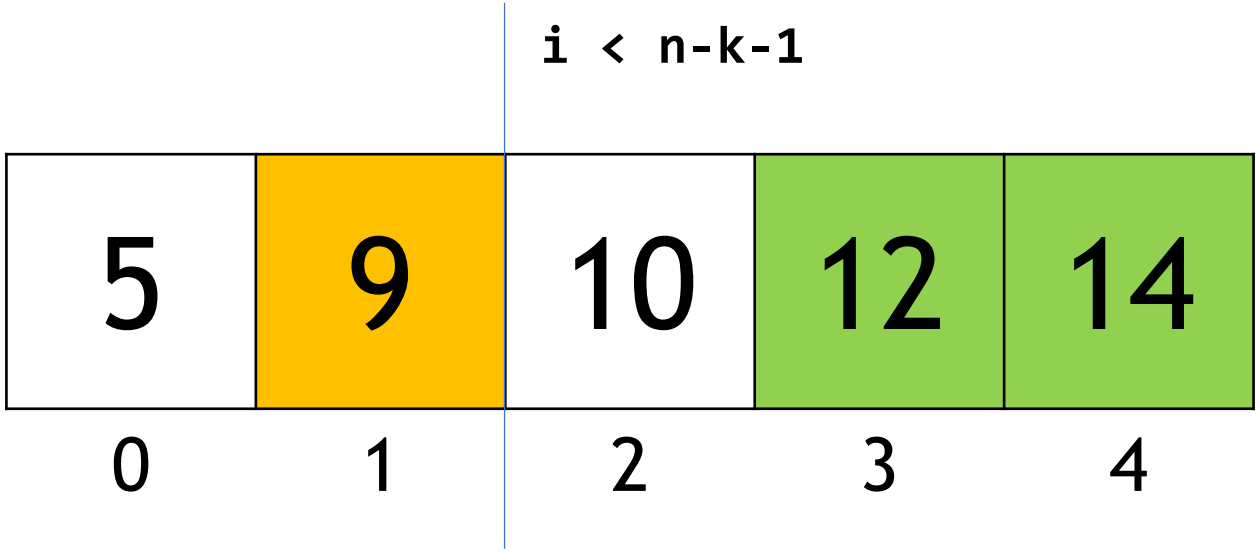
Sorting: Bubble Sort



i = 0

k = 2

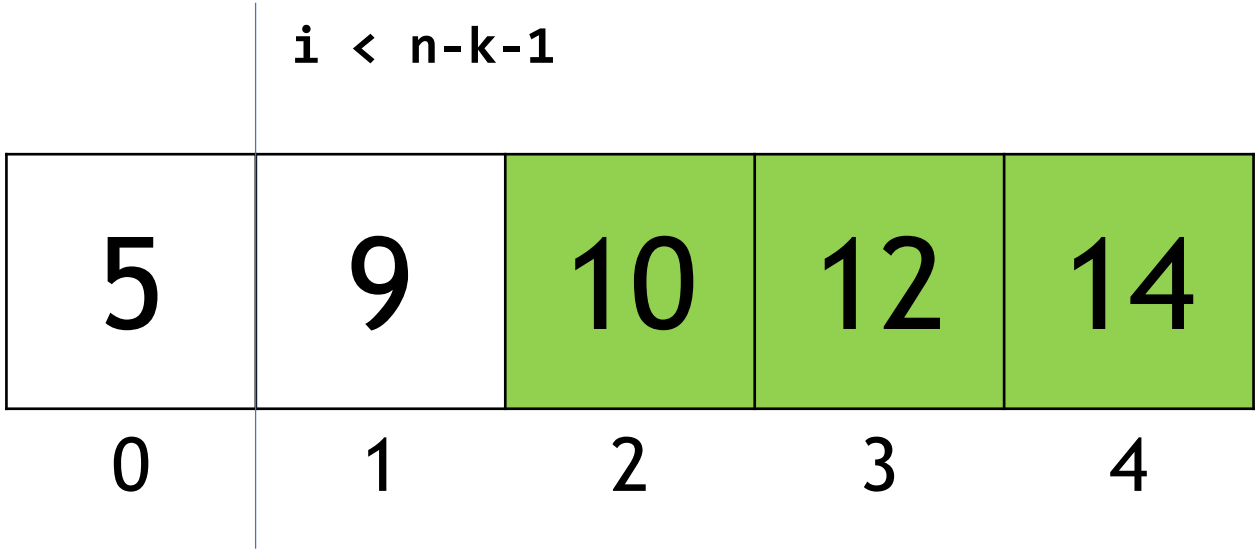
Sorting: Bubble Sort



i = 1

k = 2

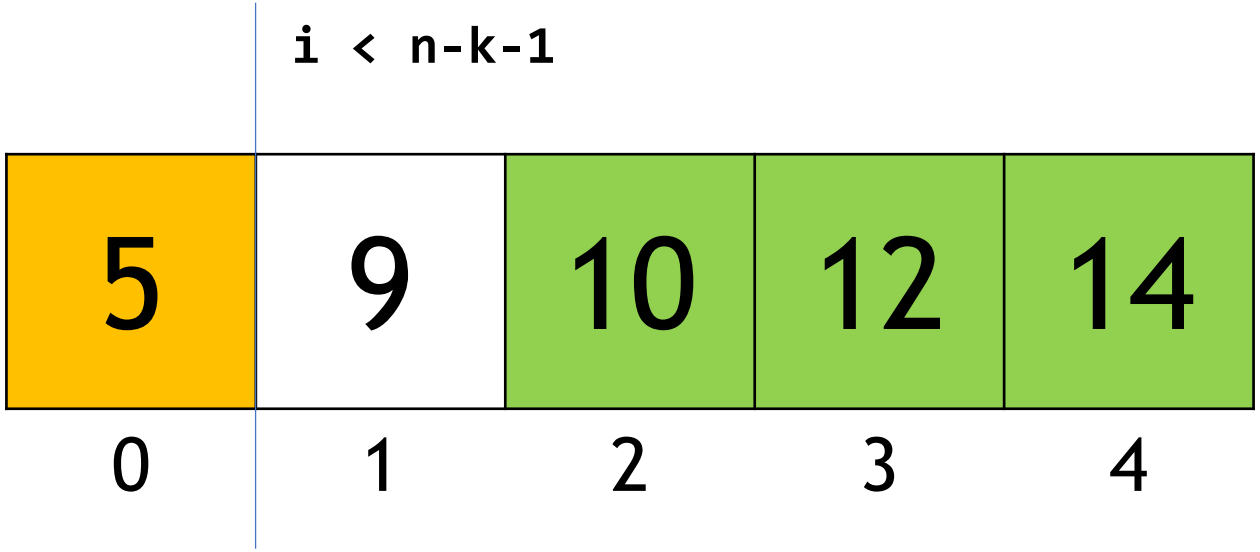
Sorting: Bubble Sort



i = N/A

k = 3

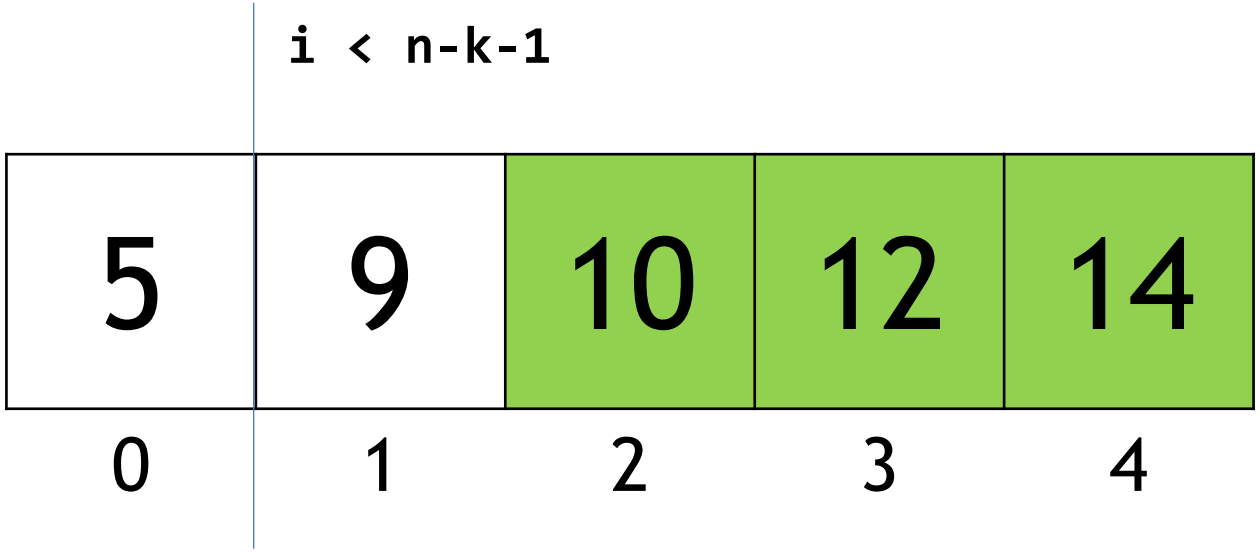
Sorting: Bubble Sort



i = 0

k = 3

Sorting: Bubble Sort

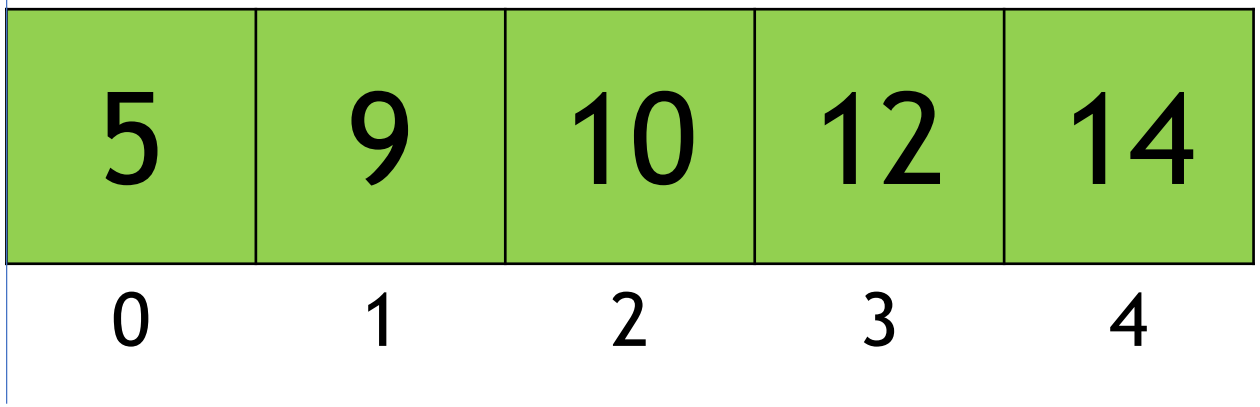


i = 0

k = 3

Sorting: Bubble Sort

```
i < n-k-1
```



i = N/A

k = N/A

Sorting: Bubble Sort

Idea: Repeatedly compare adjacent pairs of elements

Elements move up, in order, like bubbles

e.g: `int A[], length n`; sort A in ascending order

```
int temp, i, k;


for(k = 0; k < n-1; k++) {

    for(i = 0; i < n-k-1; i++) {

       if(A[i] > A[i+1])

           swap(A[i], A[i+1]);

    }

}
```

Linked List

Introduction

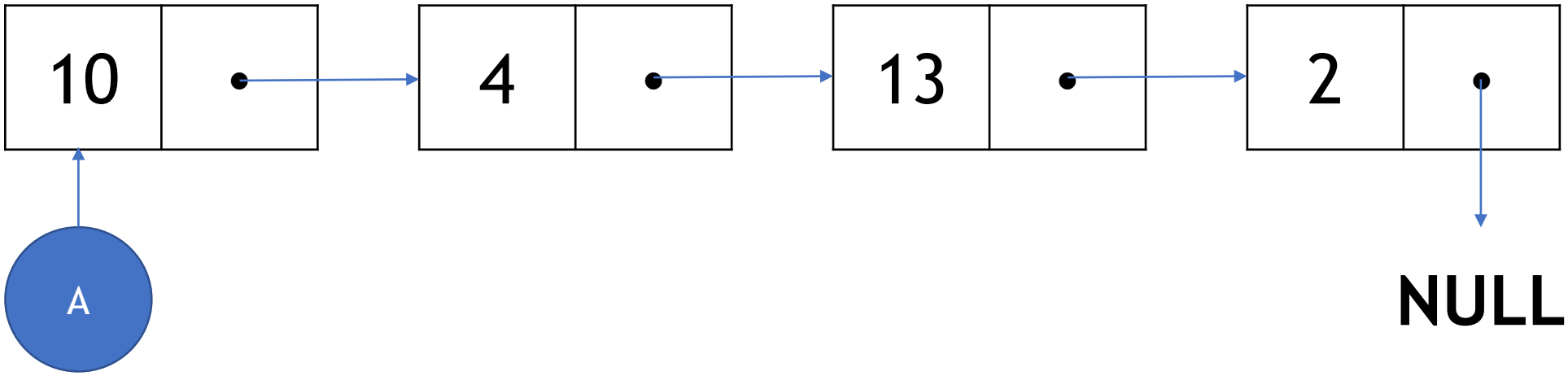
Linked List

Stores a list

List is composed of ***nodes*** (and ***links***)

Need to track only *head* of the list; last node's link is to NULL

Not contiguous in memory (unlike arrays)

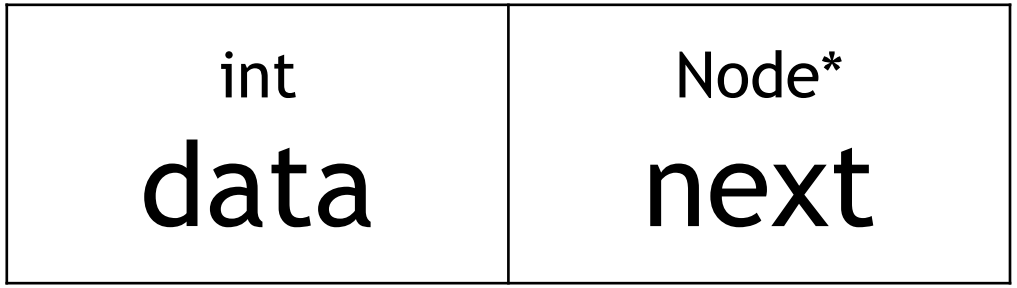


Linked List

Each ***Node*** consists of some data, not necessarily just one variable, or one type

Often the ***Link*** is also included as a data member of the node

Structures and Classes allow us to define custom data types and objects



Linked List

```
struct Node
{
    int data;
    Node* next;
};
```

```
class Node
{
public:
    int data;
    Node* next;
};
```

Exercise

Array Doubling