COMP 1020 - Searching and sorting algorithms

UNIT 5

What is searching?

- Searching: Looking through an array/ArrayList/linked list/any list for a particular item (the “key” value)
- There are two fundamental algorithms:
  - The linear search (we’ve done this many times)
  - The binary search

Linear search

- The linear search
  - Searches from beginning to end
  - It has to look at each item one after the other until the key is found
  - Does not require the list to be sorted

Linear search: code

- A basic linear search:

```
int linearSearch(int[] list, int key){
    /* Search for key within the list. If found,
    * return its position (index), if not
    * return -1. */
    for(int index=0; index < list.length; index++)
        if(list[index] == key)
            return index;
    return -1;
}//linearSearch
```

Linear search analysis

- A linear search takes linear time to run
  - i.e. it will do a number of operations that is linear in comparison with the size of the input array/list
    - because it always goes through all the positions in the array/list until it finds a match

Binary search

- The binary search
  - Divides the list in half repeatedly
  - Fast
  - Requires the list to be sorted
  - Requires fast random access to the list

Binary search

- Imagine searching, by hand, a list of 30,000 student names for “Zach Williams” using a linear search!
  - “Aaron Adams”? No.
  - “Aivee Albert”? No.
  - ……

Binary search

- If the list is sorted there's a much faster way: a binary search
  - This is actually what you do naturally when you search in a dictionary (a real dictionary, you know, with pages)!

Binary search

- If the list is sorted there's a much faster way: a binary search
  - This is actually what you do naturally when you search in a dictionary (a real dictionary, you know, with pages)!
- Basic idea: At every point in the search, keep track of the section of the array, from list[lo] to list[hi], where the key might be
- Initially lo=0, hi=size-1 (the whole array)

Binary search

- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]

Binary search

- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]
  - Pick the middle element in that range
    - list[mid] where mid = (lo+hi)/2

Binary search

- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]
  - Pick the middle element in that range
    - list[mid] where mid = (lo+hi)/2
  - If list[mid]==key, you're done, of course

Binary search

- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]
  - Pick the middle element in that range
    - list[mid] where mid = (lo+hi)/2
  - If list[mid]==key, you're done, of course
  - If list[mid]>key, then
    - Key can only be from list[lo] to list[mid-1]
      - Everything above list[mid] must be too big, too
    - Change hi = mid-1

Binary search

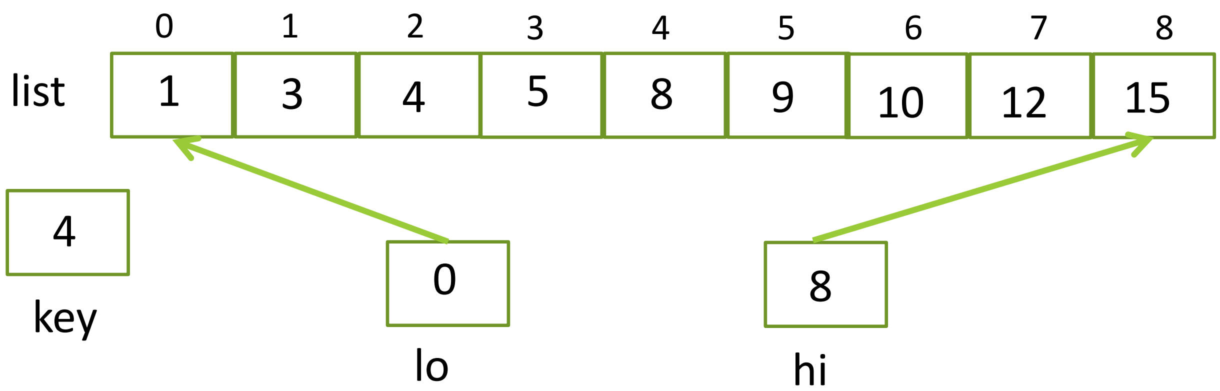
- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]
  - Pick the middle element in that range
    - list[mid] where mid = (lo+hi)/2
  - If list[mid]==key, you're done, of course
  - If list[mid]>key, then
    - Key can only be from list[lo] to list[mid-1]
      - Everything above list[mid] must be too big, too
    - Change hi = mid-1
  - Similarly if list[mid]<key change lo to mid+1

Binary search

- One step of a binary search:
  - Trying to find key in the portion of the list from list[lo] to list[hi]
  - Pick the middle element in that range
    - list[mid] where mid = (lo+hi)/2
  - If list[mid]==key, you're done, of course
  - If list[mid]>key, then
    - Key can only be from list[lo] to list[mid-1]
      - Everything above list[mid] must be too big, too
    - Change hi = mid-1
  - Similarly if list[mid]<key change lo to mid+1
- Keep going until you find key, or run out of places to look (when lo>hi)

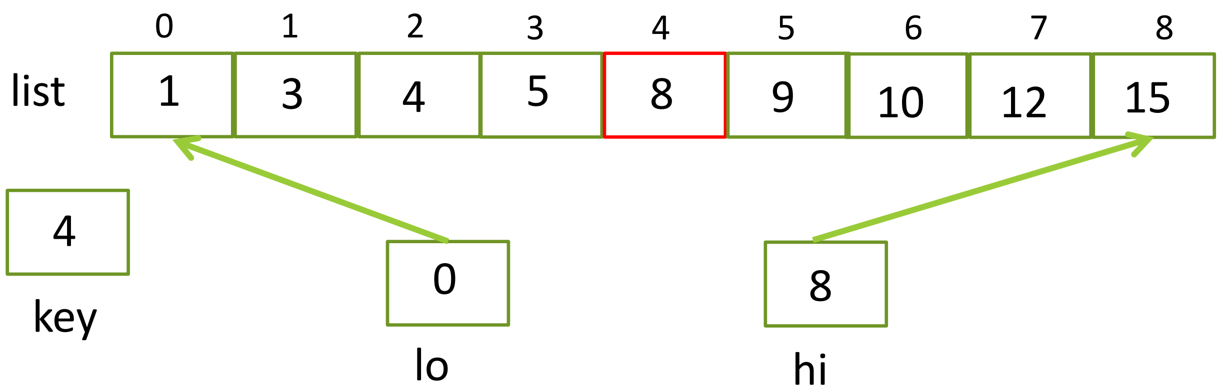
Binary search - Example

- Search this array:



Binary search - Example

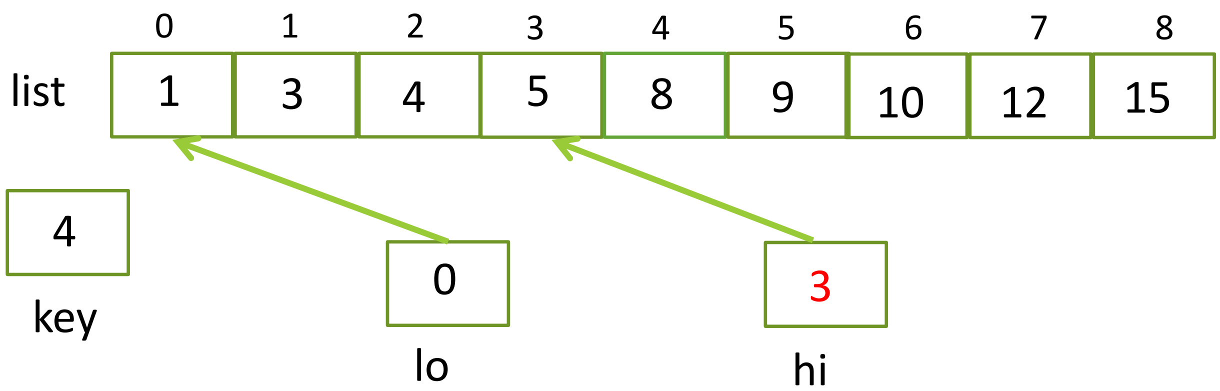
- Search this array:



- Find the middle: mid = (lo+hi)/2 = (0+8)/2 = 4
- Check list[4], which is 8

Binary search - Example

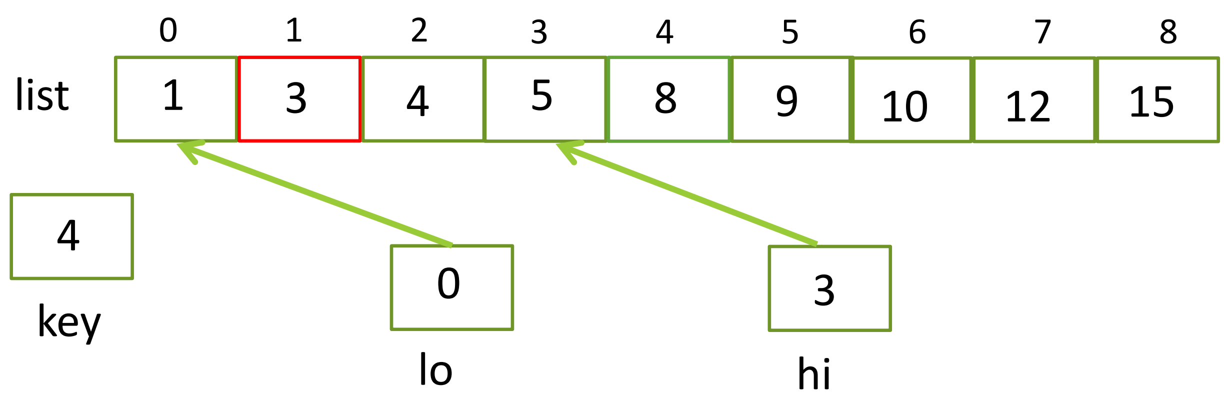
- Search this array:



- Find the middle: mid = (lo+hi)/2 = (0+8)/2 = 4
- Check list[4], which is 8
- It’s too big – change hi to mid-1 = 3

Binary search - Example

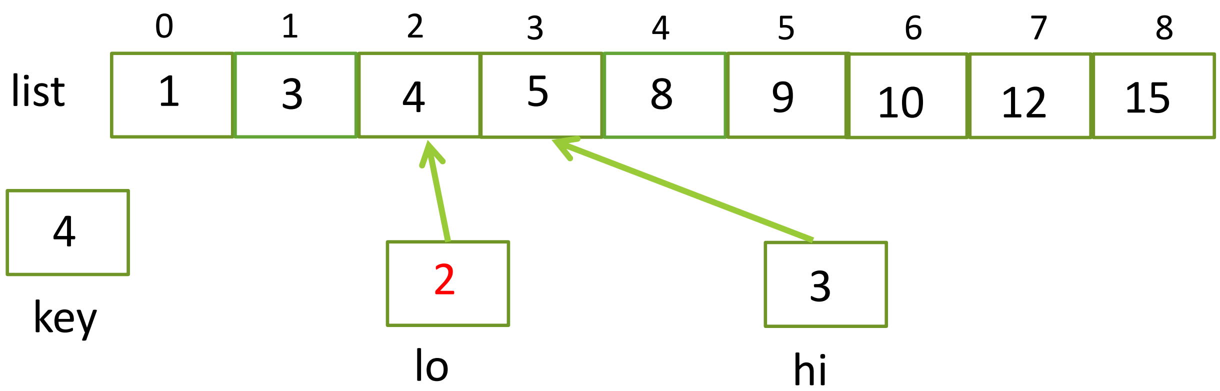
- Search this array:



- Find the middle: mid = (lo+hi)/2 = (0+3)/2 = 1
- Check list[1], which is 3

Binary search - Example

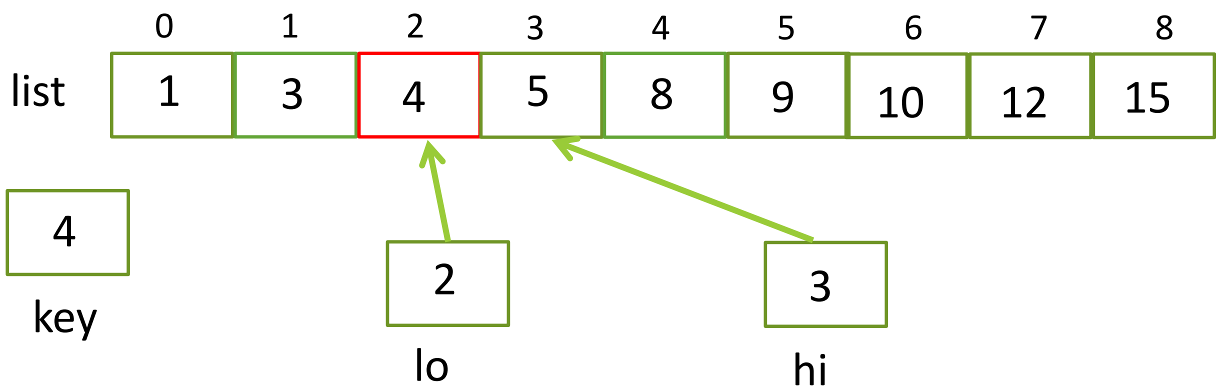
- Search this array:



- Find the middle: mid = (lo+hi)/2 = (0+3)/2 = 1
- Check list[1], which is 3
- It’s too small – change lo to mid+1 = 2

Binary search - Example

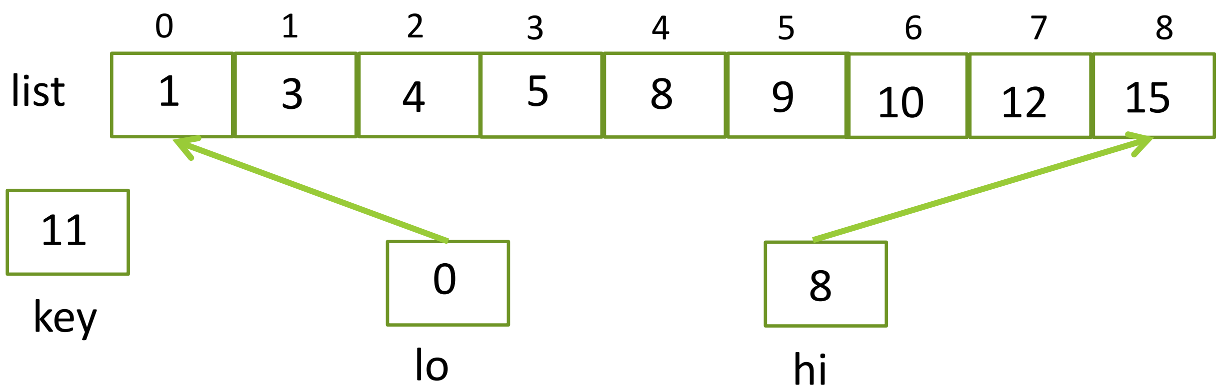
- Search this array:



- Find the middle: mid = (lo+hi)/2 = (2+3)/2 = 2
- Check list[2], which is 4
- Found!

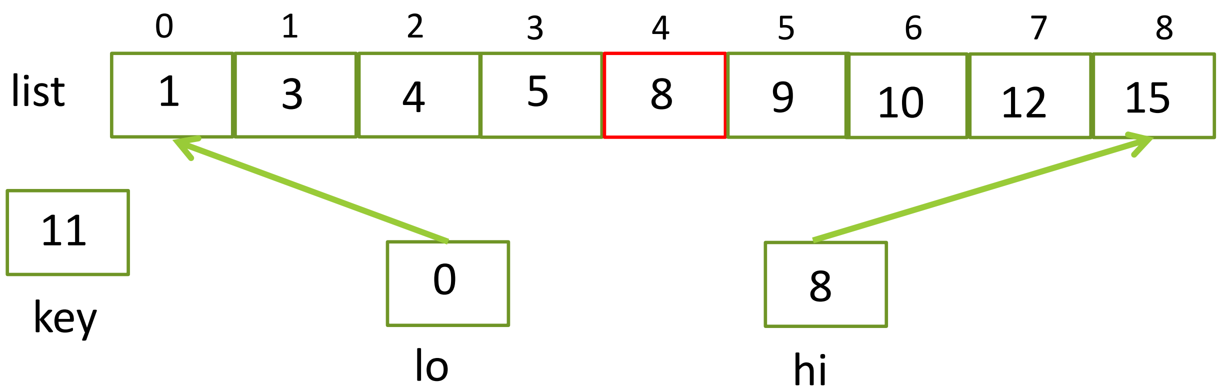
Binary search - Example

- What if the key isn't there?



Binary search - Example

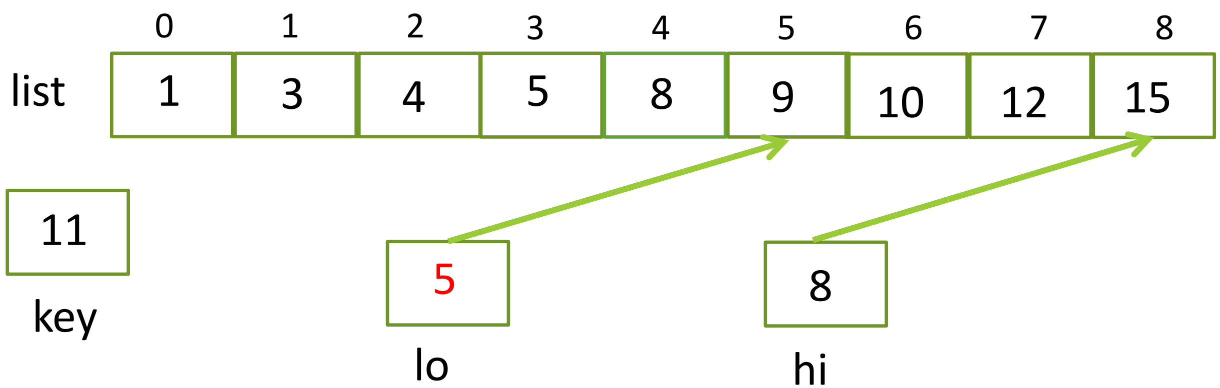
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (0+8)/2 = 4
- Check list[4], which is 8

Binary search - Example

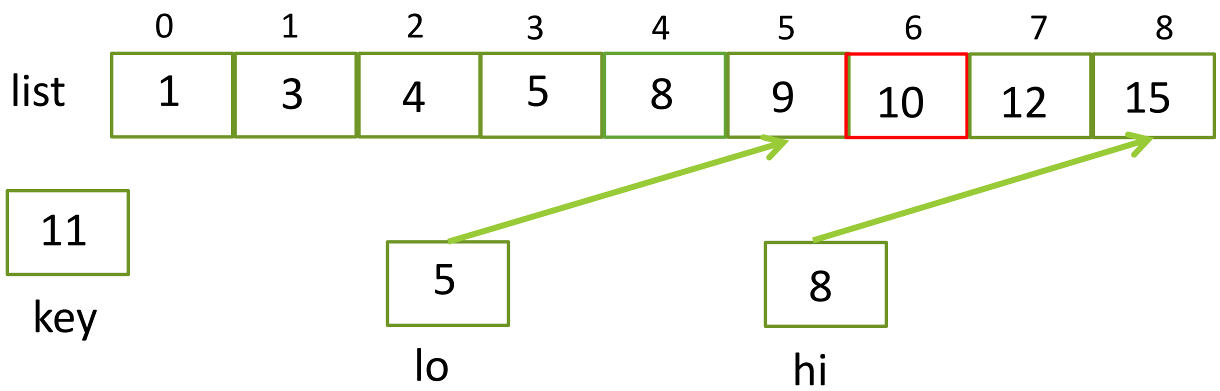
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (0+8)/2 = 4
- Check list[4], which is 8
- It's too small - change lo to mid+1 = 5

Binary search - Example

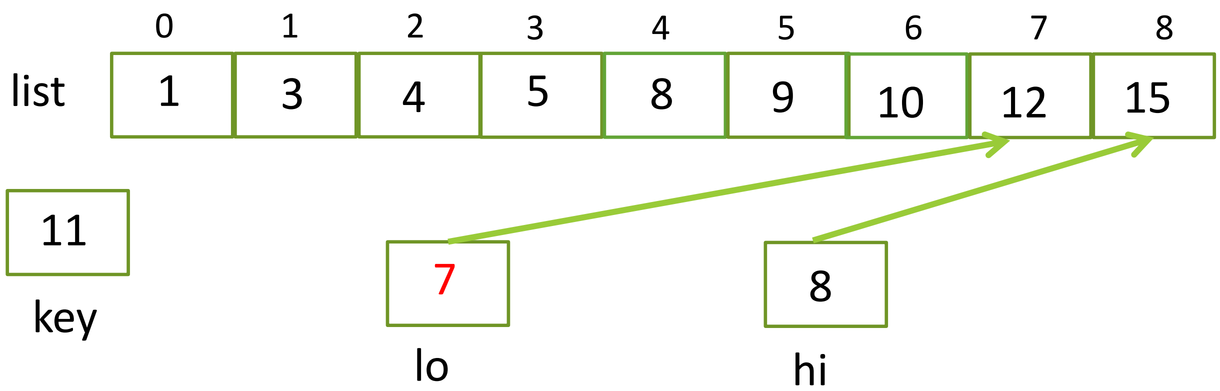
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (5+8)/2 = 6
- Check list[6], which is 10

Binary search - Example

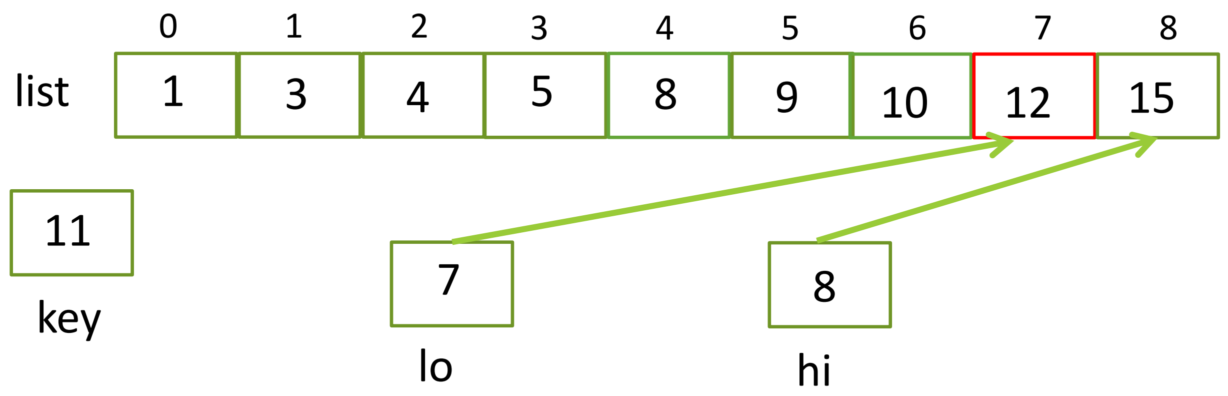
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (5+8)/2 = 6
- Check list[6], which is 10
- It's too small - change lo to mid+1 = 7

Binary search - Example

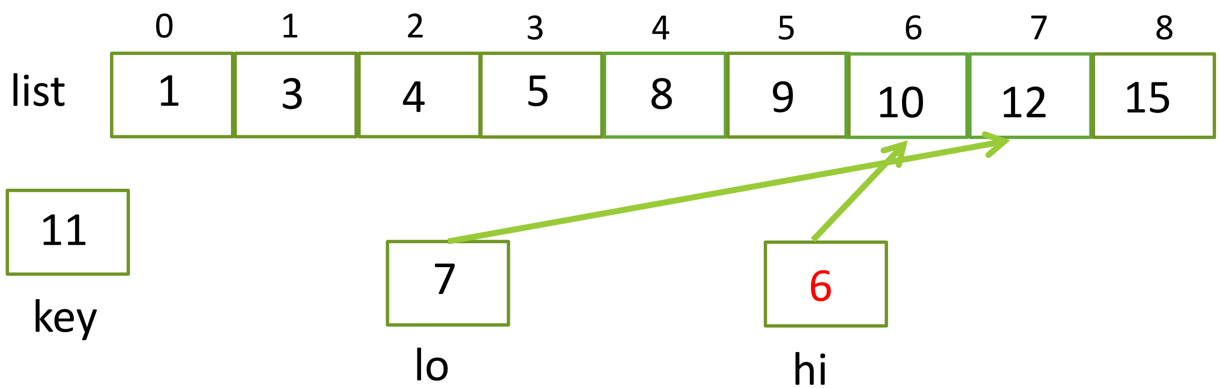
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (7+8)/2 = 7
- Check list[7], which is 12

Binary search - Example

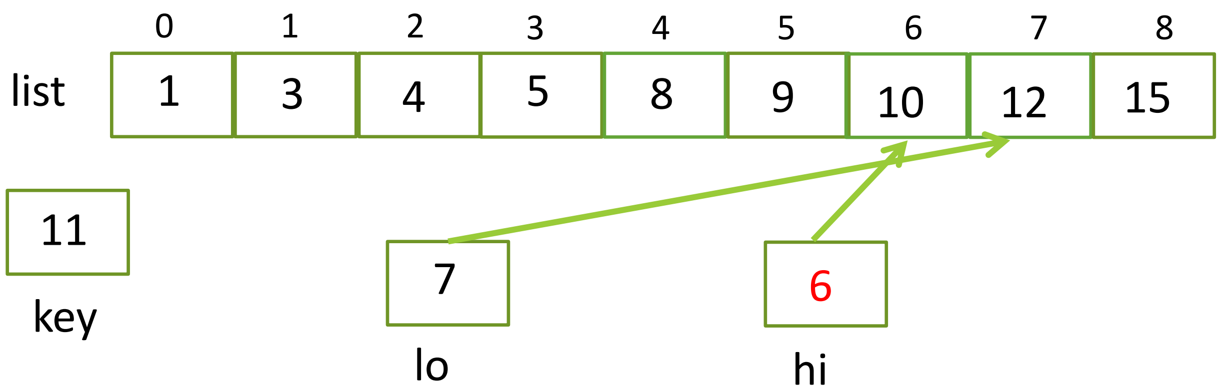
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (7+8)/2 = 7
- Check list[7], which is 12
- It's too big - change hi to mid-1 = 6

Binary search - Example

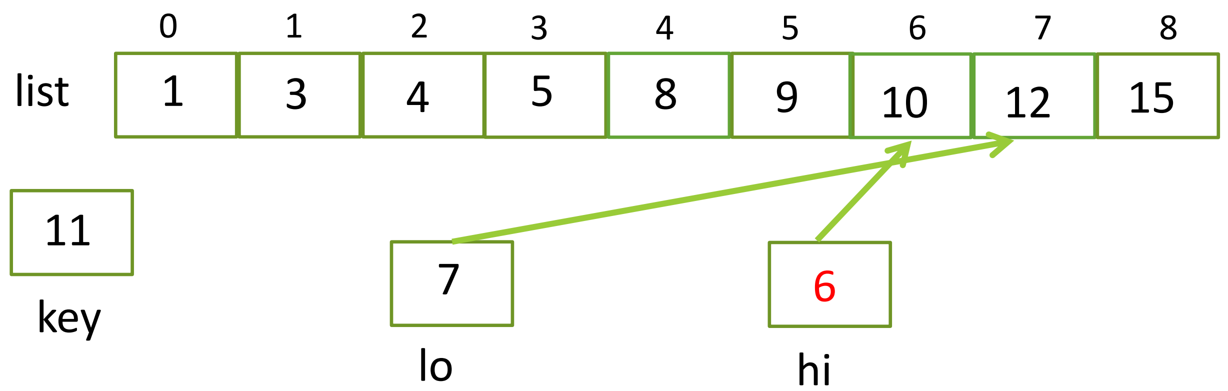
- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (7+8)/2 = 7
- Check list[7], which is 12
- It's too big - change hi to mid-1 = 6
  - Now hi < lo! → impossible to continue!
  - we know now that the key is not in the table

Binary search - Example

- What if the key isn't there?



- Find the middle: mid = (lo+hi)/2 = (7+8)/2 = 7
- Check list[7], which is 12
- It's too big - change hi to mid-1 = 6
  - Now hi < lo! → impossible to continue!
  - we know now that the key is not in the table

Let's write the code for this!

Binary search code (iterative)

```
public static int binarySearch(int[] list, int key){
    int lo=0;
    int hi=list.length-1;
    int mid;
    while(lo<=hi){
        mid=(lo+hi)/2;
        if(list[mid]==key)
            return mid;
        else if(list[mid]<key)
            lo=mid+1;
        else
            hi=mid-1;
    }//while
    return -1;
}//binarySearch
```

Relative speed comparison

- The difference between the two algorithms is huge!

- If you double the list size:
  - The linear search doubles the iterations
  - The binary search adds only 1 iteration!

List Size	Linear iterations	Binary iterations
10	10	4
20	20	5
1000	1000	10
1,000,000	1,000,000	20
1,000,000,000	1,000,000,000	30
	(max)	(max)

When to use a binary search

- The binary search needs a sorted list
  - You can keep the list in order as you build it
  - Or take an existing list and sort it

When to use a binary search

- The binary search needs a sorted list
  - You can keep the list in order as you build it
  - Or take an existing list and sort it

- Sorting a list (or keeping a list sorted) is even slower than a linear search…

  - So why bother?

When to use a binary search

- Use a binary search if:
  - You happen to have a sorted list already
  - You plan to do a LOT of searching
    - But the list doesn't change much, so keeping it sorted is easy
  - You have lots of time to sort (overnight, maybe), but when they happen, the searches need to be FAST!

Sorting an array?

- Now, let’s see how we can sort an existing array
- There exists many different algorithms, and most of them have the advantage of sorting the array in-place, i.e. they don’t require any additional space!
- We will briefly see some of them (not all of them)

Simple but slow sorting algos

- Insertion sort
  - The best one to use by default
- Selection sort
  - Also usable
  - Good when moving the data around is expensive
    - It does the fewest data movements (but more comparisons)
- Bubble sort
  - Very simple to code
  - Rarely useful, except as an example…

Bubble Sort

```
public static void bubbleSort(int data[], int n)
{
    for (int pass = 0; pass < n ; pass++) {
        for (int j = 0; j < n - 1 - pass; j++) {
            if (data[j] > data[j + 1]) {
                int temp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = temp;
            }           }       }
```

Let’s see some sorting algos

- We’ll start by taking a look at the simpler algorithms
  - Insertion sort
  - Selection sort
- The main idea of these in-place sorting algorithms is to separate the array into two parts: a sorted part (generally at the beginning) and an unsorted part (generally at the end) and gradually increase the size of the sorted part until everything is sorted

Ordered insertion?

- If we know we might need a sorted array, why not try to keep it ordered at all times, after each insertion?
  - The idea is: always keep it sorted as you're creating it

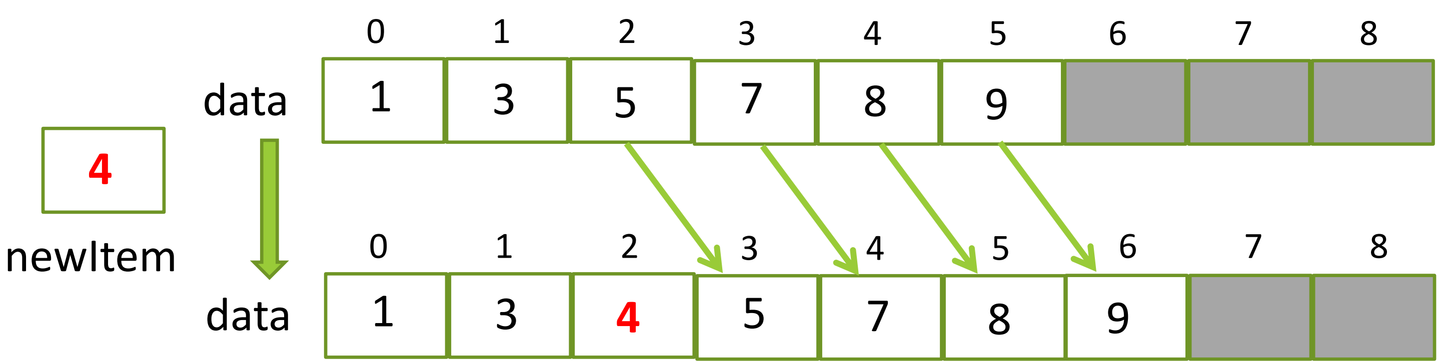
- We have seen this already in previous weeks, but let's look at it one more time

Ordered Insertion – on arrays

- When adding a new element to a (partially-filled) array
  - the old way (adding it to the end) won't work:
    data[numItems++] = newItem;

Ordered Insertion – on arrays

- When adding a new element to a (partially-filled) array
  - the old way (adding it to the end) won't work:
    data[numItems++] = newItem;
- We must now insert it into the proper spot to keep the array sorted (an "ordered insert" – slower, harder):



Ordered Insertion – code

- Assume data[0]..data[n-1] are there, and in order
- Insert newItem so that data[0]..data[n] are in order

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1; //Must start at the high end!
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {  //process larger ones
            data[index+1]=data[index]; //move them up 1 spot
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem; //index is a smaller item (or
}//ordInsert                  // -1). newItem goes next to it.
```

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	**5**

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	8	9			

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	5

spotFound	false
index	5

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	8	9			

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n: 6

newItem: 5

spotFound: false

index: 5

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	8	9	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	**5**

spotFound	false
index	**4**

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	8	9	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	**5**

spotFound	false
index	**4**

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	8	8	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	**5**

spotFound	false
index	**3**

data

0	1	2	3	4	5	6	7	8
1	2	4	7	8	8	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	5

spotFound	false
index	3

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	7	8	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	5

spotFound	false
index	2

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	7	8	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	**5**

spotFound	true
index	**2**

	0	1	2	3	4	5	6	7	8
data	1	2	4	7	7	8	9		

Ordered Insertion – example

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	6
newItem	5

spotFound	true
index	2

	0	1	2	3	4	5	6	7	8
data	1	2	4	5	7	8	9		

Ordered Insertion – now insert 0

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	7
newItem	**0**

	0	1	2	3	4	5	6	7	8
data	1	2	4	5	7	8	9		

Ordered Insertion – now insert 0

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	7
newItem	**0**

spotFound	false
index	**6**

	0	1	2	3	4	5	6	7	8
data	1	2	4	5	7	8	9		

Ordered Insertion – now insert 0

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n: 7

newItem: **0**

spotFound: false

index: **-1**

	0	1	2	3	4	5	6	7	8
data	1	1	2	4	5	7	8	9	

Ordered Insertion – now insert 0

```
public static void ordInsert(int n, int[] data, int newItem){
    int index = n-1;
    boolean spotFound = false;
    while(index>=0 && !spotFound)
        if(data[index] > newItem) {
            data[index+1]=data[index];
            index--; }
        else
            spotFound = true;
    data[index+1]=newItem;
}
```

n	7
newItem	**0**

spotFound	false
index	**-1**

	0	1	2	3	4	5	6	7	8
data	**0**	1	2	4	5	7	8	9	

Insertion sort

- If you just want one simple sorting algorithm, this should be the one
- It is relatively easy to implement
- It runs reasonably fast

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

3	9	1	7	2	5

When you start the insertion sort, initially the sorted part has only one element, the first one.

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

3	9	1	7	2	5

Then, at each step you choose the first element of the unsorted part, and put it in the correct place!

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

3	9	1	7	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

3	9	1	7	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

3	1	9	7	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	9	7	2	5

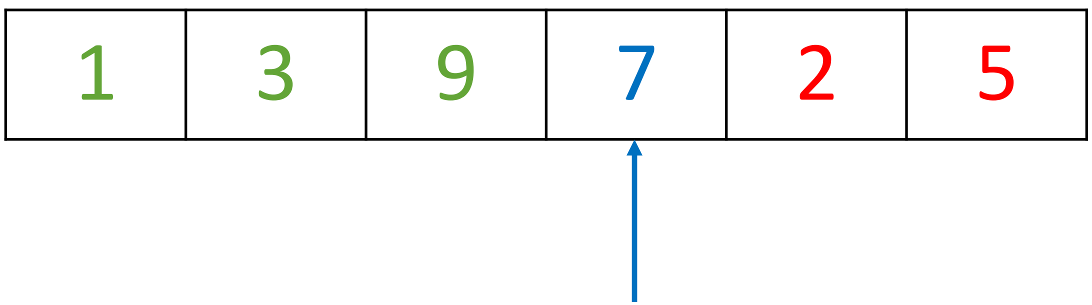
Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	9	7	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part



Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	7	9	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	7	9	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	7	9	2	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	7	2	9	5

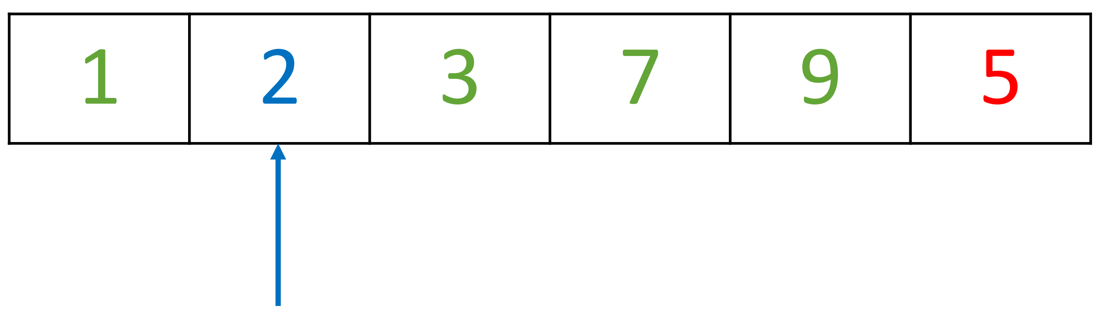
Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	3	2	7	9	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part



Insertion sort example

- The idea of insertion sort is, at each step:
    - Insert the first element of the unsorted part in the correct position of the sorted part

1	2	3	7	9	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	2	3	7	9	5

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	2	3	7	5	9

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	2	3	5	7	9

Insertion sort example

- The idea of insertion sort is, at each step:
  - Insert the first element of the unsorted part in the correct position of the sorted part

1	2	3	5	7	9

Done!

Insertion sort

- Basic concept:
  - To sort a[0] to a[n-1], simply use an ordered insert to gradually re-build the list in sorted order:
    - The single-element list a[0] to a[0] is sorted (obviously, a single element is ordered)
    - Insert a[1] to make a[0] to a[1] sorted.
    - Insert a[2] to make a[0] to a[2] sorted.
    - …
    - Insert a[n-1] to make a[0] to a[n-1] sorted. Done!

Insertion sort

- Using our previous ordInsert method (yes, the same one!), the code is simply:

```
for(int k=1; k < n; k++)
    ordInsert(k, a, a[k]);    //Insert a[k] to make
                              // a[0]..a[k] sorted
```

size of the array before insertion (in this case, size of the sorted part)

array

element to insert

Insertion sort - rough analysis

- The insertion sort is:
```
for(int k=1; k < n; k++)
    ordInsert(k, a, a[k]);
```
- It contains one simple loop that always runs n times
- Inside that loop it does an ordered insertion, which is O(n) – it does n steps
  - Actually, it does 1, 2, 3, 4, ..., n steps
  - But that's, on average, n/2, which is O(n) anyway
- So we do n steps, n times: this is $O(n^2)$

Selection sort

- Selection sort is another sort that is easy to implement

- It is a little bit slower than insertion sort in practice, even though the maximum number of operations (worst-case scenario) is the same

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

3	9	1	7	2	5

When you start the selection sort, initially the sorted part is empty, everything is unsorted!

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

3	9	1	7	2	5

You must find the smallest element of the unsorted part, and swap it with the first element of the unsorted part!

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	9	3	7	2	5

After the swap, this increases the size of the sorted part by 1!

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	9	3	7	2	5

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	7	9	5

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	7	9	5

In the right spot, no need to swap!

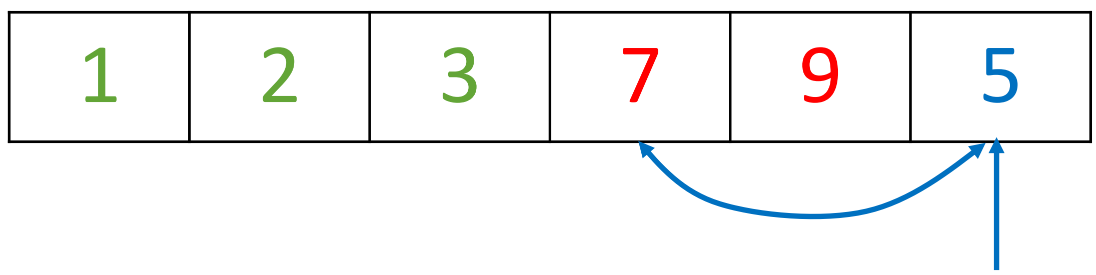
Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	7	9	5

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)



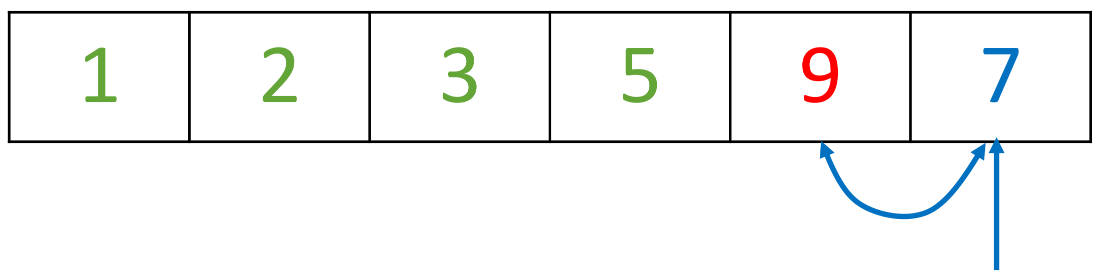
Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	5	9	7

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)



Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	5	7	9

Selection sort example

- The idea of selection sort is, at each step:
  - Select the smallest element from the unsorted part, and swap it with the element that is after the end of the sorted part (i.e. the first element of the unsorted part)

1	2	3	5	7	9

The last element does not need to be sorted, it's always in the right spot!

Selection sort

- Another simple sorting idea (again, we're sorting a[0] to a[n-1]):
  - Search a[0]..a[n-1] for the smallest element
    - Swap it into position a[0]
  - Search a[1]..a[n-1] for the next smallest one
    - Swap it into position a[1]
  - Search a[2]..a[n-1] for the next smallest one
    - Swap it into position a[2]
  - …
  - Search a[n-2]..a[n-1] for the next smallest one
    - Swap it into position a[n-2]
  - No need to search a[n-1]..a[n-1] – that's only one left! Done.

Selection sort

- We clearly need a loop:

```
for(int k=0; k<=n-2; k++) {
    //Find the smallest number from a[k] to a[n-1]
    int min = ??;   //The smallest number
    int where = ??; //it was found in a[where]
    ..
    //Swap a[k] and min, which was found in a[where]
    ..
}//for
```

Selection sort

- We clearly need a loop:

```
for(int k=0; k<=n-2; k++) {
    //Find the smallest number from a[k] to a[n-1]
    int min = ??;   //The smallest number
    int where = ??; //it was found in a[where]
    ..
    //Swap a[k] and min, which was found in a[where]
    a[where] = a[k];
    a[k] = min;
}//for
```

Selection sort

- We clearly need another loop:

```
for(int k=0; k<=n-2; k++) {
    //Find the smallest number from a[k] to a[n-1]
    int min = a[k];   //The smallest number
    int where = k; //it was found in a[where]
    for (int i = k+1; i < n; i++) {
        if (a[i] < min) {  //new min!
            min = a[i];  where = i;
        }
    }
    //Swap a[k] and min, which was found in a[where]
    a[where] = a[k];
    a[k] = min;
}//for
```

Selection sort

- Make it a method:

```
public static void selectionSort(int[] a){
    for(int k=0; k<=a.length-2; k++) {
        //Find the smallest number from a[k] to a[n-1]
        int min = a[k];   //The smallest number
        int where = k; //it was found in a[where]
        for (int i = k+1; i < a.length; i++) {
            if (a[i] < min) {  //new min!
                min = a[i];  where = i;
            }
        }
        //Swap a[k] and min, which was found in a[where]
        a[where] = a[k];
        a[k] = min;
    }//for
}//selectionSort
```