Sorting and Searching II

DM2233 Advanced Data Structures & Algorithms

Module Schedule

Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Standard Template Library / Assignment 1	Vesak Day (Mon)
Week 8 and 9: Mid-Sem Break		
10	Sorting and Searching I	
11	Sorting and Searching II	
12	Sorting and Searching III	
13	Binary Tree I	Hari Raya Puasa (Fri)
14	Lab Test	
15	Binary Tree II	
16	Binary Tree III	SG50 Day (Fri)
17	Preprocessing / Assignment 2	National Day (Mon)

Introduction to Sorting

- As you may know, there are 2 kinds of data:
  - Unordered data
  - Ordered data

- Unordered data does not follow any particular order (pattern, sequence, etc)

- Ordered data follow a particular order (e.g. ascending order, descending order, alphabetical order, etc)

Introduction to Sorting

- To have ordered data, it happens at 2 points:
  - At the point of data insertion/deletion
  - After random insertion, sorting is performed
- There are pros and cons for both ways
- As programmers, we have to identify the need for algorithms in the most effective way

Sorting Algorithms

- If there is a need to sort a group of random data, sorting algorithm need to be applied

- In our context, we would study a variety of sorting algorithms (not necessary the fastest one only) and perform analysis

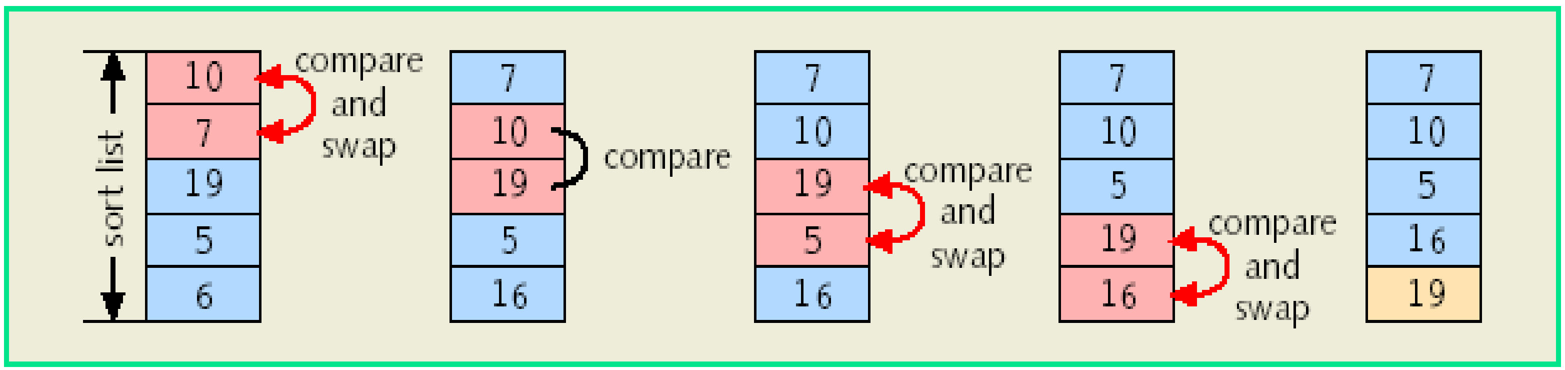
Sorting Algorithms

- There are many sorting algorithms in computer science. Below are a few sorting algorithms:
  - Bubble Sort
  - Selection
  - Insertion Sort
  - Quick Sort
  - Merge Sort

- Conceptually, sorting algorithms apply similarly to both array lists as well as linked list. However, examples are array-based for simplicity

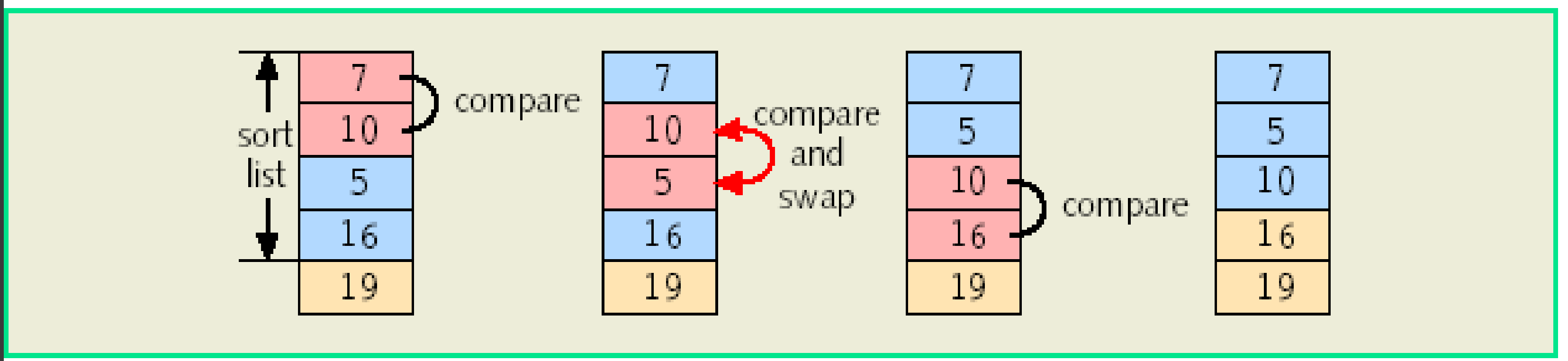
Bubble Sort Algorithm

- As the name goes, data are being “bubbled” up/down along the array until they are eventually sorted

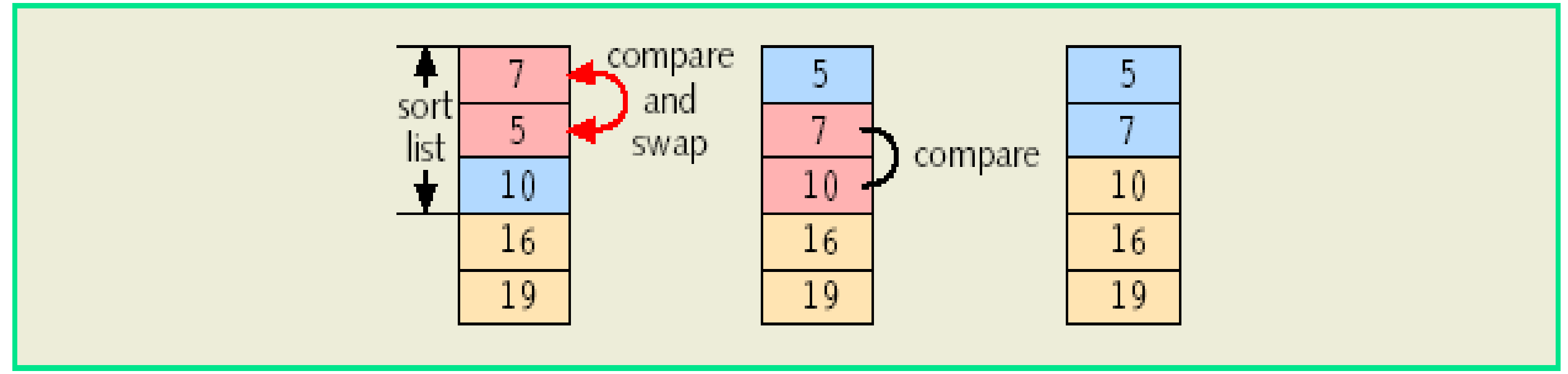


Elements of `list` during the first iteration

Bubble Sort Algorithm

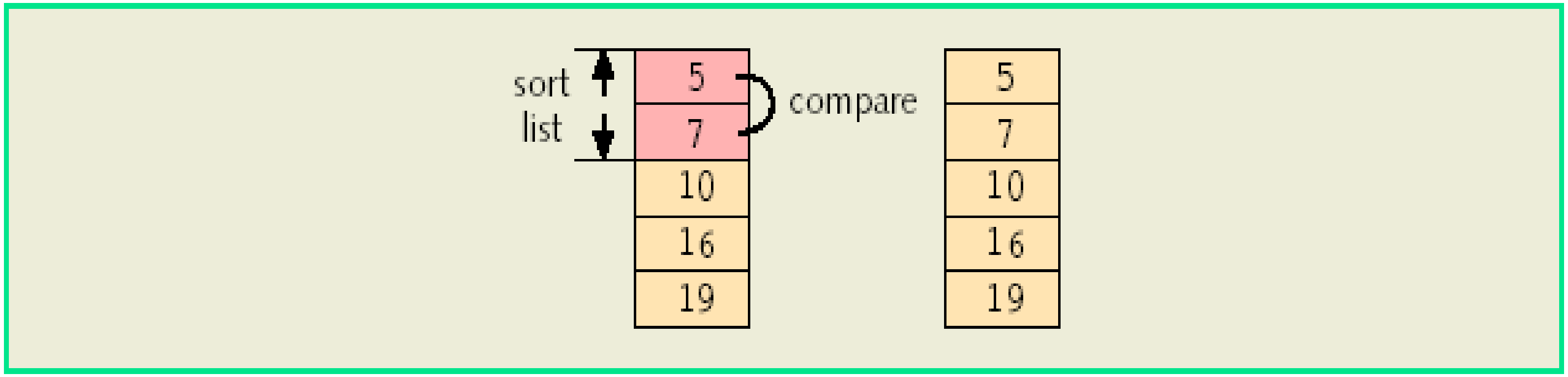


Elements of `list` during the second iteration



Elements of `list` during the third iteration

Bubble Sort Algorithm



Elements of `list` during the fourth iteration

Observation

- For 5 elements, 4 iterations is needed
- For each iteration, almost all the elements are accessed and compared
- As the iteration goes, for each iteration, the no. of elements accessed gets lesser
    - Reason being the list gets sorted from the end, there is not need for comparisons

Bubble Sort Algorithm

```
void bubbleSort(int list[], int length)
{
   for(int iter = 1; iter < length; iter++)
   {
      for(int index = 0; index < length - iter; index++)
      {
         if(list[index] > list[index+1])
         {
               //swap around
               int temp = list[index];
               list[index] = list[index+1];
               list[index+1] = temp;
         }
      }
   }
}
```

Selection Sort Algorithm

- It is about always getting the smallest item in the unsorted list portion, and swapping it with the proper location

- 2 major parts:
  - Find the smallest item in the unsorted list
  - Swap it with the proper location

Selection Sort Algorithm

- Cyan highlight denotes unsorted list portion

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

loc 0

- Swap it with array element indexed by loc

0	1	2	3	4	5	6	7
3	7	15	8	12	30	**13**	20

loc 0

Selection Sort Algorithm

- Find the smallest value in the unsorted list

0	1	2	3	4	5	6	7
3	7	15	8	12	30	13	20

loc	1

- Swap it with array element indexed by loc

0	1	2	3	4	5	6	7
3	7	15	8	12	30	13	20

loc	1

Selection Sort Algorithm

- Find the smallest value in the unsorted list

0	1	2	3	4	5	6	7
3	7	15	8	12	30	13	20

loc	2

- Swap it with array element indexed by loc

0	1	2	3	4	5	6	7
3	7	8	**15**	12	30	13	20

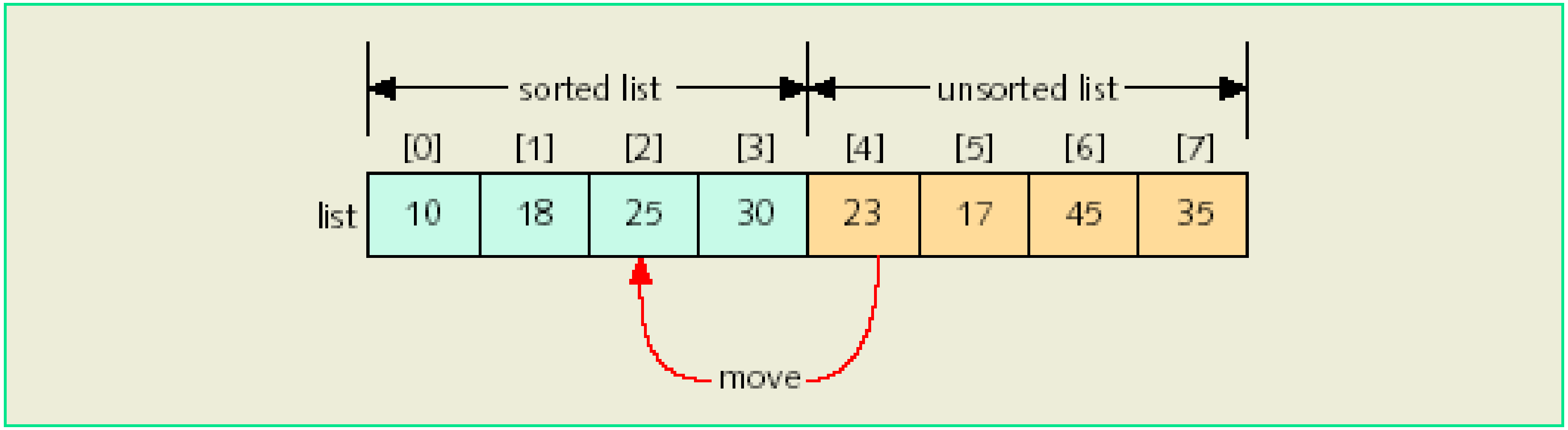
loc	2

Selection Sort Algorithm

```
void selectionSort(int list[], int length)
{
    int minIndex; //For getting the array index of the smallest
    value

    for(int loc = 0; loc < length; loc++)
    {
        minIndex = minLocation(list, loc, length);
        swap(list, loc, minIndex);
    }
}

//to find the index of the smallest value in the array given index
    range
//between loc and last
int minLocation(int list[], int loc, int last);

//to swap values in the array indexed by variables first and second
void swap(int list[], int first, int second);
```

Insertion Sort Algorithm

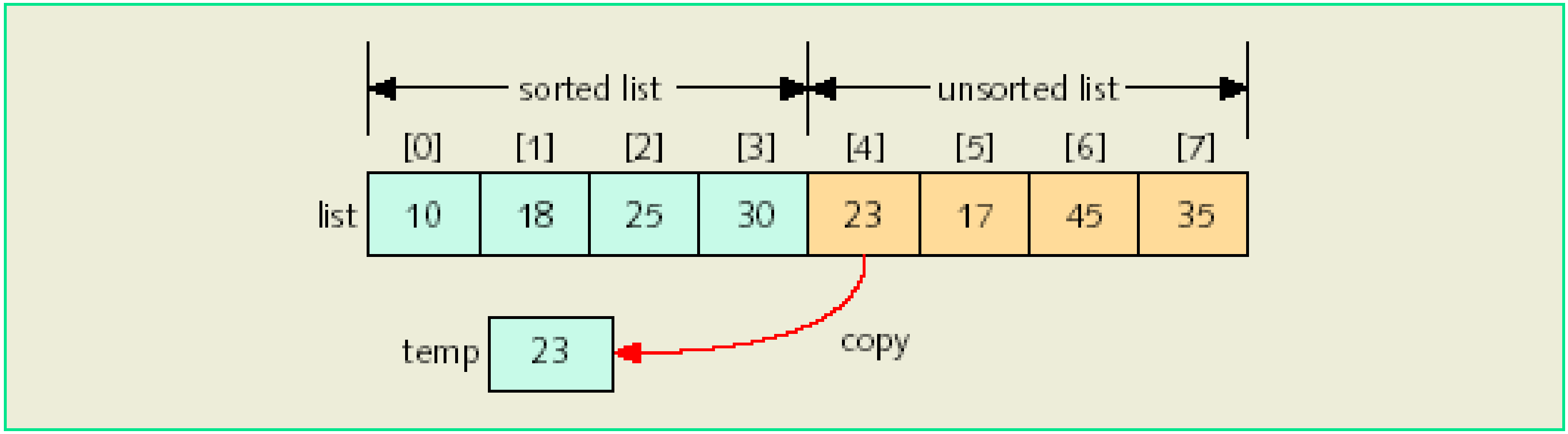
- It is about taking the first unsorted element from the unsorted portion of the array, and place correctly in the sorted portion of the array



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list	10	18	25	30	23	17	45	35

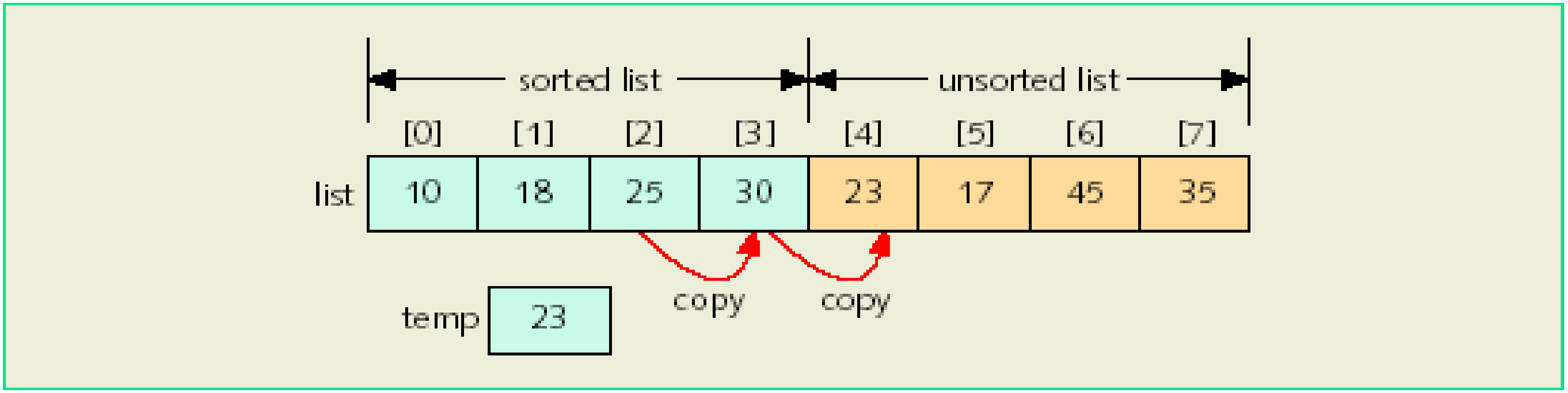
Move list[4] into list[2]

Insertion Sort Algorithm



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list	10	18	25	30	23	17	45	35

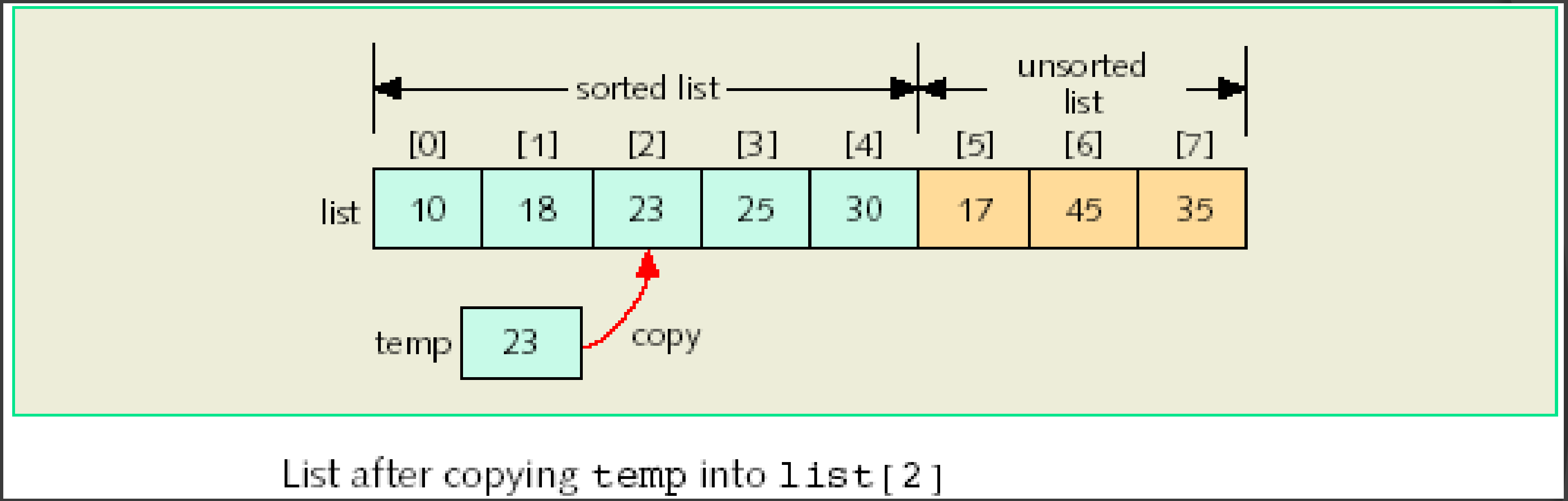
Copy list[4] into temp



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list	10	18	25	30	23	17	45	35

List before copying list[3] into list[4] and then list[2] into list[3]

Insertion Sort Algorithm



List after copying temp into list[2]

Observation

- At the start, insertion sorting algorithm always assume first element is sorted (being one and only element, it is obviously so)

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

firstOutofOrder 1

Insertion Sort Algorithm

- Cyan highlight denotes unsorted list portion

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

firstOutofOrder	1

- Take the first element in the unsorted list and compare it with previous adjacent element
  - If it is smaller, copy into temp and move front elements until proper location is found
  - If it is bigger, do nothing and go to next element

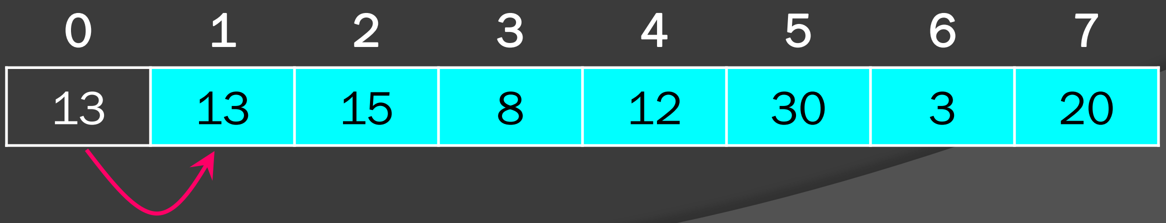
Insertion Sort Algorithm

- Cyan highlight denotes unsorted list portion

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

firstOutofOrder	1	temp	7

- Move front element(s)



0	1	2	3	4	5	6	7
13	13	15	8	12	30	3	20

firstOutofOrder	1	temp	7

Insertion Sort Algorithm

- Once the proper location is found, temp is copied in

0	1	2	3	4	5	6	7
7	13	15	8	12	30	3	20

firstOutofOrder 1 temp 7

- Get next element in unsorted list

0	1	2	3	4	5	6	7
7	13	15	8	12	30	3	20

firstOutofOrder 2 temp 7

Insertion Sort Algorithm

- Get next element in the unsorted list

0	1	2	3	4	5	6	7
7	13	15	8	12	30	3	20

firstOutofOrder	3	temp	7

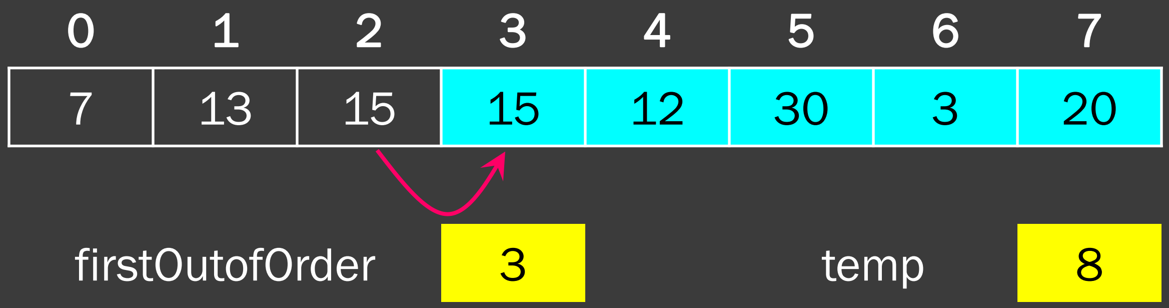
- Copy into temp

0	1	2	3	4	5	6	7
7	13	15	8	12	30	3	20

firstOutofOrder	3	temp	8

Insertion Sort Algorithm

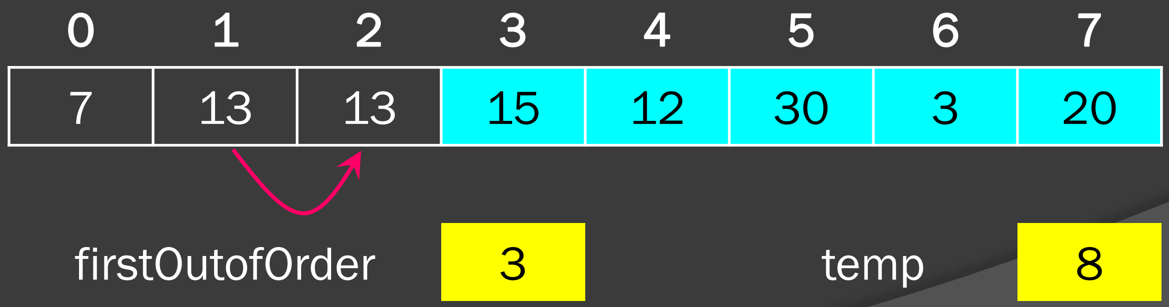
- Move front element(s)



0	1	2	3	4	5	6	7
7	13	15	15	12	30	3	20

firstOutofOrder 3

temp 8



0	1	2	3	4	5	6	7
7	13	13	15	12	30	3	20

firstOutofOrder 3

temp 8

Insertion Sort Algorithm

- Copy temp into the proper location

0	1	2	3	4	5	6	7
7	8	13	15	12	30	3	20

firstOutofOrder	3	temp	8

- Get next element

0	1	2	3	4	5	6	7
7	8	13	15	12	30	3	20

firstOutofOrder	4	temp	8

Insertion Sort Algorithm

```
void insertionSort(int list[], int length)
{
    int location, temp;

    for(int firstOutofOrder = 1; firstOutofOrder < length; firstOutofOrder++)
    {
        //check against previous adjacent element
        if(list[firstOutofOrder] < list[firstOutofOrder - 1])
        {
            temp = list[firstOutofOrder];
            location = firstOutofOrder; //initialize where to start moving back

            do
            {
                list[location] = list[location - 1];
                location--;
            }
            while(location > 0 && list[location - 1] > temp);
            list[location] = temp;
        }
    }
}
```

Summary

- Understand and implement
  - Bubble Sort
  - Selection Sort and
  - Insertion Sort Algorithms