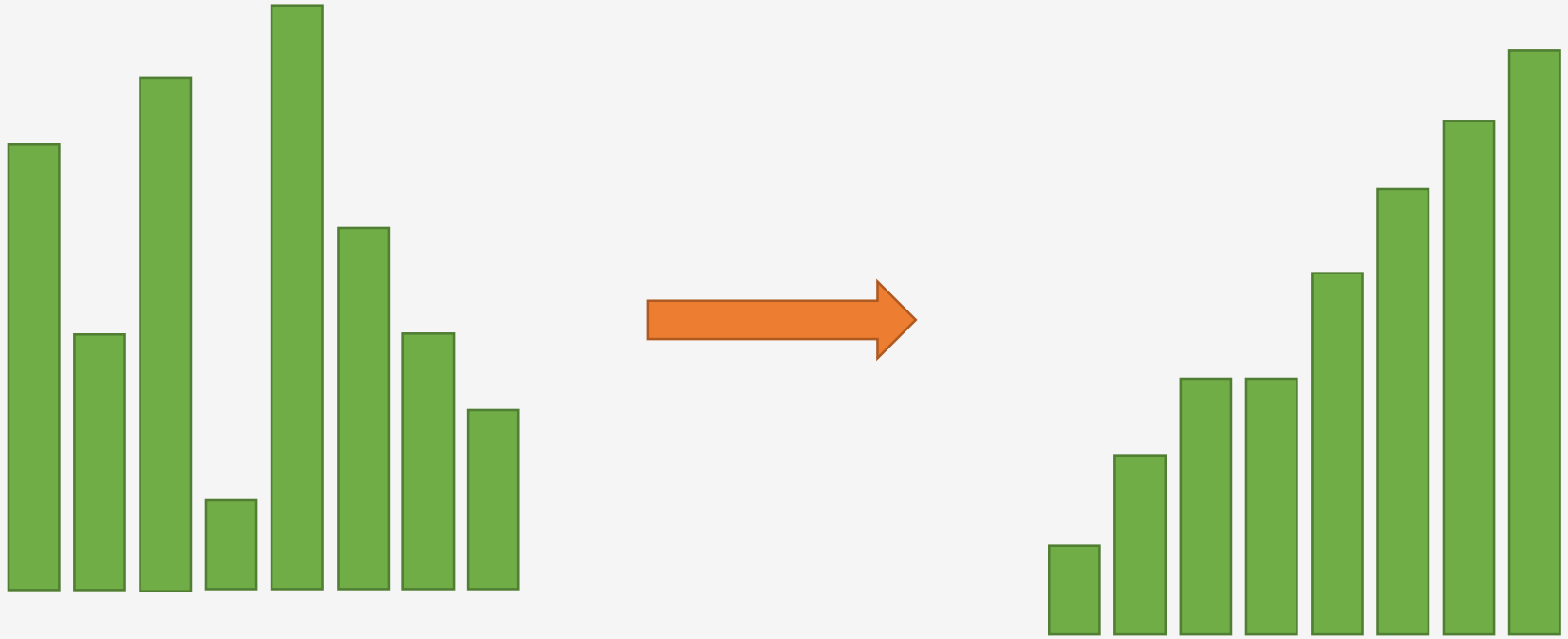
Design and Analysis of Algorithms

03-06
Divide – and – Conquer

Selection Sort

Imran Ihsan
Assistant Professor, Department of Computer Science
Air University, Islamabad, Pakistan
www.imranihsan.com

Sorting Problem



Sorting

Input: Sequence A[1 . . . n].

Output: Permutation A’[1 . . . n] of A[1 . . . n] in non-decreasing order.

Why Sorting?

Sorting data is an important step of many efficient algorithms.

Sorted data allows for more efficient queries.

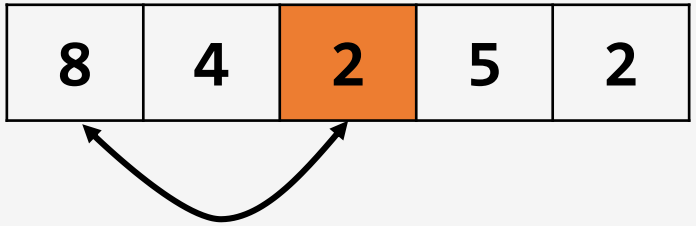
Selection Sort

8	4	2	5	2

Selection Sort: Example

Find a minimum by scanning the array

Selection Sort: Example



Find a minimum by scanning the array

Swap it with the first element

Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

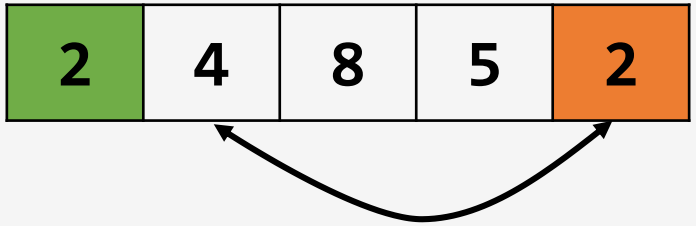
Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example



Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

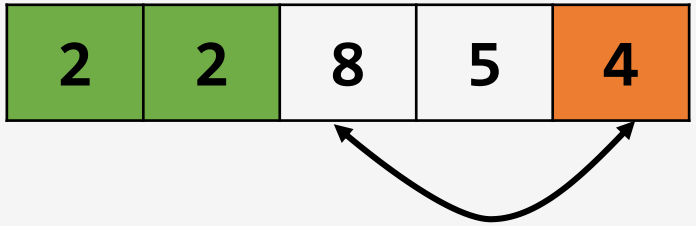
Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example



Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort: Example

Find a minimum by scanning the array

Swap it with the first element

Repeat with the remaining part of the array

Selection Sort

```
SelectionSort(A[1 . . . n])
      for i from 1 to n:
         minIndex ← i
         for j from i + 1 to n:
            if A[j ] < A[minIndex]:
               minIndex ← j
         {A[minIndex] = min A[i . . . n]}
         swap(A[i ], A[minIndex])
         {A[1 . . . i ] is in final position}
```

Online Visualization: Selection Sort

Selection Sort

Lemma

The running time of SelectionSort(A[1 . . . n]) is $O(n^2)$.

Proof

n iterations of outer loop,

at most n iterations of inner loop.

Too Pessimistic Estimate?

As `i` grows, the number of iterations of the inner loop decreases:

`j` iterates from `i + 1` to `n`.

A more accurate estimate for the total number of iterations of the inner loop is

$$(n - 1) + (n - 2) + \cdots + 1$$

We will show that this sum is $\Theta(n^2)$ implying that our initial estimate is actually tight.

Arithmetic Series

Lemma

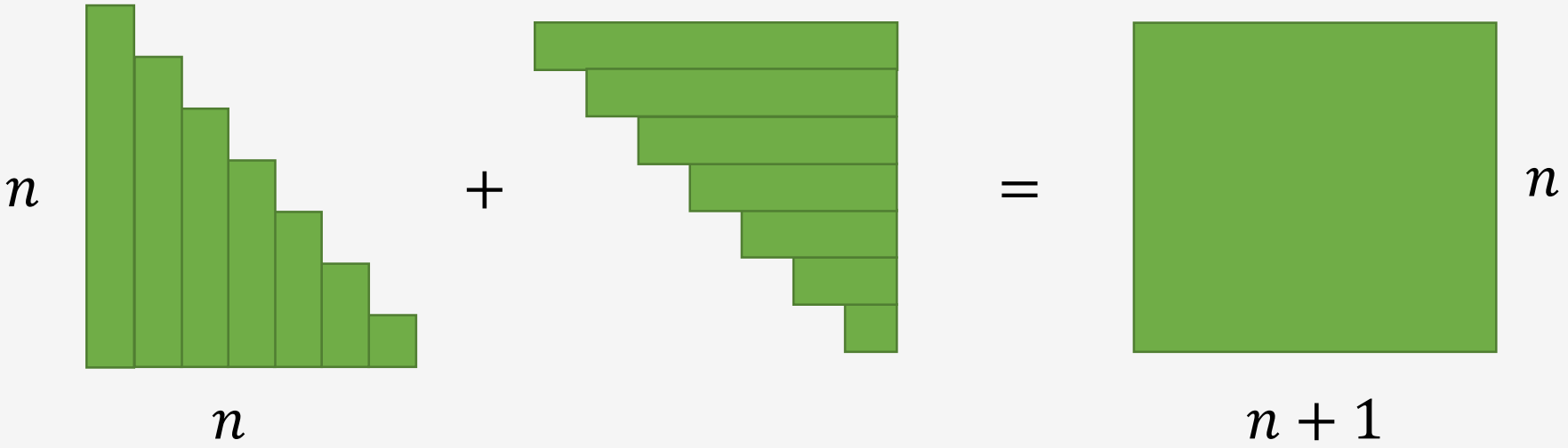
$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Proof

$$\begin{array}{cccc} 1 & 2 & \cdots & n \\ n & n-1 & \cdots & 1 \\ \hline n+1 & n+1 & \cdots & n+1 \end{array}$$

$$n(n+1)$$

Alternative proof



Selection Sort: Summary

Selection sort is an easy to implement algorithm with running time $O(n^2)$.

Sorts in Place: requires a constant amount of extra memory.

There are many other quadratic time sorting algorithms:

e.g., insertion sort, bubble sort.