Introduction to Programming

Week – *5*, Lecture – *1*

Iterative Operations on Arrays

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

What we know about arrays till now…

Arrays are collections of variables of the same type

What we know about arrays till now…

Arrays are collections of variables of the same type

Individual variables are accessed through an index, which starts with 0

What we know about arrays till now…

Arrays are collections of variables of the same type

Individual variables are accessed through an index, which starts with 0

The size of an array remains fixed, after its creation

What we know about arrays till now…

Arrays are collections of variables of the same type

Individual variables are accessed through an index, which starts with 0

The size of an array remains fixed, after its creation

We can initialize the array by providing the values for different variables

What we know about arrays till now…

Arrays are collections of variables of the same type

Individual variables are accessed through an index, which starts with 0

The size of an array remains fixed, after its creation

We can initialize the array by providing the values for different variables

Usually, a loop is used to “iterate” through these variables, and use them appropriately

What we know about arrays till now…

Arrays are collections of variables of the same type

Individual variables are accessed through an index, which starts with 0

The size of an array remains fixed, after its creation

We can initialize the array by providing the values for different variables

Usually, a loop is used to “iterate” through these variables, and use them appropriately

We will now have a look at two most common operations performed through loops on an array

- Sorting the values in an array
- Searching for a value in an array

Example – Cards in your hands

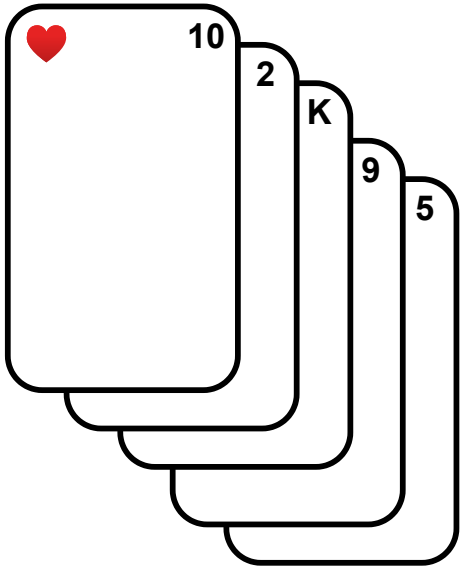
Have you played a Card game, where cards are numbered?

Example – Cards in your hands

Have you played a Card game, where cards are numbered?

In such games, players usually try to keep their cards in increasing or decreasing order

Example – Cards in your hands

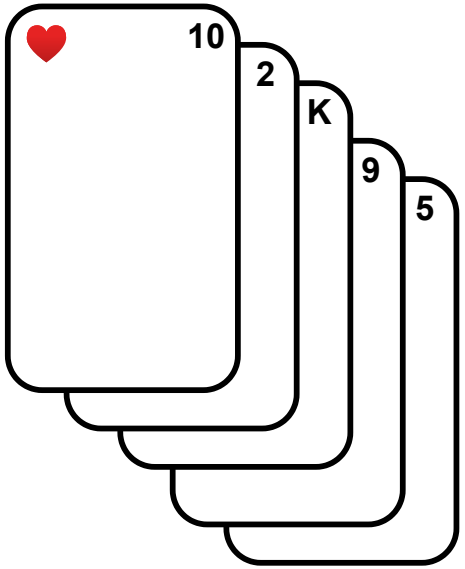


Have you played a Card game, where cards are numbered?

In such games, players usually try to keep their cards in increasing or decreasing order

But when they get these cards, they are in some random order, as shown in the figure

Example – Cards in your hands



Have you played a Card game, where cards are numbered?

In such games, players usually try to keep their cards in increasing or decreasing order

But when they get these cards, they are in some random order, as shown in the figure

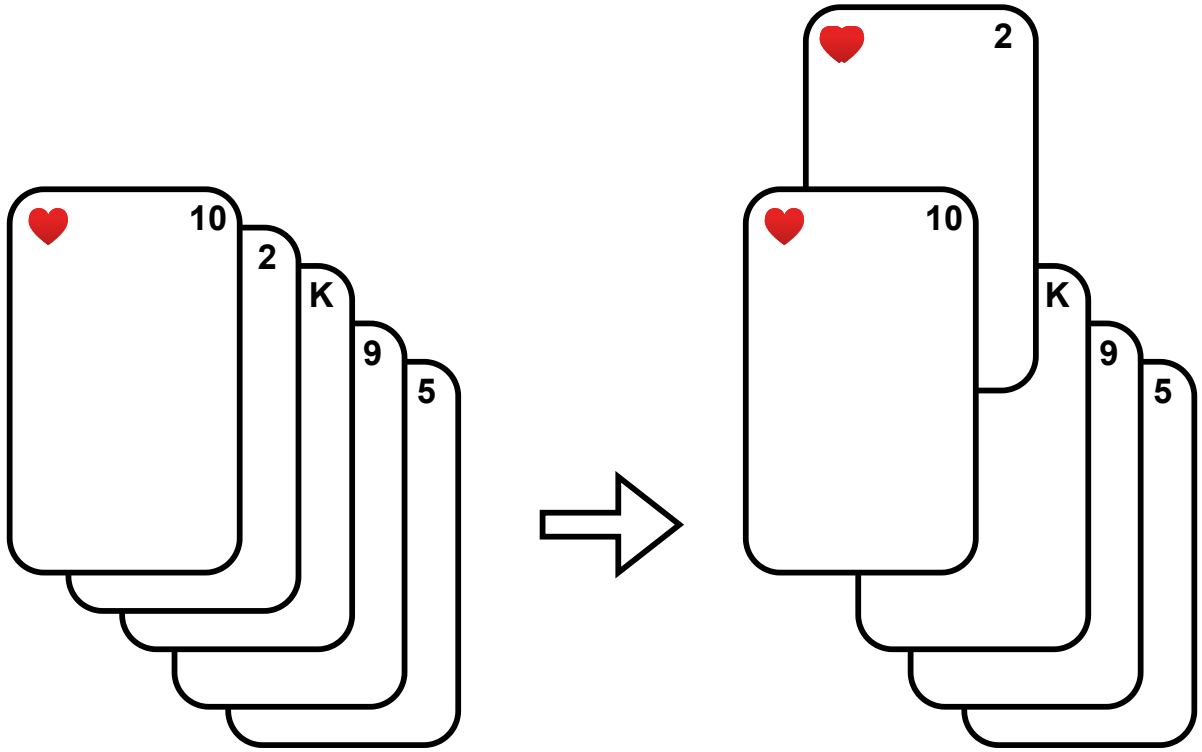
How do they usually do it?

“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand

“Sorting” the cards in hand

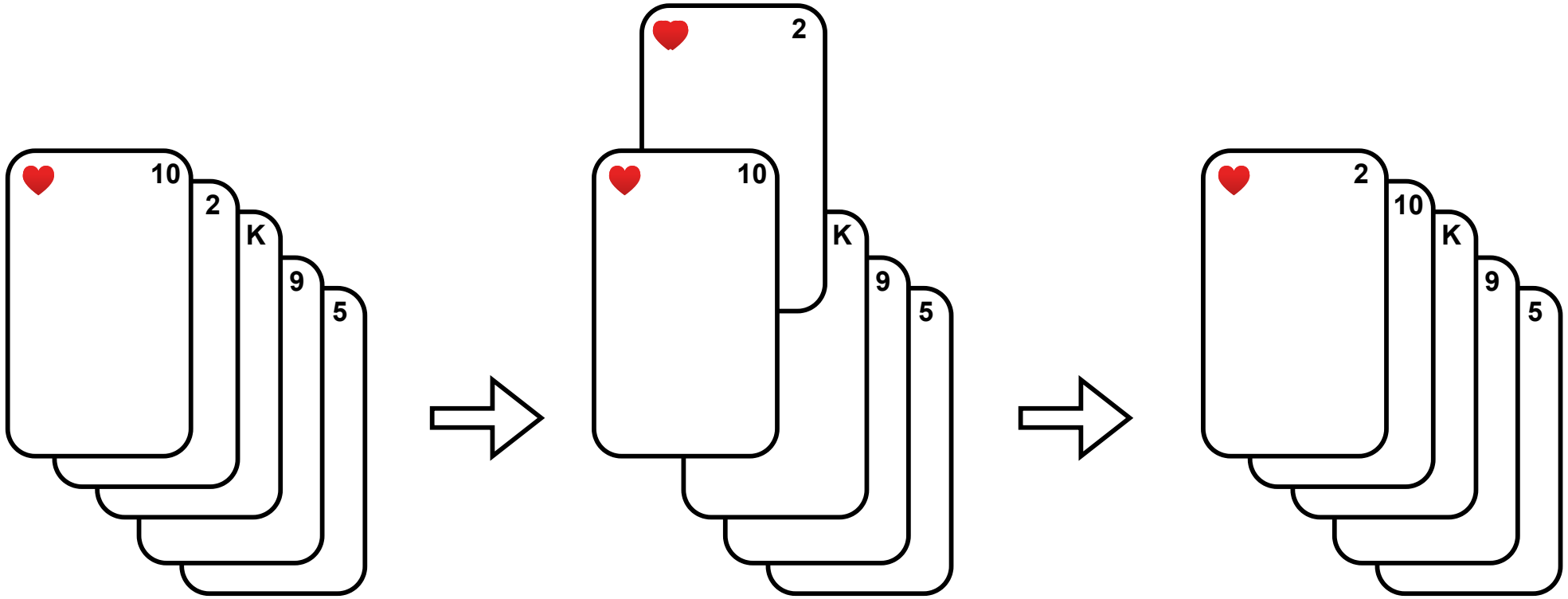


“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top

“Sorting” the cards in hand

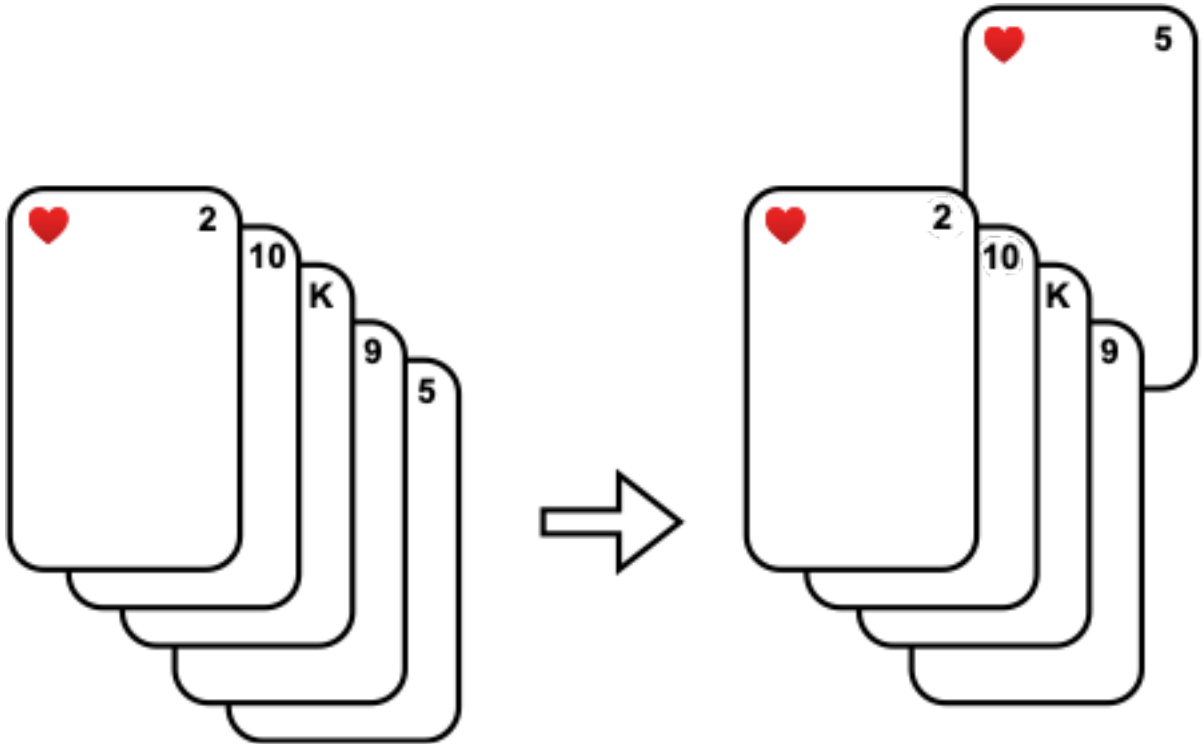


“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them

“Sorting” the cards in hand

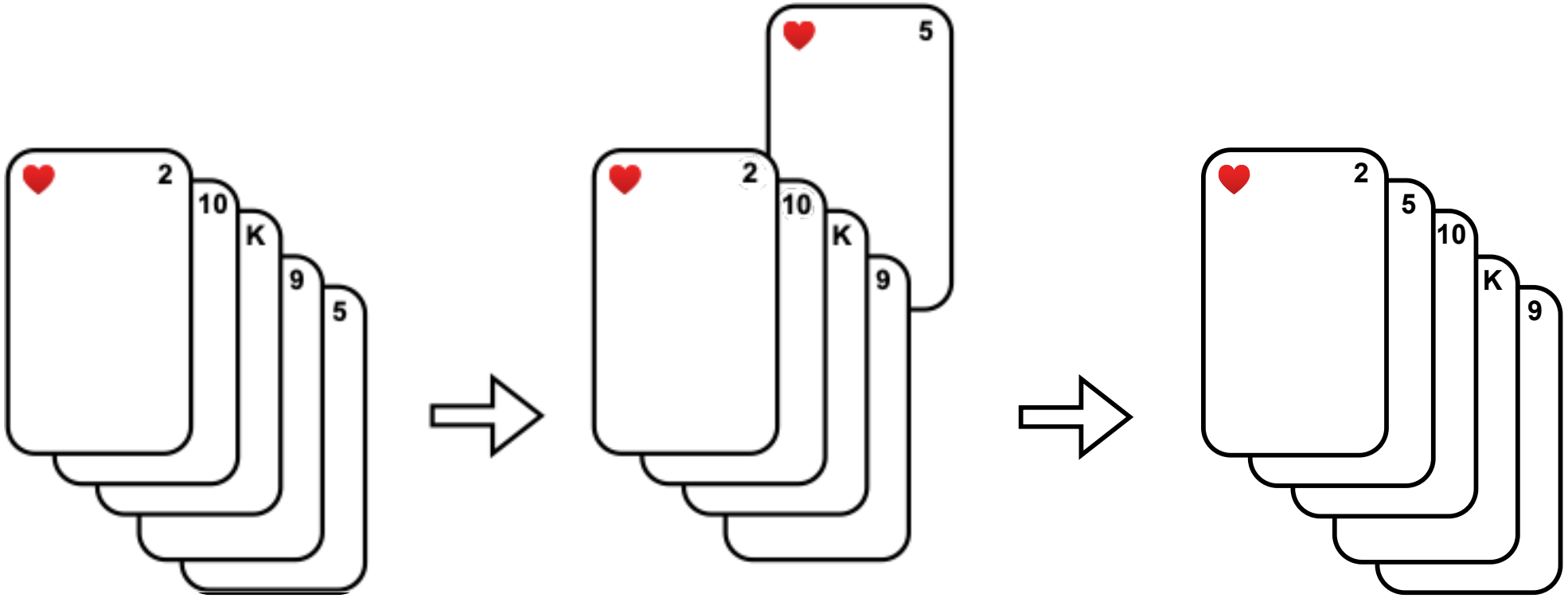


“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them
- Pick out the card, and place it just below the card that was picked previously

“Sorting” the cards in hand



“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them
- Pick out the card, and place it just below the card that was picked previously
- Keep repeating this process, and you’ll end up with the cards in order

“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them
- Pick out the card, and place it just below the card that was picked previously
- Keep repeating this process, and you’ll end up with the cards in order

The process of putting things in some order is called *sorting*

“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them
- Pick out the card, and place it just below the card that was picked previously
- Keep repeating this process, and you’ll end up with the cards in order

The process of putting things in some order is called *sorting*

Assuming that the set of cards represent an array, with each card represents a variable

- … sorting an array involves re-shuffling of the values that the array stores
- … in order to bring it in “some” order

“Sorting” the cards in hand

A common approach to get the cards in order (assuming the smallest card is on top) is this:

- Browse though the cards, and find the smallest card in the hand
- Pick out the card, and bring it on top
- Browse through the rest of the cards, and find the smallest card among them
- Pick out the card, and place it just below the card that was picked previously
- Keep repeating this process, and you’ll end up with the cards in order

The process of putting things in some order is called *sorting*

Assuming that the set of cards represent an array, with each card represents a variable

- … sorting an array involves re-shuffling of the values that the array stores
- … in order to bring it in “some” order

What we just saw included examples of two common array operations

- *“inserting”* a value and *“deleting”* the value at a given index (in this case, it was the same value)

Insertions in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertions in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value

Insertions in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value
- Picking an index in the array to “insert” the new value; let the index be `i`

Insertions in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value
- Picking an index in the array to “insert” the new value; let the index be `i`
- Shifting elements at index `i` and and all “valid” indices greater than `i`, to the right

Insertions in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value
- Picking an index in the array to “insert” the new value; let the index be `i`
- Shifting elements at index `i` and and all “valid” indices greater than `i`, to the right
- Inserting the new value at index `i`

Insertion in arrays

Capacity: 5

10	13	9	5	X

Size	**Vacant Slots?**
4	Yes

Insertion in arrays

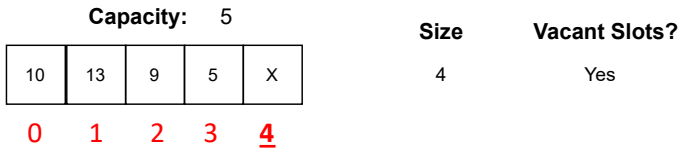
Capacity: 5

10	13	9	5	X

Size	Vacant Slots?
4	Yes

Assume that we have an array of capacity 5 and current size as 4

Insertion in arrays

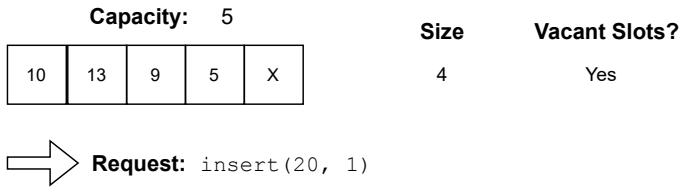


Size	Vacant Slots?
4	Yes

Assume that we have an array of capacity 5 and current size as 4

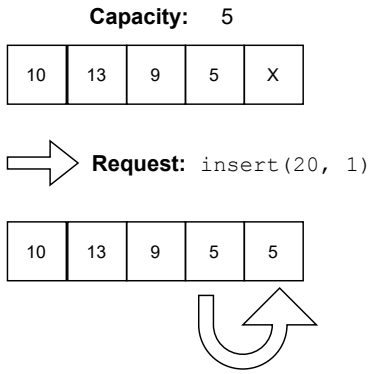
Here, there is no “recognized” value at index 4 (i.e., it could be some random value, that we don’t care)

Insertion in arrays



Assume that we now get a request to insert a new value, `20`, at index `1` of the array

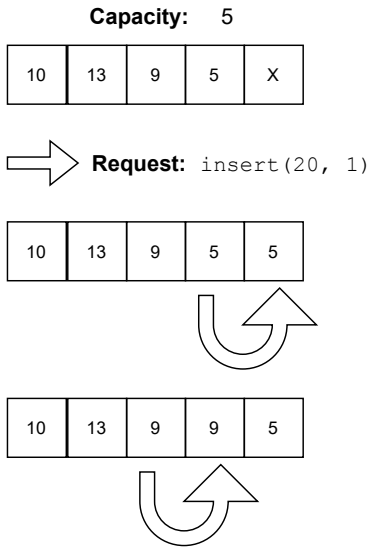
Insertion in arrays



Size	Vacant Slots?
4	Yes

We copy the value at last occupied index, i.e., index 3 to the next index, i.e., index 4

Insertion in arrays

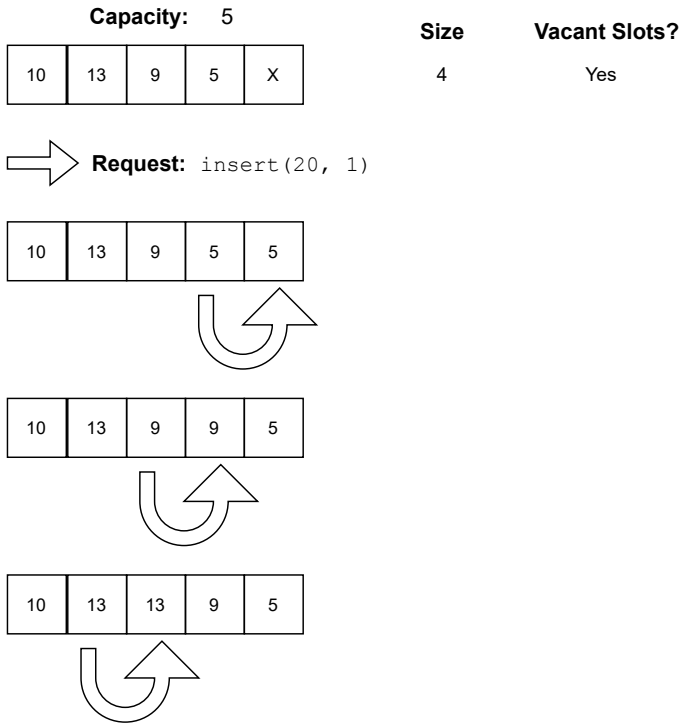


Size	Vacant Slots?
4	Yes

We copy the value at last occupied index, i.e., index 3 to the next index, i.e., index 4

… then we copy the value at index 2 to index 3

Insertion in arrays

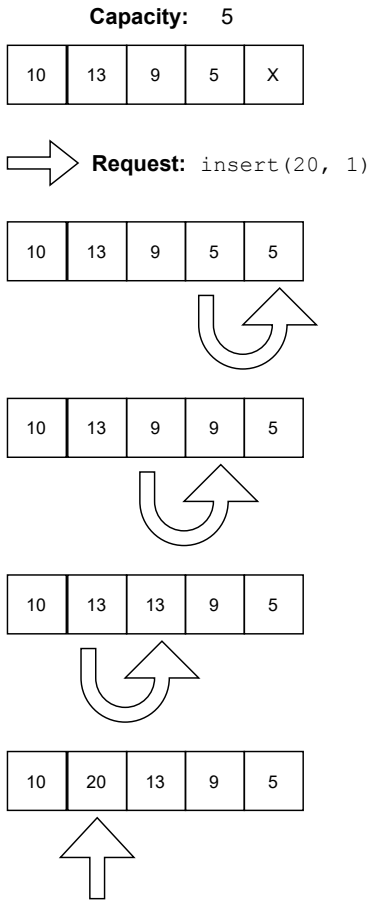


We copy the value at last occupied index, i.e., index 3 to the next index, i.e., index 4

… then we copy the value at index 2 to index 3

… and finally, we copy value at index 1 to index 2

Insertion in arrays



Size	Vacant Slots?
4	Yes

Size	Vacant Slots?
5	No

Finally, we can now insert new value at index 1

Insertion in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value
- Picking an index in the array to “insert” the new value; let the index be `i`
- Shifting elements at index `i` and and all “valid” indices greater than `i`, to the right
- Inserting the new value at index `i`

Here, we are explicitly maintaining the size of an array

Insertion in arrays

In the card sorting example, we *inserted* a card at the appropriate location in each step

Insertion involves

- Checking if the array has any vacant slots; if there are, we can accommodate a new value
- Picking an index in the array to "insert" the new value; let the index be `i`
- Shifting elements at index `i` and and all "valid" indices greater than `i`, to the right
- Inserting the new value at index `i`

Here, we are explicitly maintaining the size of an array

It can range from 0 to the array capacity

- Array capacity is essentially its declared size

Insertion in arrays

```
// Check for vacancy
if(size == CAPACITY)
{
        printf("The array is full; remove some elements first !\n");
        break;
}

printf("Enter the index where you would like to insert the value: ");
scanf("%d", &op1);

// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 > size)
{
        printf("WARNING: The element will be added at the end, i.e. index %d\n",size);
        op1 = size;
}

printf("Enter the value to insert: ");
scanf("%d", &op2);

// Shift elements to the right
for(i = size; i > op1; i--)
        arr[i] = arr[i-1];

// Insert the new element
arr[op1] = op2;

// Update the size
size++;
```

Insertion in arrays

```c
// Check for vacancy
if(size == CAPACITY)
{
        printf("The array is full; remove some elements first !\n");
        break;
}

printf("Enter the index where you would like to insert the value: ");
scanf("%d", &op1);

// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 > size)
{
        printf("WARNING: The element will be added at the end, i.e. index %d\n",size);
        op1 = size;
}

printf("Enter the value to insert: ");
scanf("%d", &op2);

// Shift elements to the right
for(i = size; i > op1; i--)
        arr[i] = arr[i-1];

// Insert the new element
arr[op1] = op2;

// Update the size
size++;
```

This is C code for inserting a value in an array

Insertion in arrays

```
// Check for vacancy
if(size == CAPACITY)
{
        printf("The array is full; remove some elements first !\n");
        break;
}

printf("Enter the index where you would like to insert the value: ");
scanf("%d", &op1);

// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 > size)
{
        printf("WARNING: The element will be added at the end, i.e. index %d\n",size);
        op1 = size;
}

printf("Enter the value to insert: ");
scanf("%d", &op2);

// Shift elements to the right
for(i = size; i > op1; i--)
        arr[i] = arr[i-1];

// Insert the new element
arr[op1] = op2;

// Update the size
size++;
```

`op1` contains the index of insertion

Insertion in arrays

```
// Check for vacancy
if(size == CAPACITY)
{
        printf("The array is full; remove some elements first !\n");
        break;
}

printf("Enter the index where you would like to insert the value: ");
scanf("%d", &op1);

// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 > size)
{
        printf("WARNING: The element will be added at the end, i.e. index %d\n",size);
        op1 = size;
}

printf("Enter the value to insert: ");
scanf("%d", &op2);

// Shift elements to the right
for(i = size; i > op1; i--)
        arr[i] = arr[i-1];

// Insert the new element
arr[op1] = op2;

// Update the size
size++;
```

`op2` contains the value to be inserted

Insertion in arrays

```c
// Check for vacancy
if(size == CAPACITY)
{
        printf("The array is full; remove some elements first !\n");
        break;
}

printf("Enter the index where you would like to insert the value: ");
scanf("%d", &op1);

// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 > size)
{
        printf("WARNING: The element will be added at the end, i.e. index %d\n",size);
        op1 = size;
}

printf("Enter the value to insert: ");
scanf("%d", &op2);
```

```c
// Shift elements to the right
for(i = size; i > op1; i--)
        arr[i] = arr[i-1];

// Insert the new element
arr[op1] = op2;

// Update the size
size++;
```

Focus on this part, and convince yourself that it performs the insertion

Deletion in arrays

In the card sorting example, we also *deleted* a card every time (before we inserted it back again)

Deletion in arrays

In the card sorting example, we also *deleted* a card every time (before we inserted it back again)

Deletion involves

- Checking if the array has any elements; if there are, we can remove any one of them

Deletion in arrays

In the card sorting example, we also *deleted* a card every time (before we inserted it back again)

Deletion involves

- Checking if the array has any elements; if there are, we can remove any one of them
- Picking an index in the array, the value at which, is to be removed; let the index be `i`

Deletion in arrays

In the card sorting example, we also *deleted* a card every time (before we inserted it back again)

Deletion involves

- Checking if the array has any elements; if there are, we can remove any one of them
- Picking an index in the array, the value at which, is to be removed; let the index be `i`
- Shifting elements at index `i+1` and and all “valid” indices greater than `i`, to the left

Deletion in arrays

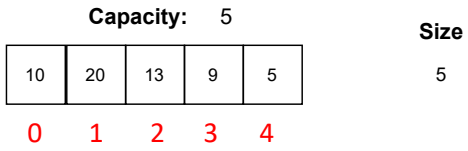
Capacity: 5

10	20	13	9	5

Size

5

Deletion in arrays

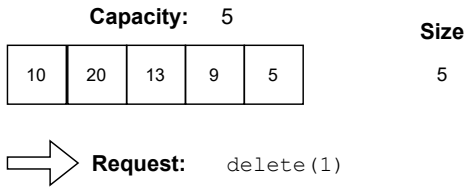


Size

5

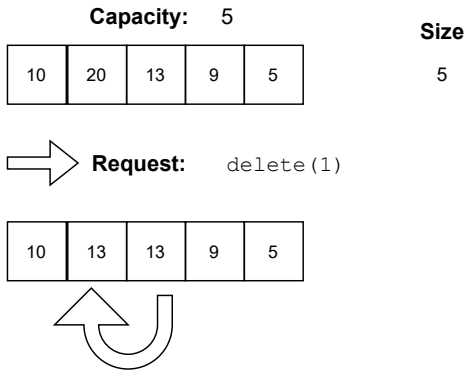
Let us go back to the array where we inserted the value

Deletion in arrays



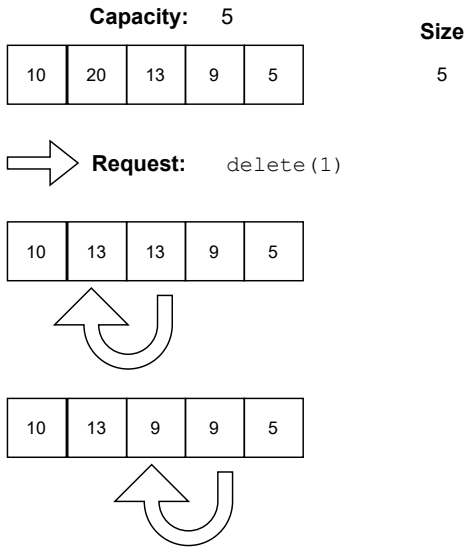
Assume that we now get a request to delete the value at index `1` of the array

Deletion in arrays



We copy the value at index `i+1` to `i`, i.e., index `2` to the previous index, i.e., index `1`

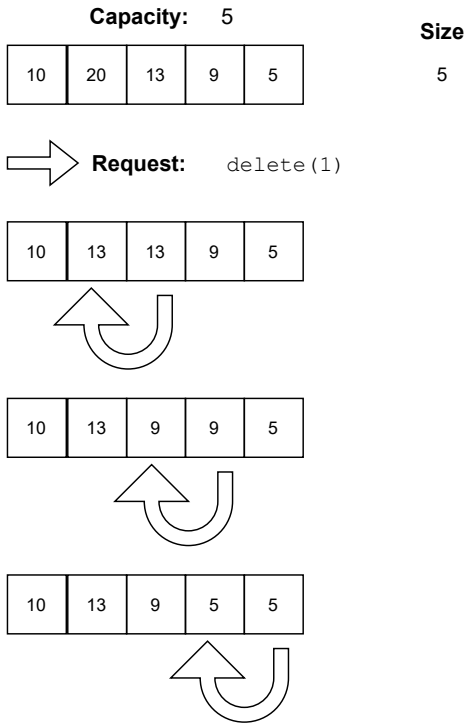
Deletion in arrays



We copy the value at index `i+1` to `i`, i.e., index `2` to the previous index, i.e., index `1`

… then we copy the value at index `3` to index `2`

Deletion in arrays

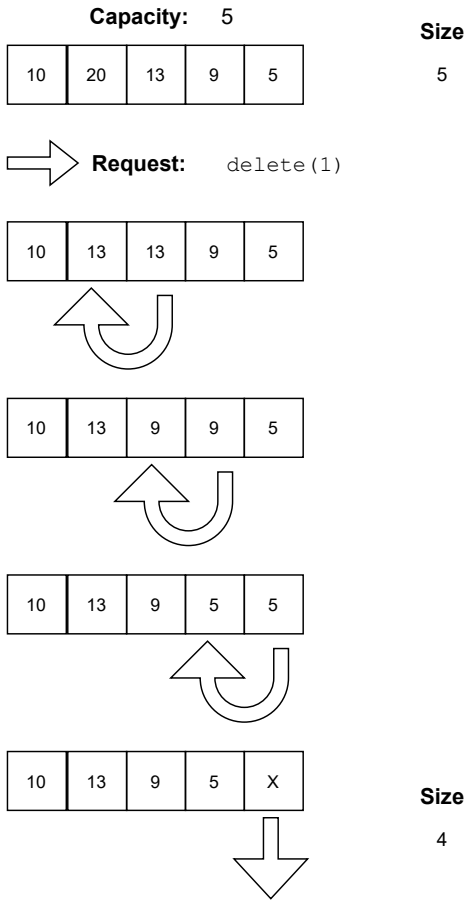


We copy the value at index `i+1` to `i`, i.e., index `2` to the previous index, i.e., index `1`

… then we copy the value at index `3` to index `2`

… and finally, we copy value at index `4` to index `3`

Deletion in arrays



We don’t really need to “delete” the value at last index…
we just reduce the size, and “ignore” the stored value

Deletion in arrays

```
// Check if array has any elements
if(size == 0)
{
        printf("The array is empty; add some elements first !\n");
        break;
}

printf("Enter the index of the value you wish to delete: ");
scanf("%d", &op1);


// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 >= size)
{
        printf("WARNING: The last element will be deleted, i.e. at index %d\n",size-1);
        op1 = size;
}

// Shift elements to the left
for(i = op1; i < size - 1; i++)
        arr[i] = arr[i+1];

// Update the size
size--;
```

Deletion in arrays

```
// Check if array has any elements
if(size == 0)
{
        printf("The array is empty; add some elements first !\n");
        break;
}

printf("Enter the index of the value you wish to delete: ");
scanf("%d", &op1);


// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 >= size)
{
        printf("WARNING: The last element will be deleted, i.e. at index %d\n",size-1);
        op1 = size;
}

// Shift elements to the left
for(i = op1; i < size - 1; i++)
        arr[i] = arr[i+1];

// Update the size
size--;
```

This is C code for deleting a value from an array

Deletion in arrays

```
// Check if array has any elements
if(size == 0)
{
        printf("The array is empty; add some elements first !\n");
        break;
}

printf("Enter the index of the value you wish to delete: ");
scanf("%d", &op1);


// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 >= size)
{
        printf("WARNING: The last element will be deleted, i.e. at index %d\n",size-1);
        op1 = size;
}

// Shift elements to the left
for(i = op1; i < size - 1; i++)
        arr[i] = arr[i+1];

// Update the size
size--;
```

`op1` contains the index for deletion

Deletion in arrays

```
// Check if array has any elements
if(size == 0)
{
        printf("The array is empty; add some elements first !\n");
        break;
}

printf("Enter the index of the value you wish to delete: ");
scanf("%d", &op1);


// Check if the index is valid
if(op1 < 0 || op1 >= CAPACITY)
{
        printf("Invalid insertion index\n");
        break;
}
else if(op1 >= size)
{
        printf("WARNING: The last element will be deleted, i.e. at index %d\n",size-1);
        op1 = size;
}
```

```
// Shift elements to the left
for(i = op1; i < size - 1; i++)
        arr[i] = arr[i+1];

// Update the size
size--;
```

Focus on this part, and convince yourself that it performs the deletion

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)
- Thus, the unsorted part contains (size – 1) elements

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)
- Thus, the unsorted part contains (size – 1) elements
- At each step, we pick the first element from the unsorted part…

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)
- Thus, the unsorted part contains (size – 1) elements
- At each step, we pick the first element from the unsorted part…
- … and “insert” it at its “correct” index in the sorted array

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)
- Thus, the unsorted part contains (size – 1) elements
- At each step, we pick the first element from the unsorted part…
- … and “insert” it at its “correct” index in the sorted array
- Then, we increase the size of the sorted part by 1 (thus, decreasing the size of unsorted part by 1)

The Insertion Sort

The sorting method we saw with the cards, using insertions and deletions…

- … is actually a variant of a sorting protocol !!

If we use insertions and deletions in a more systematic way, it is called the *Insertion Sort*

The idea behind Insertion Sort is summarised as below

- At each step, we divide the array into 2 “imaginary” parts
- One of the part is “sorted”, while the other is “unsorted”
- When we start, the sorted part contains just 1 element (a 1 element array is sorted by default)
- Thus, the unsorted part contains (size – 1) elements
- At each step, we pick the first element from the unsorted part…
- … and “insert” it at its “correct” index in the sorted array
- Then, we increase the size of the sorted part by 1 (thus, decreasing the size of unsorted part by 1)
- When the size of unsorted part becomes 0, the whole array has become sorted

The Insertion Sort

10	20	13	9	5

The Insertion Sort

Let us do insertion sort on our sample array !!

The Insertion Sort

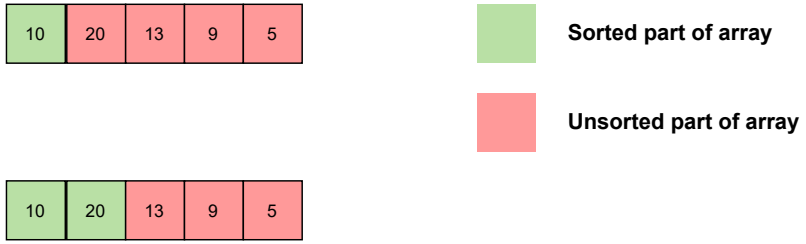
At the start, the sorted part only contains the index `0`, while the unsorted part contains indices `1` to `4`

The Insertion Sort

At the start, the sorted part only contains the index `0`, while the unsorted part contains indices `1` to `4`

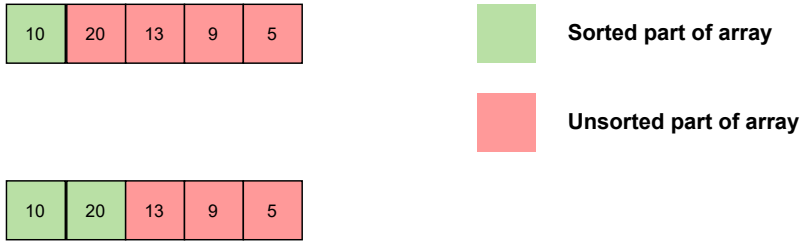
We delete the first element of the unsorted part, i.e., `20` at index `1`…

The Insertion Sort



… and insert it in the sorted part at its correct position, i.e., index 1 itself !!

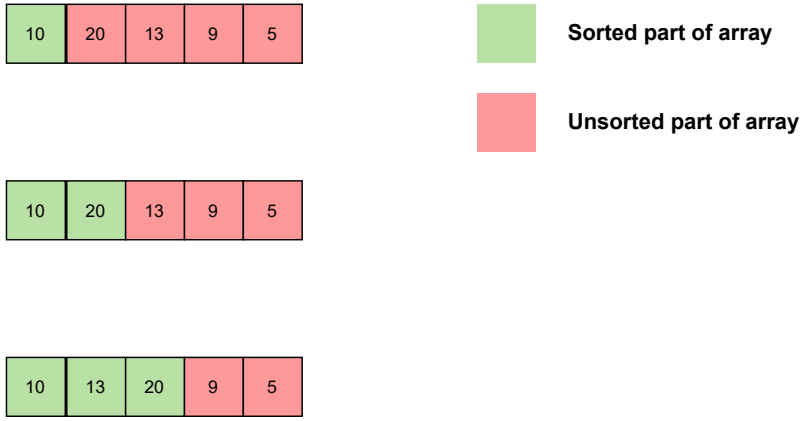
The Insertion Sort



… and insert it in the sorted part at its correct position, i.e., index 1 itself !!

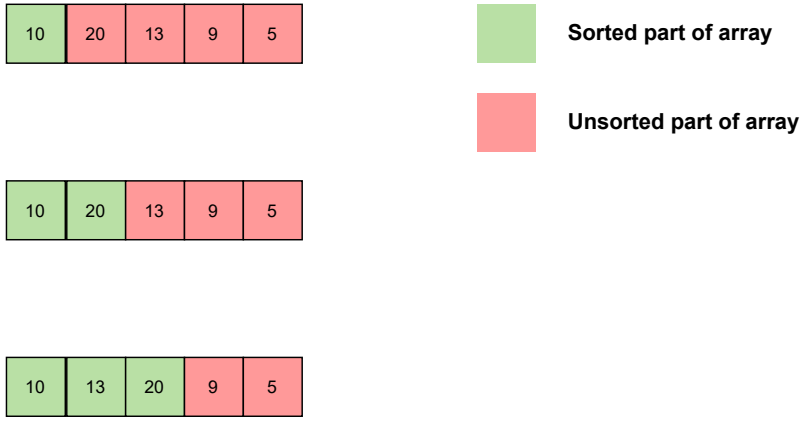
Next, we delete 13 from index 2…

The Insertion Sort



… and insert it in the sorted part at its correct position, i.e., index 1 !!

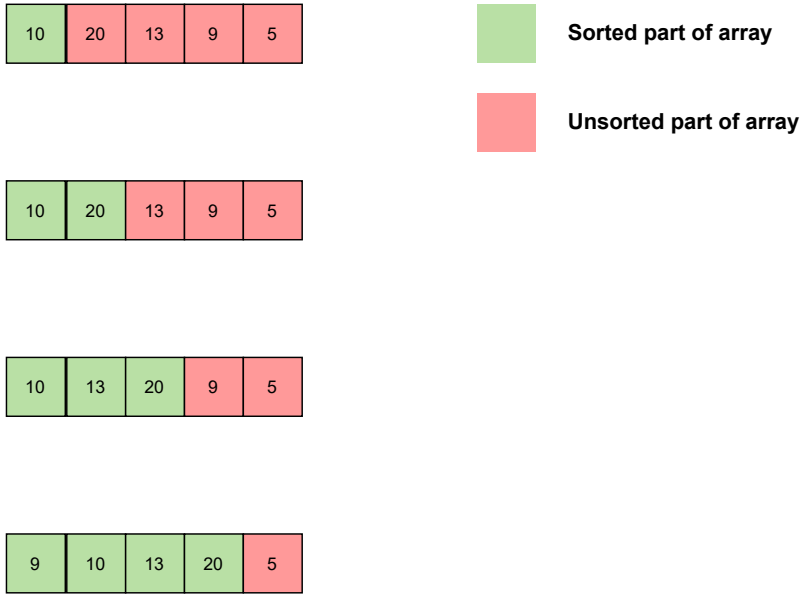
The Insertion Sort



… and insert it in the sorted part at its correct position, i.e., index 1 !!

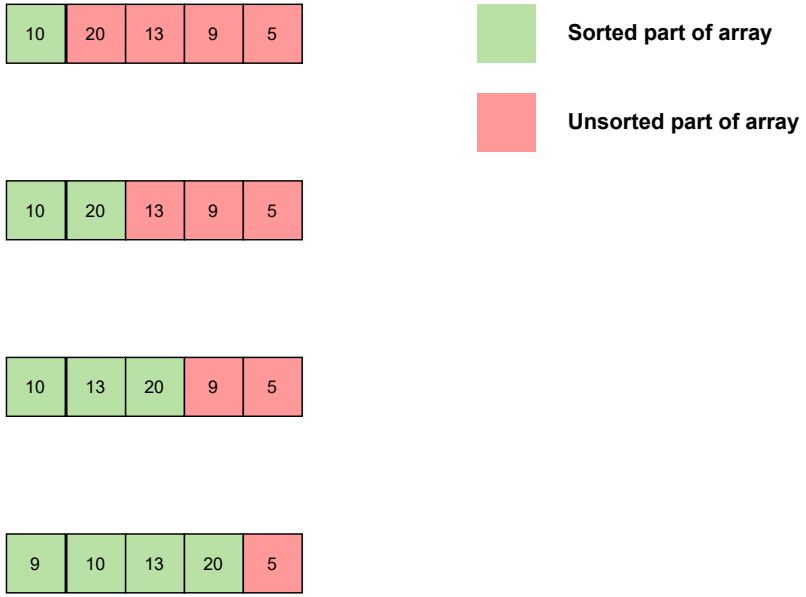
The first element of the unsorted part, i.e., 9 at index 3 is now deleted…

The Insertion Sort



… which gets inserted at index 0 in the sorted part of the array !!

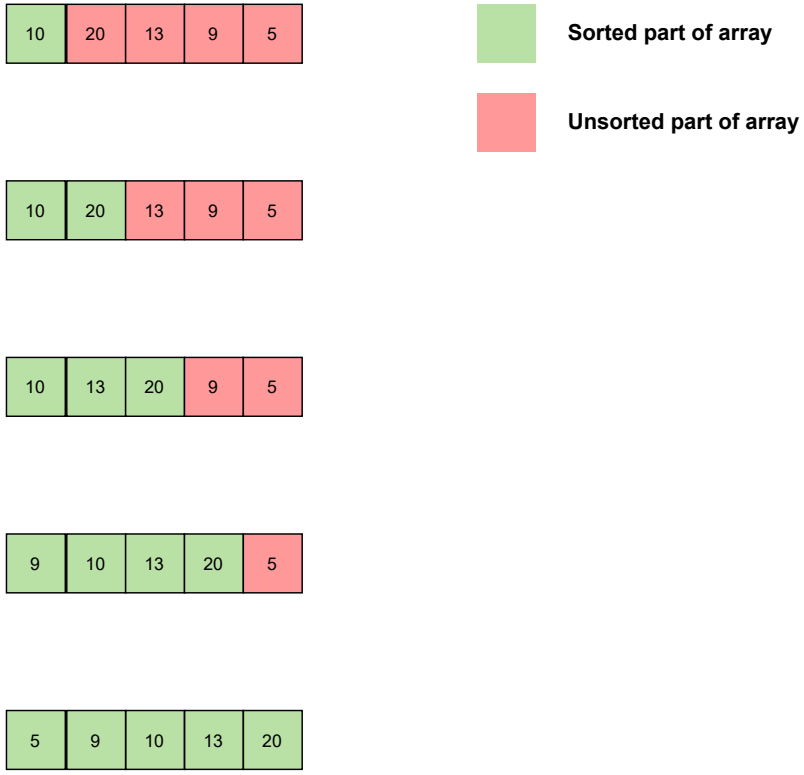
The Insertion Sort



… which gets inserted at index 0 in the sorted part of the array !!

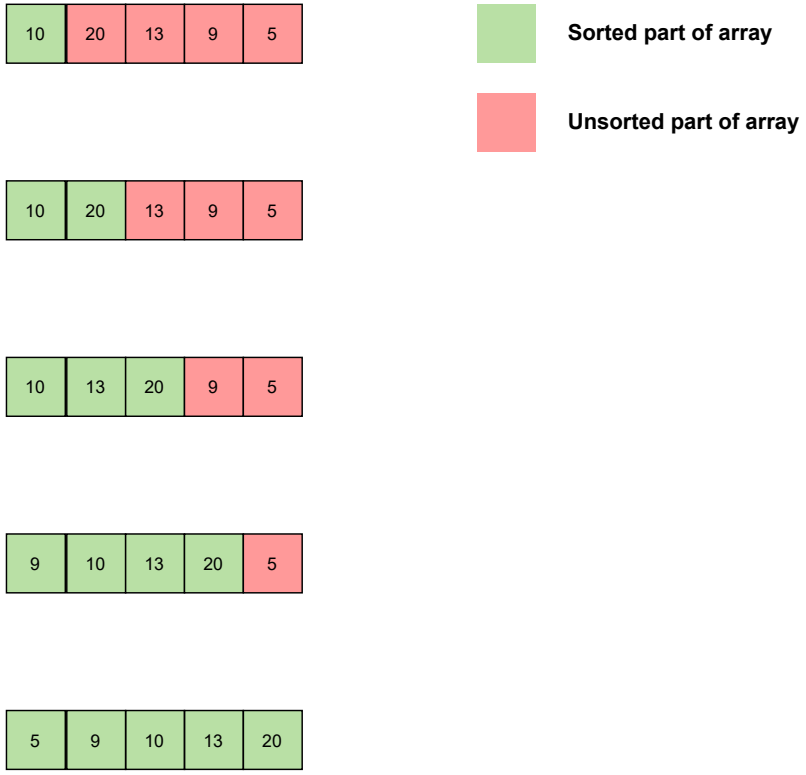
The last element of the unsorted part, i.e., 5 at index 4 is the last value to de deleted…

The Insertion Sort



… which is inserted at the index `0`, its correct position in the sorted part

The Insertion Sort



… which is inserted at the index `0`, its correct position in the sorted part

With this, the sorted part now contains the whole array… and we can feel proud that we have sorted the array :D

Searching in an array

How can you search for an element in an unsorted array?

- We don’t have much choice, but to check values at all the array indices, one by one

Searching in an array

How can you search for an element in an unsorted array?

- We don’t have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

Searching in an array

How can you search for an element in an unsorted array?

- We don’t have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array

Searching in an array

How can you search for an element in an unsorted array?

- We don’t have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you’ve hit the jackpot !!

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!
- So based on whether the element you are searching is smaller or larger than the value at index `i`…

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!
- So based on whether the element you are searching is smaller or larger than the value at index `i`…
- … you only need to search in either the left or the right part of the array, while the other can be ignored

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!
- So based on whether the element you are searching is smaller or larger than the value at index `i`…
- … you only need to search in either the left or the right part of the array, while the other can be ignored
- If you do this repeatedly, you'll eventually end up with a part of the array, that has a single element !!

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!
- So based on whether the element you are searching is smaller or larger than the value at index `i`…
- … you only need to search in either the left or the right part of the array, while the other can be ignored
- If you do this repeatedly, you'll eventually end up with a part of the array, that has a single element !!
- At this point, if the element is present at this index, you've found it, or, the element is simply absent…

Searching in an array

How can you search for an element in an unsorted array?

- We don't have much choice, but to check values at all the array indices, one by one

But can we do any better if we know that the array is sorted (let us say, in ascending order)?

- Assume that you check some random index, `i` of the array
- If the element at index `i`, is the element you were looking for, you've hit the jackpot !!
- But even if you are not lucky, you have still, essentially, divided the array in two parts…
- The part on the left of `i`, is sure to only have values smaller than the value at index `i`…
- … and the part on the right of `i`, is sure to only have values larger than the value at index `i` !!
- So based on whether the element you are searching is smaller or larger than the value at index `i`…
- … you only need to search in either the left or the right part of the array, while the other can be ignored
- If you do this repeatedly, you'll eventually end up with a part of the array, that has a single element !!
- At this point, if the element is present at this index, you've found it, or, the element is simply absent…
- **If you always pick the mid-point of the array's part you are searching as `i`, it is called *Binary Search***

Searching in an array

5	9	10	13	20

Searching in an array

Going back to our array

Searching in an array

To search: 9

5	9	10	13	20

Assume that we have to search 9 in this sorted array

Searching in an array

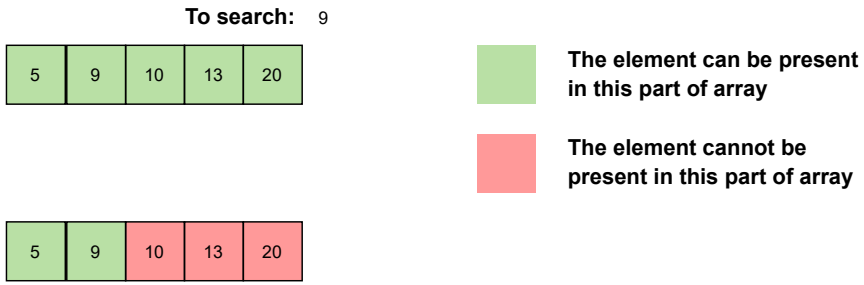
In the beginning, we assume that 9 can be present anywhere in the array

Searching in an array

In the beginning, we assume that 9 can be present anywhere in the array

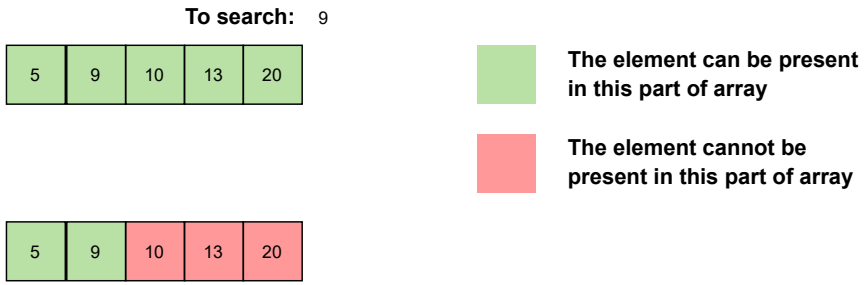
We compare it with the value at index 2, the mid-point of the part of the array where we must search (i.e., the whole array)

Searching in an array



Since 9 is smaller than 10, we can be sure that it cannot be present at indices 2 or higher, so only the left part of the array remains a candidate for further searching

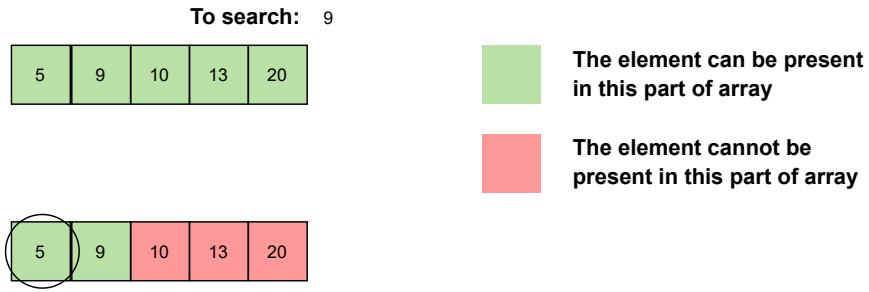
Searching in an array



Since 9 is smaller than 10, we can be sure that it cannot be present at indices 2 or higher, so only the left part of the array remains a candidate for further searching

Since it is a two-element part, there is really no “mid-point” as such… so we can pick either as the next point to search…

Searching in an array

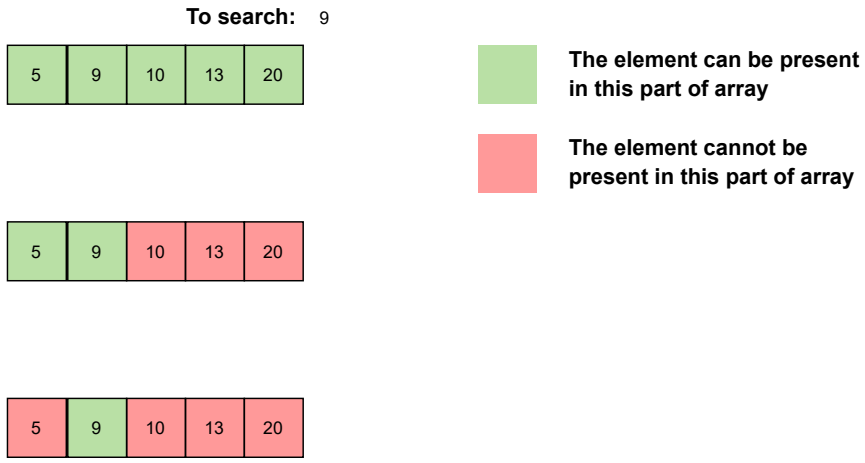


Since 9 is smaller than 10, we can be sure that it cannot be present at indices 2 or higher, so only the left part of the array remains a candidate for further searching

Since it is a two-element part, there is really no “mid-point” as such… so we can pick either as the next point to search…

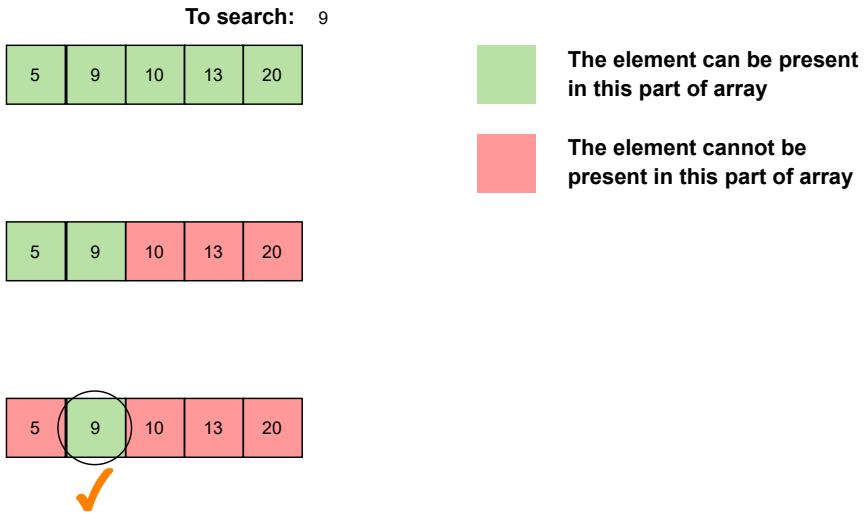
Let us pick index 0 for the next search

Searching in an array



Since 9 is larger than 5, it can only be present to the right of 5 in the array, leaving only index 1 to be searched

Searching in an array



… and this is where our element is !!

Homework !!

Go through the Example C file called `BasicArrayOperations.c`

- Create a new C file called `ArrayOperations.c`, containing two more operations…
- … “insertion sort array” operation, with operation code 3
- … and “binary search in array” operation, with operation code 4
- (assume that the user is “disciplined” enough to sort the array first, and then use binary search)

There are many other searching algorithms, you can easily comprehend at least two of them now

- Bubble Sort
- Selection Sort
- Try to write C code for them as well on your own