Cheatsheet - Comparison and Non-Comparison Sorting Algorithms

Fabio Lama – fabio.lama@pm.me

1. About

This cheatsheet provides an overview of some common sorting algorithms.

2. Comparison Sort Overview

Name	Worst case complexity	Best case complexity
Bubble	$\Theta(N^2)$	$\Theta(N)$
Insertion	$\Theta(N^2)$	$\Theta(N)$
Selection	$\Theta(N^2)$	$\Theta(N^2)$
Quicksort	$\Theta(N^2)$	$\Theta(N \times \log N)$
Mergesort	$\Theta(N \times \log N)$	$\Theta(N \times \log N)$
Radix Sort	$\Theta(d \times N)$	$\Theta(d \times N)$
Bucket Sort	$\Theta(N^2)$	$\Theta\left(N + \frac{N}{b} + b\right)$

Note that the d in Radix Sort is the number of digits and b in Bucket Sort is the number of buckets. Because comparison sorts must compare pairs of elements, **they cannot** run faster than $N \times \log N$.

3. Note: Sorting Algorithm Visualizer

It's recommended to lookup a sorting algorithm visualizer online, such as: https://www.toptal.com/developers/sorting-algorithms

4. Bubble Sort

```
function BubbleSort(A, N)
  swapped = true
  while (swapped) do
    swapped = false
    for 0 ≤ i < N − 1 do
      if (A[i] > A[i + 1]) then
        swap(A[i], A[i + 1])
        swapped = true
      end if
    end for
    N = N − 1
  end while
  return A
end function
```

4.1. Time Complexity

The **best case** for bubble sort is:

$$T(N) = C_0 \times N + C_1$$

Additionally:

- $T(N)$ is $O(N)$, $O(N^2)$ and $O(N^3)$, etc.
- $T(N)$ is $\Omega(N)$, $\Omega(\log N)$ and $\Omega(1)$, etc.
- $T(N)$ is $\Theta(N)$

The **worst case** for bubble sort is:

$$T(N) = C_0 \times N^2 + C_1 \times N + C_2.$$

Additionally:

- $T(N)$ is $O(N^2)$ and $O(N^3)$, etc.
- $T(N)$ is $\Omega(N^2)$, $\Omega(\log N)$ and $\Omega(1)$, etc.
- $T(N)$ is $\Theta(N^2)$

5. Insertion Sort

```
function InsertionSort(A, N)
  for 1 ≤ j ≤ N − 1 do
    ins = A[j]
    i = j − 1
    while (i ≥ 0 and ins < A[i]) do
      A[i + 1] = A[i]
      i = i − 1
    end while
    A[i + 1] = ins
  end for
end function
```

6. Selection Sort

```
function SelectionSort(A, N)
  for 0 ≤ i < N − 1 do
    min = pos_min(A, i, N − 1)
    swap(A[i], A[min])
  end for
end function
```

The function $\text{pos\_min}(A, a, b)$ returns the position of the minimum value between positions a and b (both inclusive) in array A.

7. Quicksort

```
function Quicksort(A, low, high)
  if low < high then
    p = partition(A, low, high)
    Quicksort(A, low, p-1)
    Quicksort(A, p+1, high)
  end if
end function
function partition(A, low, high)
  pivot = A[high]
  i = low − 1
  for j = low to high − 1 do
    if A[j] < pivot then
    i = i + 1
    swap(A[i], A[j])
  end for
  swap(A[i + 1], A[high])
  return i+1
end function
```

7.1. Explanation

The function **partition**$(A, \texttt{low}, \texttt{high})$ selects a pivot element (usually the last element in the current segment of the array). It then rearranges the elements in the array such that all elements less than the pivot are moved to the left of the pivot and all elements greater than or equal to the pivot are moved to the right. The pivot is then placed in its correct position, and the index of the pivot is returned.

8. Mergesort

```
function MergeSort(A, low, high)
    if (low < high)
        mid = low + floor((high − 1) ÷ 2)
        MergeSort(A, low, mid)
        MergeSort(A, mid + 1ow, high)
        Merge(A, low, mid, high)
    end if
end function
```

The function Merge creates two arrays of both halves (left and right) and then merges them to produce a single, sorted array.

9. Radix Sort

```
function RadixSort(A, N)
    max = findMax(A, N)
    exp = 1
    while max ÷ exp > 0 do
        CountSort(A, N, exp)
        exp = exp × 10
    end while
end function
function CountSort(A, N, exp)
    output = new array of size N
    count = new array of size 10 initialized to 0
    for 0 ≤ i < N do
        index = (A[i] ÷ exp) % 10
        count[index] = count[index] + 1
    end for
    for 1 ≤ i < 10 do
        count[i] = count[i] + count[i − 1]
    end for
    i = N − 1
    while i ≥ 0 do
        index = (A[i] ÷ exp) % 10
        output[count[index] − 1] = A[i]
        count[index] = count[index] − 1
        i = i − 1
    end while
    for 0 ≤ i < N do
        A[i] = output[i]
    end for
end function
```

10. Bucket Sort

```
function BucketSort(A, N, max)
    buckets = new array of empty lists of size N
    for i = 0 to N − 1 do
        index = floor(A[i] ÷ (max + 1) × N)
        append(buckets[index], A[i])
    end for
    for i = 0 to N − 1 do
```

```
        InsertionSort(buckets[i])
    end for
    k = 0
    for i = 0 to N − 1 do
        for j = 0 to len(buckets[i]) − 1 do
            A[k] = buckets[i][j]
            k = k + 1
        end for
    end for
end function
```

Last updated 2024-06-06 19:35:30 UTC