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

function insertionSort(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)
end function

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</td>
<td>n^2</td>
</tr>
<tr>
<td>Bucket sort</td>
<td>n log n</td>
<td>n^2 k</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>Splitsort</td>
<td>n</td>
<td>n log n</td>
</tr>
<tr>
<td>Timsort</td>
<td>n</td>
<td>kn</td>
</tr>
<tr>
<td>Unshuffle sort</td>
<td>n</td>
<td>kn</td>
</tr>
</tbody>
</table>

Merge sort

Resources

http://bigcheatsheet.com

By pryl
cheatography.com/pryl/

Published 27th August, 2018.
Page 1 of 2.
### 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>log² n</td>
<td>n log² n</td>
</tr>
<tr>
<td>Bogosort</td>
<td>n × n!</td>
<td>1</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>n²</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>n²</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>n²</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>n²</td>
</tr>
<tr>
<td>Sortsort</td>
<td>n(log 3/log1.5)</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 )*

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

### Quick sort

*algorithm quicksort(A, lo, hi)* is

```plaintext
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

```plaintext
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

*procedure selection sort*

```plaintext
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
```

By pryl

cheatography.com/pryl/

Published 27th August, 2018.
Page 2 of 2.

Sponsored by CrosswordCheats.com

Learn to solve cryptic crosswords!

http://crosswordcheats.com