DATE : / /

Sorting Algorithms

- Prakash Hegade

Bubble Sort

Trace for 89 45 68 90 29 34 17

Iteration 01:

89 $\overset{?}{\Leftrightarrow}$ 45 68 90 29 34 17
45 89 $\overset{?}{\Leftrightarrow}$ 68 90 29 34 17
45 68 89 $\overset{?}{\Leftrightarrow}$ 90 $\overset{?}{\Leftrightarrow}$ 29 34 17
45 68 89 29 90 $\overset{?}{\Leftrightarrow}$ 34 17
45 68 89 29 34 90 $\overset{?}{\Leftrightarrow}$ 17
45 68 89 29 34 17 | 90 → 90 has found its home

Iteration 02:

45 $\overset{?}{\Leftrightarrow}$ 68 $\overset{?}{\Leftrightarrow}$ 89 $\overset{?}{\Leftrightarrow}$ 29 34 17 90
45 68 29 89 $\overset{?}{\Leftrightarrow}$ 34 17 90
45 68 29 34 89 $\overset{?}{\Leftrightarrow}$ 17 90
45 68 29 34 17 | 89 90

Iteration 03:

45 $\overset{?}{\Leftrightarrow}$ 68 $\overset{?}{\Leftrightarrow}$ 29 34 17 89 90
45 29 68 $\overset{?}{\Leftrightarrow}$ 34 17 89 90
45 29 34 68 $\overset{?}{\Leftrightarrow}$ 17 89 90

45 29 34 17 | 68 89 90

Iteration 04:

45 $\overset{?}{\leftrightarrow}$ 29 34 17 68 89 90

29 45 $\overset{?}{\leftrightarrow}$ 34 17 68 89 90

29 34 45 $\overset{?}{\leftrightarrow}$ 17 68 89 90

29 34 17 | 45 68 89 90

Iteration 05:

29 $\overset{?}{\leftrightarrow}$ 34 $\overset{?}{\leftrightarrow}$ 17 45 68 89 90

29 17 | 34 45 68 89 90

Iteration 06:

29 $\overset{?}{\leftrightarrow}$ 17 34 45 68 89 90

17 | 29 34 45 68 89 90

Iteration 07:

| 17 29 34 45 68 89 90

Algorithm

ALGORITHM BubbleSort(A[0..n-1])
// sorts given array by bubble sort
// Input: An array A[0...n-1] of orderable
// elements
// Output: Array A[0..n-1] sorted in ascending
// order

for $i \leftarrow 0$ to $n-2$ do
  for $j \leftarrow 0$ to $n-2-i$ do
    if $A[j+1] < A[j]$
      swap $A[j]$ and $A[j+1]$

Basic Operation:

Comparison.
- Is the same, for any kind of input

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1$$

$$= \sum_{i=0}^{n-2} n-2-i-0+1$$

$$= \sum_{i=0}^{n-2} n-i-1$$

$$= \sum_{i=0}^{n-2} n-1 - \sum_{i=0}^{n-2} i$$

$$= n-1 \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$= (n-1)(n-2-0+1) - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)(n-1) - \frac{(n-1)(n-2)}{2}$$

$$= \frac{2(n-1)(n-1) - (n-1)(n-2)}{2}$$

$$= \frac{(n-1)}{2}\left[2n-2-n+2\right]$$

$$= \frac{n-1}{2} * n$$

$$= \frac{n^2-n}{2} \approx \frac{n^2}{2}$$

$$c(n) \in \Theta(n^2)$$

Note

- For a decreasing array input, being the worst case, the number of key comparisons & swaps are the same.
- Bubble Sort is in-place

Scanned by CamScanner

Improvement:

In an iteration pass through, if no swaps were made then the list is sorted. We can ~~swaps~~ stop the algorithm.

Variants:

1. Recursive Bubble sort

2. Cocktail Sort
   - First stage loops through the array from left to right
   - Second stage loops through the array from right to left

3. Odd-Even Sort / Brick Sort
   - Each iteration has two phases
   - In odd phase, we perform a bubble sort on odd indexed elements & in even phase we perform bubble sort on even indexed elements.

9 8 4 2 1 6
9 8 4 2 1 6

One iteration

PAGE NO. :
DATE : / /

Selection Sort

Trace for: 89 45 68 90 29 34 17

| 89 45 68 90 29 34 17

17 | 45 68 90 29 34 89

17 29 | 68 90 45 34 89

17 29 34 | 90 45 68 89

17 29 34 45 | 90 68 89

17 29 34 45 68 | 90 89

17 29 34 45 68 89 | 90

Algorithm:

ALGORITHM SelectionSort (A[0...n-1])
// Sorts a given array by selection sort
// Input : An array A[0..n-1] of orderable elements
// Output : Array A[0..n-1] sorted in ascending order

PAGE NO. :
DATE : / /

```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]
            min ← j
    swap A[i] and A[min]
```

Basic Opeation:

Comparison.

$$c(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n-1-i-1+1$$

$$= \sum_{i=0}^{n-2} n-1-i \approx \frac{n^2}{2}$$

$$c(n) \in \Theta(n^2)$$

The number of key swaps is $\Theta(n)$, $n-1$ to be precise

Improvement:

Also take maximum on every pass & place it in correct position. In every pass, we keep track of both maximum & minimum & array becomes sorted from both ends.

Scanned by CamScanner

DATE : / /

Variant:

1. Stable Selection Sort.

i/p: 4 5 3 2 4 1
o/p: 1 5 3 2 4 ~~5~~ 4 → X

Stable must produce two keys with ~~so~~ value appear in same order in sorted output as they appear in input.

Stable selection sort - instead of swapping, the minimum element is placed in its position without swapping & by pushing every element one step forward.

2. Recursive Selection Sort.

PAGE NO
DATE / /

Insertion Sort

Trace for: 89 45 68 90 29 34 17

89 | 45 68 90 29 34 17
45 89 | 68 90 29 34 17
45 68 89 | 90 29 34 17
45 68 89 90 | 29 34 17
29 45 68 89 90 | 34 17
29 34 45 68 89 90 | 17
17 29 34 45 68 89 90

Algorithm:

```
ALGORITHM InsertionSort(A[0..n-1])
//sorts the given array by insertion sort
//Input: An Array A[0..n-1] of orderable
         elements
//output: An array A[0..n-1] sorted in increasing
          order
for i←1 to n-1 do
   v←A[i]
   j←i-1
   while j≥0 and A[j]>v do
        A[j+1]←A[j]
        j←j-1
   A[j+1]←v
```

PAGE NO. :
DATE : / /

Basic Operation: $A[j] > v$

Worst Case: Descending order Sorted input.

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} i-1-0+1 \quad = \sum_{i=1}^{n-1} i$$

$$= \frac{(n-1)n}{2} \quad \in \Theta(n^2)$$

Best Case: Ascending order sorted input

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 \quad = n-1-1+1$$

$$= n-1$$

$$\in \Omega(n)$$

Average Case:

Consider an insertion of 3rd element, we have 3 combinations with same probability. No of Comparisons $= 1+2+3$

$$\text{Average} = \frac{1+2+3}{3} = 2$$

element can be placed at any position.

PAGE NO:
DATE: / /

Generalizing:

$$\frac{\sum_{j=1}^{i} j}{i} = \frac{i(i+1)}{2i} = \frac{i+1}{2}$$ no of comparisons.

So,

$$C_{avg}(n) = \sum_{i=1}^{n-1} \frac{i+1}{2} = \frac{1}{2}\sum_{i=1}^{n-1} i+1$$

$$= \frac{1}{2}\left[2+3+4+5+\ldots n\right]$$

$$= \frac{1}{2}\left[\frac{n(n+1)}{2} - 1\right]$$

$$= \frac{1}{2}\left[\frac{n^2}{2} + \frac{n}{2} - \frac{1}{2}\right]$$

$$= \frac{n^2}{4} - \frac{n}{4} - \frac{1}{2}$$

$$\approx n^2$$

$$\in \Theta(n^2)$$

PAGE NO. :
DATE : / /

Improvements & Variants:

1. Finding position of insert with left to right scan or right to left scan

2. Binary insertion sort

3. Recursive Insertion Sort

4.