### Sorting algorithms Cheat Sheet

**Sorting algorithms**
- Bubble sort
- Heapsort
- Insertion sort
- Introsort
- Merge sort
- Patience sorting
- Quick sort
- Selection sort
- Timsort
- Unshuffle sort

**Methods**
- Exchanging
- Selection
- Insertion
- Partitioning & Selection
- Merging
- Insertion & Selection
- Insertion & Merging
- Insertion
- Selection
- Partitioning
- Distribution and Merge

### Best and Worst Case

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Best Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bogosort</td>
<td>n</td>
<td>∞</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>n log n</td>
<td>n²</td>
</tr>
<tr>
<td>Bucket sort (uniform keys)</td>
<td>n log n</td>
<td>n²k</td>
</tr>
<tr>
<td>Heap sort</td>
<td>n log n</td>
<td>n log n</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>n log n</td>
<td>n log n</td>
</tr>
<tr>
<td>Merge sort</td>
<td>n log n</td>
<td>n log n</td>
</tr>
<tr>
<td>Quick sort</td>
<td>n log n</td>
<td>n²</td>
</tr>
<tr>
<td>Selection sort</td>
<td>n²</td>
<td>n²</td>
</tr>
<tr>
<td>Shell sort</td>
<td>n log n</td>
<td>n⁴/³</td>
</tr>
<tr>
<td>Spreadsort</td>
<td>n</td>
<td>n(k/s+d)</td>
</tr>
<tr>
<td>Timsort</td>
<td>n</td>
<td>n log n</td>
</tr>
<tr>
<td>Unshuffle sort</td>
<td>n</td>
<td>kn</td>
</tr>
</tbody>
</table>

### Insertion sort

```java
function insertionSortR(array A, int n)
    if n>0
        insertionSortR(A, n-1)
        x ← A[n]
        j ← n-1
        while j >= 0 and A[j] > x
            j ← j-1
        end while
        A[j+1] ← x
    end if
end function
```

### Merge sort

```java
function merge_sort(list m)
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m
    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    if i < (length of m)/2 then
        add x to left
    else
        add x to right
    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)
    // Then merge the now-sorted sublists.
    return merge(left, right)
end function
```

### Bogosort

```java
while not isInOrder(deck):
    shuffle(deck)
```

### Bucket sort

```java
function bucketSort(array, n) is
    buckets ← new array of n empty lists
    for i = 0 to n - 1 do
        buckets[(array[i] / n)] ← array[i] ;
    end for
    // Then merge the now-sorted sublists.
    return merge(buckets[0], ..., buckets[n-1])
end function
```

### Resources

- [http://bigcheatsheet.com](http://bigcheatsheet.com)
Algorithm complexities

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Average Case</th>
<th>Memory complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitonic sorter</td>
<td>(\log^2 n)</td>
<td>(n \log^2 n)</td>
</tr>
<tr>
<td>Bogosort</td>
<td>(n \times n!)</td>
<td>1</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>(n^2)</td>
<td>1</td>
</tr>
<tr>
<td>Bucket sort (uniform keys)</td>
<td>(n+k)</td>
<td>(nk)</td>
</tr>
<tr>
<td>Burstsort</td>
<td>(n(k/d))</td>
<td>(n(k/d))</td>
</tr>
<tr>
<td>Counting sort</td>
<td>(n+r)</td>
<td>(n+r)</td>
</tr>
<tr>
<td>Heap sort</td>
<td>(n \log n)</td>
<td>1</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>(n^2)</td>
<td>1</td>
</tr>
<tr>
<td>Introsort</td>
<td>(n \log n)</td>
<td>(\log n)</td>
</tr>
<tr>
<td>LSD Radix Sort</td>
<td>(n(k/d))</td>
<td>(n+2^d)</td>
</tr>
<tr>
<td>Merge sort</td>
<td>(n \log n)</td>
<td>(n)</td>
</tr>
<tr>
<td>MSD Radix Sort (in-place)</td>
<td>(n(k/d))</td>
<td>(2^d)</td>
</tr>
<tr>
<td>Patience sort</td>
<td>-</td>
<td>(n)</td>
</tr>
<tr>
<td>Pigeonhole sort</td>
<td>(n+2^k)</td>
<td>(2^k)</td>
</tr>
<tr>
<td>Quicksort</td>
<td>(n \log n)</td>
<td>(\log n)</td>
</tr>
<tr>
<td>Selection sort</td>
<td>(n^2)</td>
<td>1</td>
</tr>
<tr>
<td>Shell sort</td>
<td>Depends on gap sequence</td>
<td>1</td>
</tr>
<tr>
<td>Spaghetti sort</td>
<td>(n)</td>
<td>(n^2)</td>
</tr>
<tr>
<td>Stooget sort</td>
<td>(n(\log 3/\log 1.5))</td>
<td>(n)</td>
</tr>
<tr>
<td>Timsort</td>
<td>(n \log n)</td>
<td>(n)</td>
</tr>
</tbody>
</table>

Bubble sort

procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            if A[i-1] > A[i] then
                swap A[i-1], A[i]
                swapped = true
            end if
        end for
    until not swapped
end procedure

Quicksort

algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1)
        quicksort(A, p + 1, hi)
    end if
end procedure

Selection sort

procedure selection sort
    list : array of items
    n : size of list
    for i = 1 to n - 1
        set current element as minimum
        min = i
        check the element to be minimum
        for j = i+1 to n
            if list[j] < list[min] then
                min = j;
            end if
            end for
        if indexMin != i
            swap the minimum element with the current element
            swap list[min] and list[i]
        end if
    end for
end procedure