Sorting

- Important operation when organizing data
  - Ordering of elements
  - Finding duplicate elements
  - Ranking elements (i.e., $n^{th}$ largest)
  - etc.
- An example
  
  \[
  \text{input: } 3 \ 6 \ 1 \ 9 \ 4 \ 2 \quad \text{output: } 1 \ 2 \ 3 \ 4 \ 6 \ 9
  \]
- There are many different strategies and we need to care about
  - Correctness
  - Efficiency (in terms of time complexity)

Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Merge sort
- Quick sort

Bubble Sort

- Comparison of neighbors
  - Swap if not in order
  - Iterate left to right
- After iteration
  - The largest element is located at the rightmost spot.
- Repeat loop
  - No need to revisit the largest element
  - Loop keeps decreasing in length.
**Bubble Sort**

1st iteration: 
3 6 1 9 4 2
2nd iteration: 
3 6 1 9 4 2
3rd iteration: 
3 6 1 9 4 2
4th iteration: 
3 1 6 9 4 2
5th iteration: 
3 1 6 9 4 2
Final result: 
1 2 3 4 6 9

**Bubble Sort Algorithm**

```java
public void bubble_sort(int[] values) {
    if (values == null || values.length == 0) return;
    for (int i=0; i<values.length-1; i++)
        for (int j=0; j<values.length-i-1; j++)
            if (values[j] > values[j+1])
                swap(values, j, j+1);
}

public void swap(int[] arr, int i, int j) {
    int t = arr[i];
    arr[i] = arr[j];
    arr[j] = t;
}
```

**Analysis of Bubble Sort**

- Number of comparisons and swaps in the worst case:
  \[(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} \sim O(n^2)\]

- Time complexity:
  - Best case: \(O(n)\)
  - Worst case: \(O(n^2)\)
  - Average case: \(O(n^2)\)

**Selection Sort**

- Problem with bubble sort: too many swaps
  - Number of comparisons: \(O(n^2)\)
  - Number of swaps: \(O(n^2)\)
  - Costly if swap operation takes time

- Better approach: one swap per iteration
  - Identify element to swap through comparisons
  - Swap the smallest element with the one in the left-most position.
  - Iterative growth of “sorted area” in array
Selection Sort

1st iteration

1 6 3 9 4 2

2nd iteration

1 6 3 9 4 2

3rd iteration

1 2 3 9 4 6

4th iteration

1 2 3 9 4 6

5th iteration

1 2 3 4 9 6

Final result

1 2 3 4 6 9

public void selection_sort(int[] values)
{
    // Check for empty or null array
    if (values == null || values.length == 0)
        return;
    for (int i=0; i<values.length-1; i++)
    {
        int minIndex = i;
        for (int j=i+1; j<values.length; j++)
        {
            if (values[j] < values[minIndex])
                minIndex = j;
        }
        swap(values, i, minIndex);
    }
}

Analysis of Selection Sort

- Time complexity
  - Best case: $O(n^2)$ comparisons
  - Worst case: $O(n^2)$ comparisons
  - Average case: $O(n^2)$ comparisons
  - Much less swaps: $O(n)$

Insertion Sort

- Slightly better than selection sort
- Idea:
  - Insert next element into partially sorted array
  - Iterate
- Insertion requires shifting of elements.
Insertion Sort

1st iteration
3 6 1 9 4 2

2nd iteration
3 6 1 9 4 2

3rd iteration
1 3 6 9 4 2

4th iteration
1 3 6 9 4 2

5th iteration
1 2 3 4 6 9

Final result
1 2 3 4 6 9

Insertion Sort Algorithm

```java
public void insertion_sort(int[] values)
{
    // Check for empty or null array
    if (values == null || values.length == 0)
        return;
    for (int i=1; i<values.length; i++)
    {
        int temp = values[i];
        int j = i;
        while (j>0 && values[j-1]>temp)
        {
            values[j] = values[j-1];
            j--;
        }
        values[j]=temp;
    }
}
```

Analysis of Insertion Sort

- Time complexity
  - Best case: \(O(n)\) comparisons and swaps
  - Worst case: \(O(n^2)\) comparisons and swaps
  - Average case: \(O(n^2)\) comparisons and swaps

Comparisons of Sorting Algorithms

The three algorithms have the same time complexity but their empirical speeds are different.