Searching & Sorting

Searching

- Search the first n elements of x for a target value & return its location if found, else -1

- `public static int search(int n, int[] x, int target)`

Searching

- Search the first n elements of x for a target value & return its location if found, else -1

```
public static int search(int n, int[] x, int target) {
    int pos = 0;
    while ( pos < n )
       pos++;
    if ( x[pos] == target)
       return pos;           // found at pos


}
```

Sequential search O(n)

Searching

- Search the first n elements of x for a target value & return its location if found, else -1
- Complexity?
- More efficient solution?

Binary Search

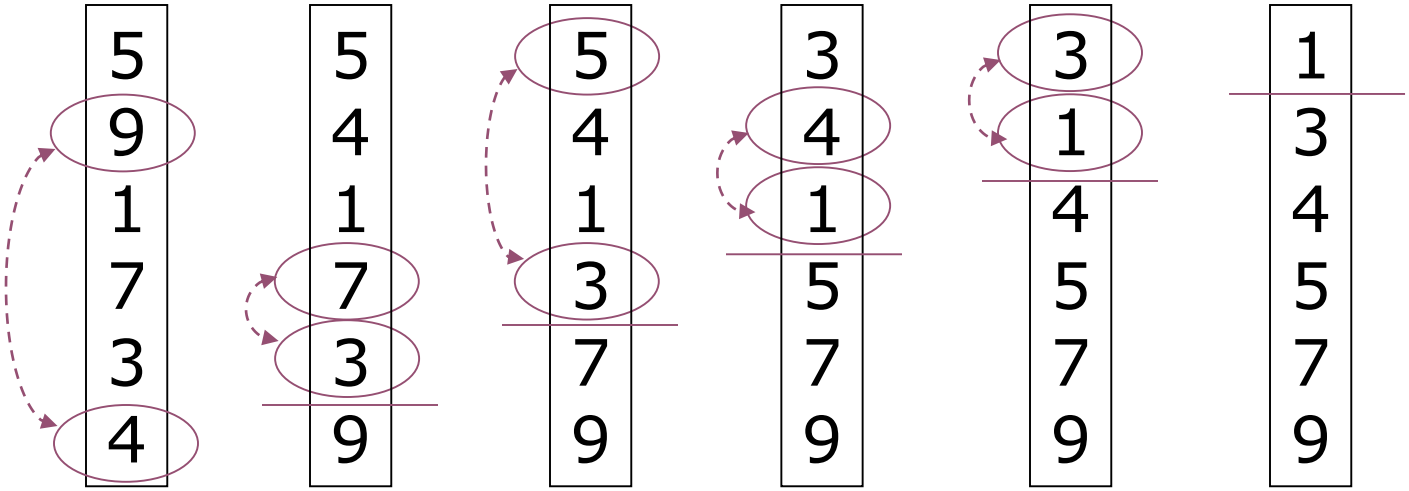
Binary Search

```
public static void binarySearch(int[ ] array, int lowerbound, int upperbound, int key)
{
   int position;

  // To start, find the index of the middle position.
  position = ( lowerbound + upperbound) / 2;

  while((array[position] != key) && (lowerbound <= upperbound))
  {
     if (array[position] > key)          // If the number is > key, decrease position by one.
     {
        upperbound = position - 1;
     }
        else
    {
        lowerbound = position + 1;    // Else, increase position by one.
    }
    position = (lowerbound + upperbound) / 2;
  }
  if (lowerbound <= upperbound)
  {
      System.out.println("The number was found in array index" + position);

  }
  else
      System.out.println("Sorry, the number is not in this array. ");
 }
}
```

Sorting

Sorting

- Selection sort



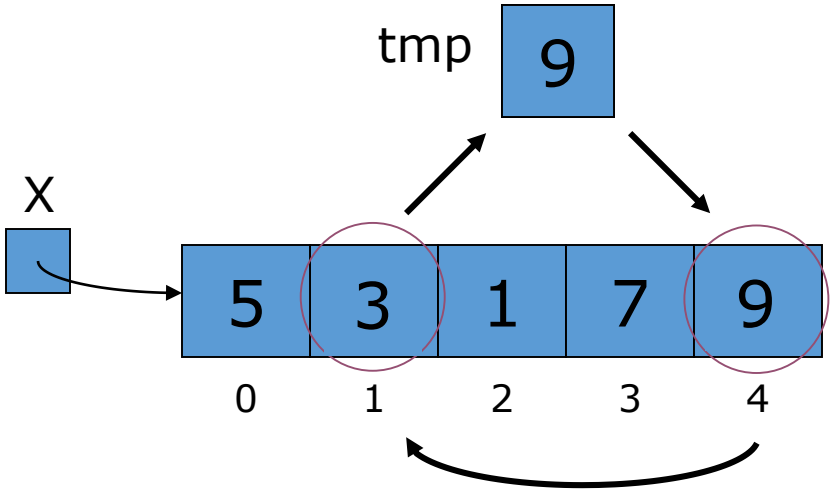
n=6 + n=5 + n=4 + n=3 + n=2 + n=1

Sum = n.(n +1) / 2 = $n^2/2 + n/2$ **O(n^2)**

Selection Sort

- To sort first n elements of array X

```
while n > 1
    find location of max value in first n elements of X
    swap element at location of max with n'th element
    decrement n
```




```
swap( int i, int j) {
    int tmp;
    tmp = X[i];
    X[i} = X[j]:
    X[j] = tmp;
}
```

Selection Sort (alternative)

```
public class SelectionSort {

   private static void swap(int[] a, int i, int j) {
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   public static int[] selectionSort(int[] list) {
      for (int i = 0; i < list.length - 1; i++) {
         // Find the index of the minimum value
         int minPos = i;
         for (int j = i + 1; j < list.length; j++) {
            if (list[j] < list[minPos]) {
               minPos = j;
            }
         }
         swap(list, minPos, i);
      }
      return list;
   }
```

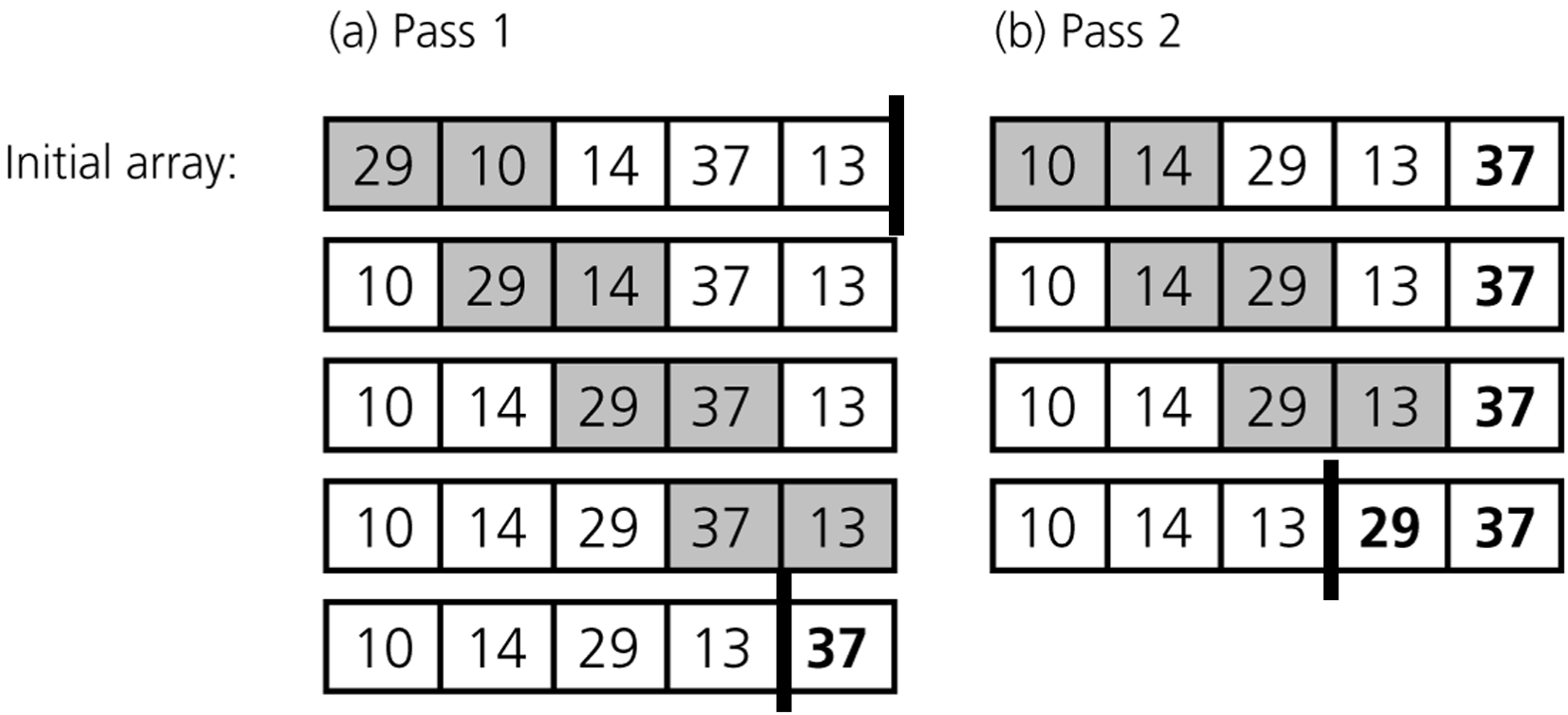
Sorting

- Complexity?
- What if the array is already sorted?
- More efficient techniques?

Bubble Sort

- Array divided into two sublists: *sorted* and *unsorted*.
- The largest element is bubbled from the unsorted part and moved to the sorted part.
- After that, the wall moves one element back, increasing the number of sorted elements and decreasing the number of unsorted ones.
- One sort pass: each time an element moves from the unsorted part to the sorted part.
- Given a list of *n* elements, bubble sort requires up to n-1 passes (maximum passes) to sort data.

Bubble Sort (cont.)



Bubble Sort

```
public static void BubbleSort( int [ ] num ) {
    boolean flag = true;   // set flag to true to begin first pass
    int temp;   //holding variable

    while ( flag ) {
        flag= false;    //set flag to false awaiting a possible swap
        for( j=0;  j < num.length -1;  j++ ) {
            if ( num[ j ] > num[j+1] ) {   // change to < for descending sort
                temp = num[ j ];               //swap elements
                num[ j ] = num[ j+1 ];
                num[ j+1 ] = temp;
                flag = true;              //shows a swap occurred
            }
        }
    }
}
```