Algorithms

(Sorting)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

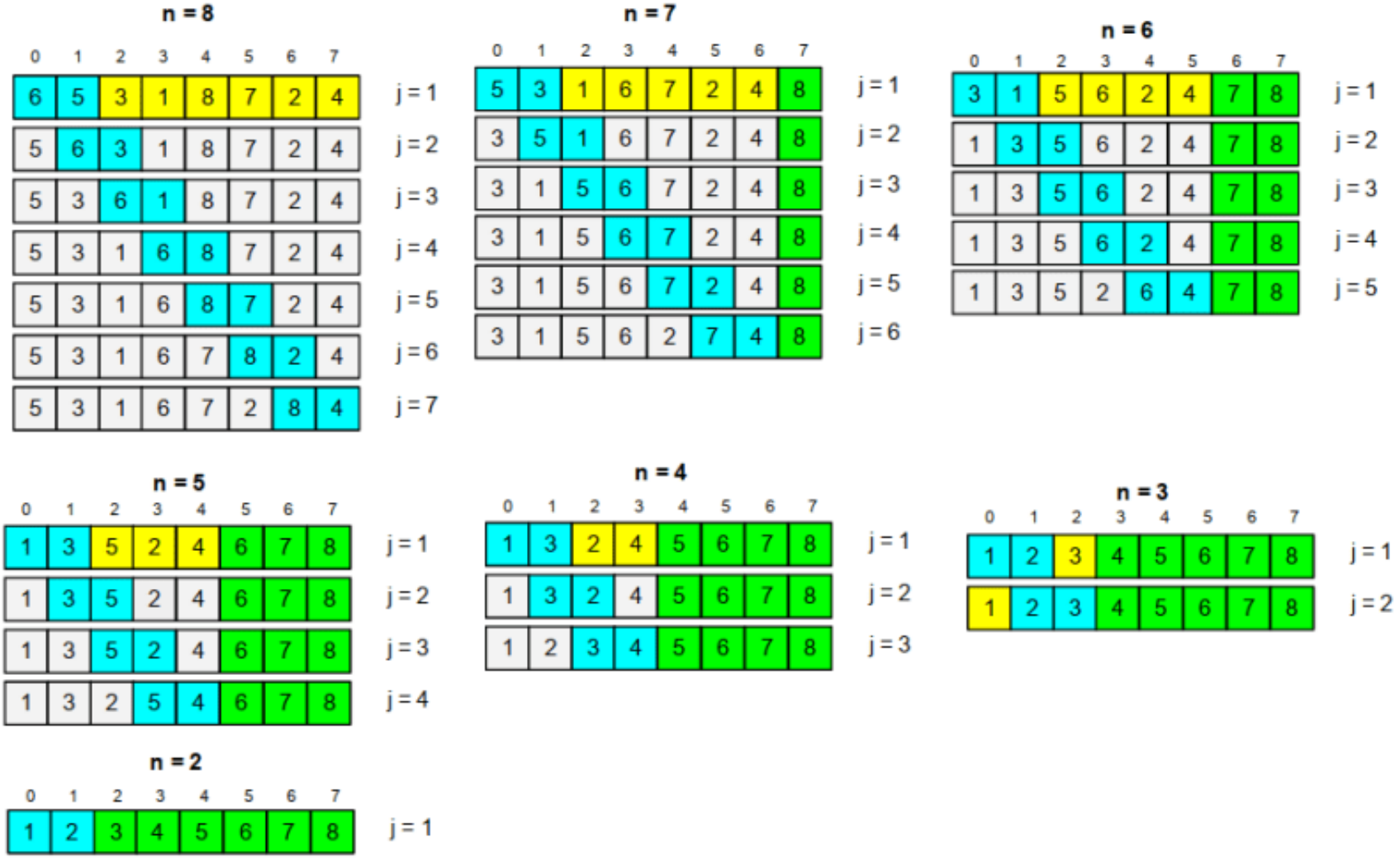
April 26, 2021

Contents

Comparison table of sorting algorithms

Algorithm	Best	Average	Worst	Space	Stable?	Method
Bubble sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	✓	Exchange
Selection sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	✗	Selection
Insertion sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	✓	Insertion
Merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	✓	Merge
Quicksort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\log n)$	✗	Partition
Heap sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	✗	Selection
Tree sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	✓	Insertion

Bubble sort



https://i0.wp.com/eleni.blog/wp-content/uploads/2019/06/bubble_sort.png

Bubble sort

BUBBLE-SORT($A[0..n-1]$)

Input: An array $A[0..n-1]$ of n orderable elements
Output: Array $A[0..n-1]$ sorted in nondecreasing order

```
1. for i ← 0 to n − 2 do
2.   for j ← 0 to n − i − 2 do
3.     if A[j] > A[j + 1] then
4.       SWAP(A[j + 1], A[j])
```

Selection sort

Source: `https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm`

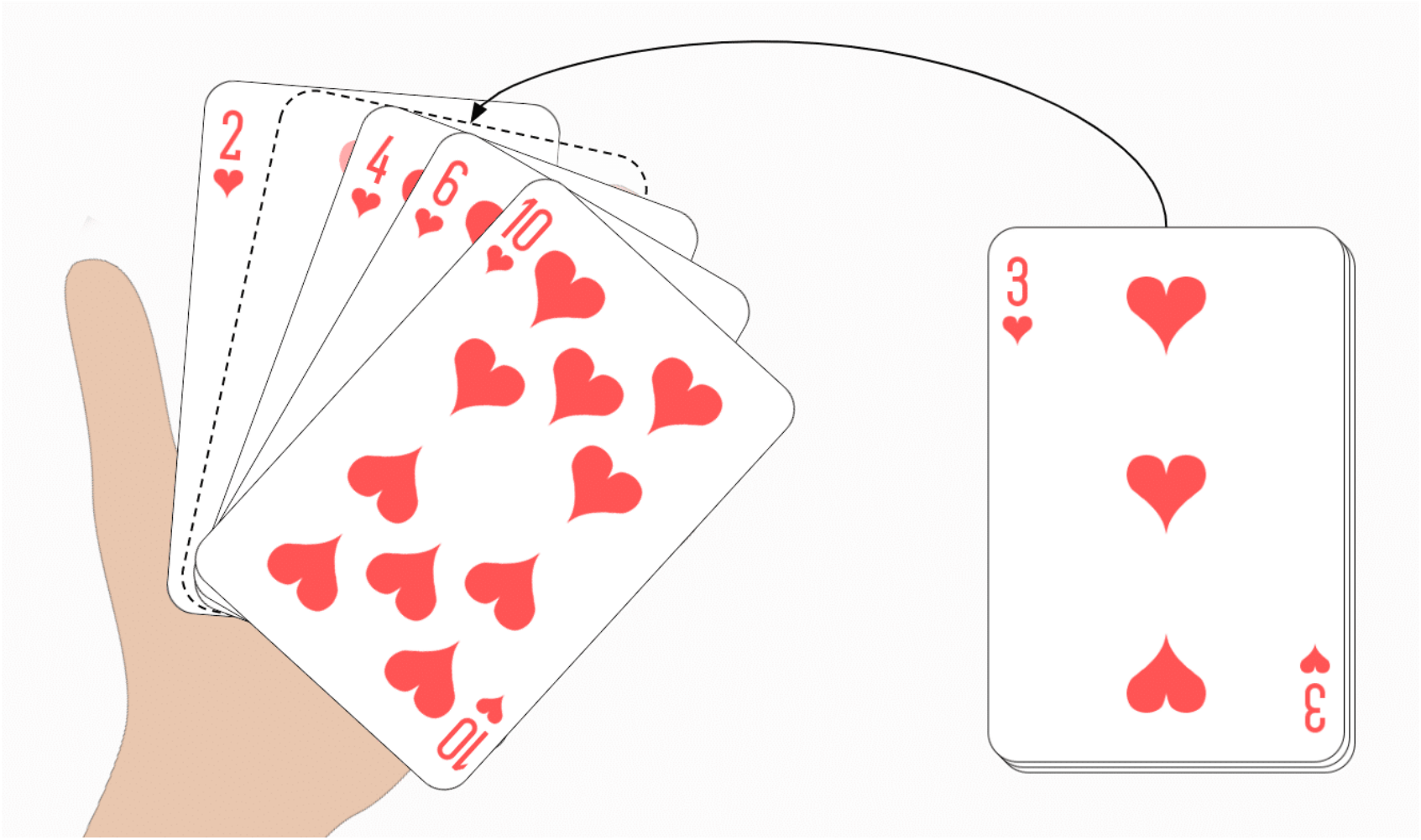
Selection sort

SELECTION-SORT($A[0..n-1]$)

Input: An array $A[0..n-1]$ of n orderable elements
Output: Array $A[0..n-1]$ sorted in nondecreasing order

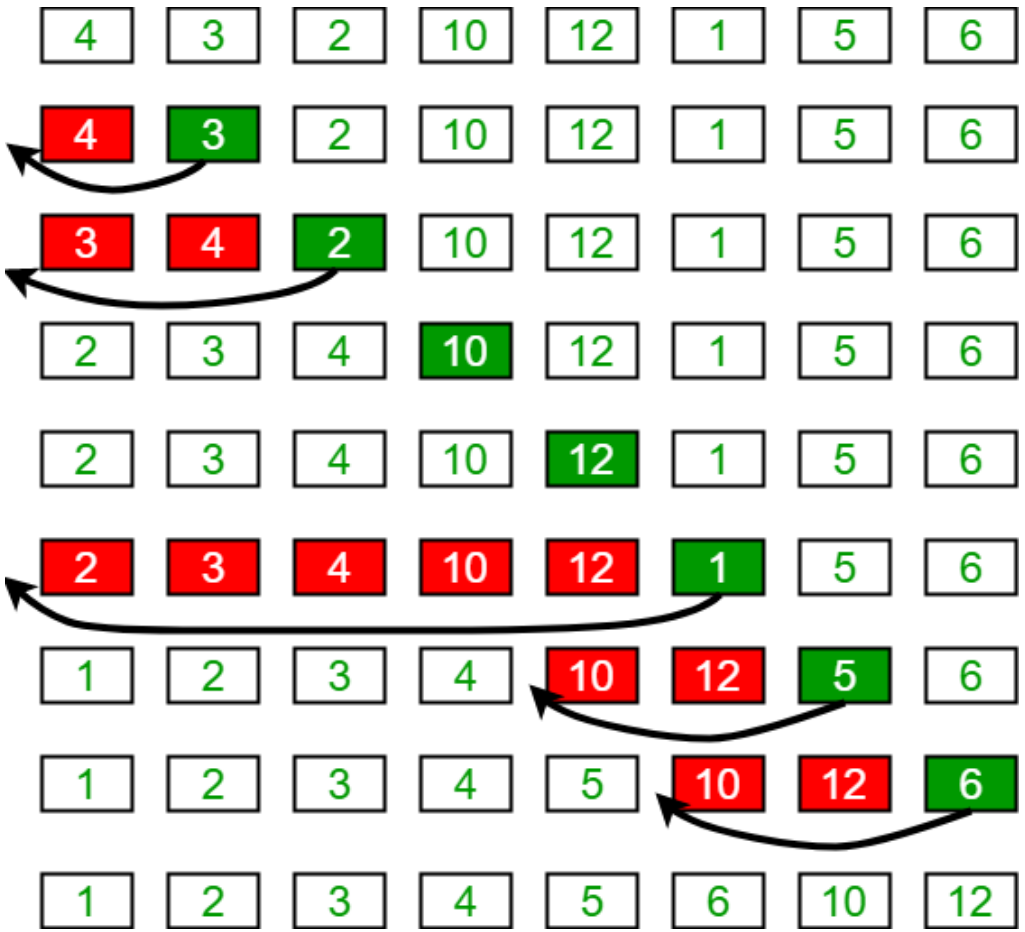
```
1. for i ← 0 to n − 2 do
2.   min ← i
3.   for j ← i + 1 to n − 1 do
4.     if A[j] < A[min] then
5.       min ← j
6.   SWAP(A[i], A[min])
```

Insertion sort



https://www.happycoders.eu/wp-content/uploads/2020/05/Insertion_Sort_Playing_Card_Example.png

Insertion sort



Source: `https://media.geeksforgeeks.org/wp-content/uploads/insertionsort.png`

Insertion sort

INSERTION-SORT($A[0..n-1]$)

Input: An array $A[0..n-1]$ of n orderable elements
Output: Array $A[0..n-1]$ sorted in nondecreasing order

1. **for** $i \leftarrow 1$ **to** $n-1$ **do**
2. $\quad v \leftarrow A[i]$
3. $\quad j \leftarrow i-1$
4. $\quad$ **while** $j \geq 0$ **and** $A[j] > v$ **do**
5. $\quad\quad A[j+1] \leftarrow A[j]$
6. $\quad\quad j \leftarrow j-1$
7. $\quad A[j+1] \leftarrow v$

Heap sort

HEAP-SORT$(A[0..n-1])$

Sorts an array using heap sort
Input: An array $A[0..n-1]$ of orderable elements
Output: Array $A[0..n-1]$ sorted in nondecreasing order
1. BUILD-MAXHEAP$(A[0..n-1])$
2. **for** $i \leftarrow n-1$ **to** 0 **do**
3. SWAP$(A[0], A[i])$ ▷ get correct element at index i
4. MAXHEAPIFY$(A[0..i-1], 0)$ ▷ down-heap bubbling

BUILD-MAXHEAP$(A[0..n-1])$

Input: An array $A[0..n-1]$
Output: Array $A[0..n-1]$ representing a max heap
1. **for** $i \leftarrow n/2-1$ **to** 0 **do**
2. MAXHEAPIFY$(A[0..n-1], i)$ ▷ down-heap bubbling

Heap sort

MaxHeapify$(A[0..n-1], i)$

Restore the max heap property for index i

Input: An array $A[0..n-1]$ of orderable elements and index i

Output: Array $A[0..n-1]$ satisfying the max heap property for index i

1. $max \leftarrow i$; $left \leftarrow 2i+1$; $right \leftarrow 2i+2$
2. **if** $left < n$ **and** $A[left] > A[max]$ **then**
3. $\quad max \leftarrow left$
4. **if** $right < n$ **and** $A[right] > A[max]$ **then**
5. $\quad max \leftarrow right$
6. **if** $max \neq i$ **then**
7. $\quad$ **Swap**$(A[i], A[max])$
8. $\quad$ **MaxHeapify**$(A[0..n-1], max)$ $\triangleright$ down-heap bubbling

Complexity

- Time complexity.

  $T_{\textsc{MaxHeapify}}(n) \in \mathcal{O}(\log n)$

  $T_{\textsc{Build-MaxHeapify}}(n) \in \mathcal{O}(n)$

  $T_{\textsc{Sort}}(n) = T_{\textsc{Build-MaxHeapify}}(n) + \Theta(n) \times T_{\textsc{MaxHeapify}}(n)$

  Solving, $T_{\textsc{Sort}}(n) \in \mathcal{O}(n \log n)$

- Space complexity.

  Extra space is $\Theta(1)$.