### Sorting algorithms Cheat Sheet

**Sorting algorithms and Methods**

<table>
<thead>
<tr>
<th>Sorting algorithms</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble sort</td>
<td>Exchanging</td>
</tr>
<tr>
<td>Heapsort</td>
<td>Selection</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>Insertion</td>
</tr>
<tr>
<td>Introsort</td>
<td>Partitioning &amp; Selection</td>
</tr>
<tr>
<td>Merge sort</td>
<td>Merging</td>
</tr>
<tr>
<td>Patience sorting</td>
<td>Insertion &amp; Selection</td>
</tr>
<tr>
<td>Quicksort</td>
<td>Partitioning</td>
</tr>
<tr>
<td>Selection sort</td>
<td>Selection</td>
</tr>
<tr>
<td>Timsort</td>
<td>Insertion &amp; Merging</td>
</tr>
<tr>
<td>Unshuffle sort</td>
<td>Distribution and Merge</td>
</tr>
</tbody>
</table>

**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>(\infty)</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>n</td>
<td>(n^2)</td>
</tr>
<tr>
<td>Bucket sort (uniform keys)</td>
<td>-</td>
<td>(n^{2k})</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</td>
<td>(n^2)</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>(n^2)</td>
</tr>
<tr>
<td>Selection sort</td>
<td>(n^2)</td>
<td>(n^2)</td>
</tr>
<tr>
<td>Shell sort</td>
<td>(n \log n)</td>
<td>(n^{4/3})</td>
</tr>
<tr>
<td>Smoothsort</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**

```plaintext
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**

```plaintext
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.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        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)
```

**Bogosort**

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

**Bucket sort**

```plaintext
function bucketSort(array, n) is
    buckets ← new array of n empty lists
    for i = 0 to (length(array)-1) do
        insert array[i] into buckets[msbits(array[i], k)]
    for i = 0 to n - 1 do
        nextSort(buckets[i]);
    return the concatenation of buckets[0], ......, buckets[n-1]
```

**Resources**

- [http://bigocheatsheet.com](http://bigocheatsheet.com)
### Sorting 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>$\log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>$n^2$</td>
<td>1</td>
</tr>
<tr>
<td>Introsort</td>
<td>$\log n$</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>$\log n$</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>$\log n$</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>Spreadsort</td>
<td>$n(k/d)$</td>
<td>$(k/d)2^d$</td>
</tr>
<tr>
<td>Stooge 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 (cont)

```plaintext
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
    n = n - 1
until not swapped

end procedure
```

### Quick sort

```plaintext
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

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi - 1 do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
            swap A[i] with A[hi]
        end if
    end for
    return i
```

### Selection sort

```plaintext
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
    / swap the minimum element with the current element/
    if indexMin != i then
        swap list[min] and list[i]
    end if
end for
```