DESIGN AND ANALYSIS OF ALGORITHMS

UE19CS251

Shylaja S S
Department of Computer Science
& Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

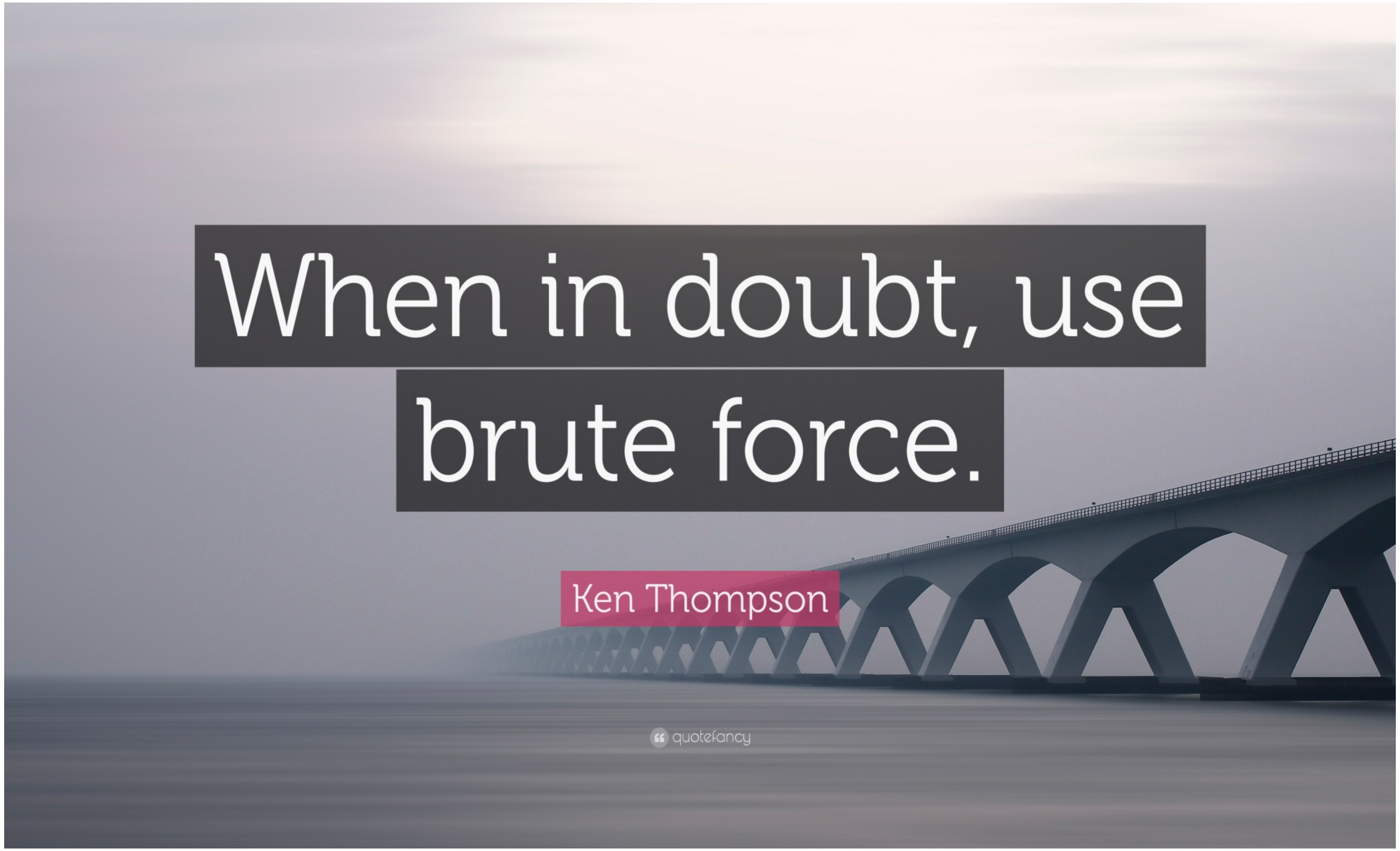
Brute Force: Selection Sort

Major Slides Content: Anany Levitin

Shylaja S S

Department of Computer Science & Engineering

Brute Force



> When in doubt, use brute force.
>
> Ken Thompson

Brute Force

- Brute Force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved

Brute Force

- In computer science, **brute-force search** or **exhaustive search**, also known as **generate and test**, is a very general problem-solving technique and algorithmic paradigm that consists of:
    - systematically enumerating all possible candidates for the solution
    - checking whether each candidate satisfies the problem's statement

https://en.wikipedia.org/wiki/Brute-force_search

Brute Force

- A brute-force algorithm to find the divisors of a natural number n would
  - enumerate all integers from 1 to n
  - check whether each of them divides n without remainder
- A brute-force approach for the eight queens puzzle would
  - examine all possible arrangements of 8 pieces on the 64-square chessboard
  - check whether each (queen) piece can attack any other, for each arrangement
  - The brute-force method for finding an item in a table (linear search)
  - checks all entries of the table, sequentially, with the item

Brute Force

- A brute-force search is simple to implement, and will always find a solution if it exists
- But, its cost is proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases (Combinatorial explosion)
- Brute-force search is typically used
  - when the problem size is limited
  - when there are problem-specific heuristics that can be used to reduce the set of candidate solutions to a manageable size
  - when the simplicity of implementation is more important than speed

Brute Force Sorting Algorithms

- Selection Sort
- Bubble Sort

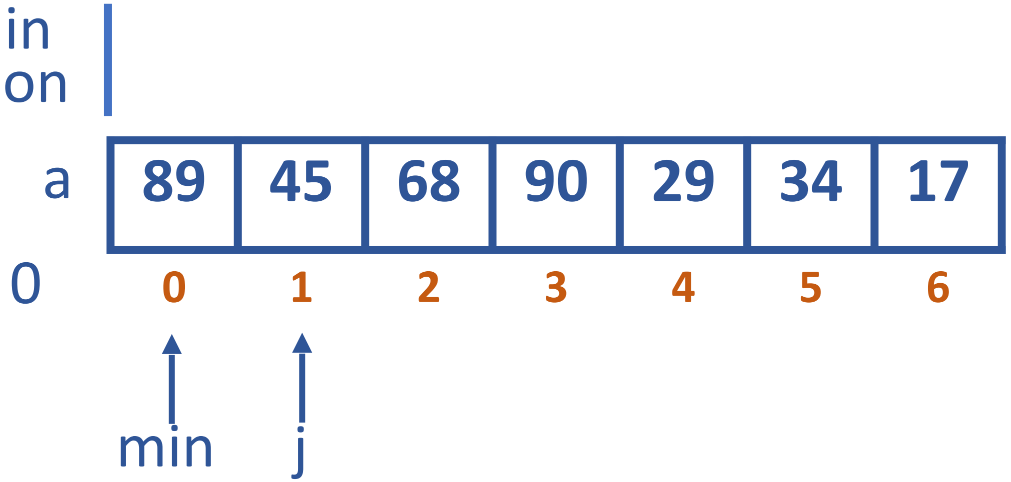
Selection Sort

- Scan the array to find its smallest element and swap it with the first element, putting the smallest element in its final position in the sorted list
- Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second element, putting the second smallest element in its final position
- Generally, on pass i ($0 \le i \le n-2$), find the smallest element in $A[i..n-1]$ and swap it with $A[i]$

- $A[0] \le A[1] \le A[2] \ldots \le A[i-1] \mid A[i], \ldots.., A[min], \ldots., A[n-1]$
- in their final positions the last n – i elements

Selection Sort

elements in
final position



a	89	45	68	90	29	34	17
	0	1	2	3	4	5	6

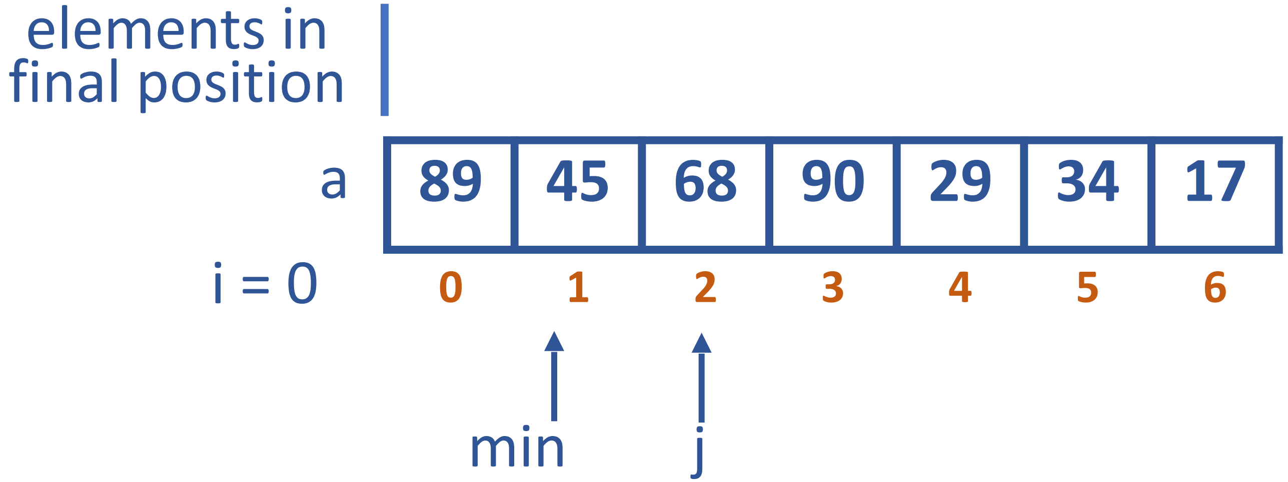
Pass 0 i = 0

45 < 89 ?? ☑

min = 01

j = 1

Selection Sort



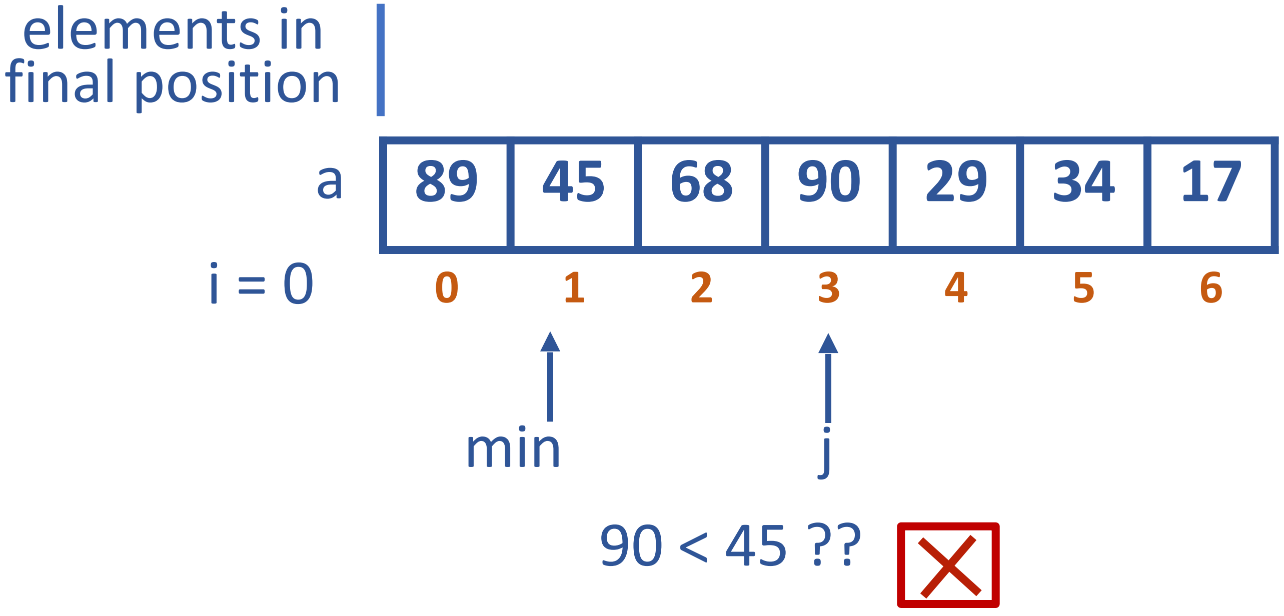
68 < 45 ?? ☒

min = 1

j = 2

DESIGN AND ANALYSIS OF ALGORITHMS

Selection Sort



elements in final position

a	89	45	68	90	29	34	17
i = 0	0	1	2	3	4	5	6

min

j

90 < 45 ?? ☒

min = 1

j = 3

Selection Sort

elements in
final position

a	89	45	68	90	29	34	17
i = 0	0	1	2	3	4	5	6

min ↑ (index 1)   j ↑ (index 4)

29 < 45 ?? ☑

min = 4

j = 4

Selection Sort

elements in
final position

a	**89**	**45**	**68**	**90**	**29**	**34**	**17**
i = 0	0	1	2	3	4	5	6

min

j

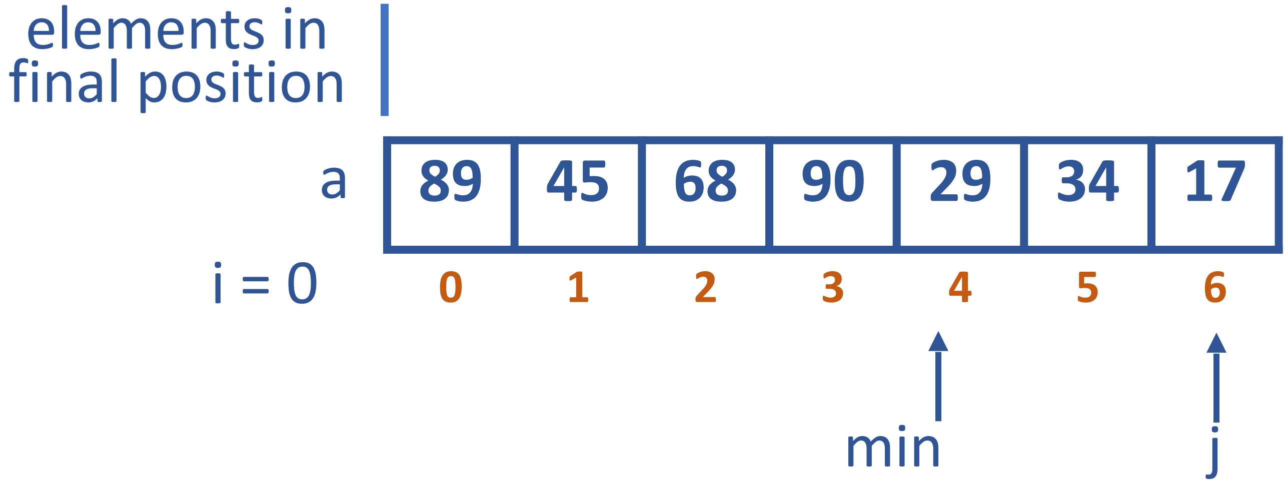
34 < 29 ?? ☒

min = 4

j = 5

DESIGN AND ANALYSIS OF ALGORITHMS

Selection Sort

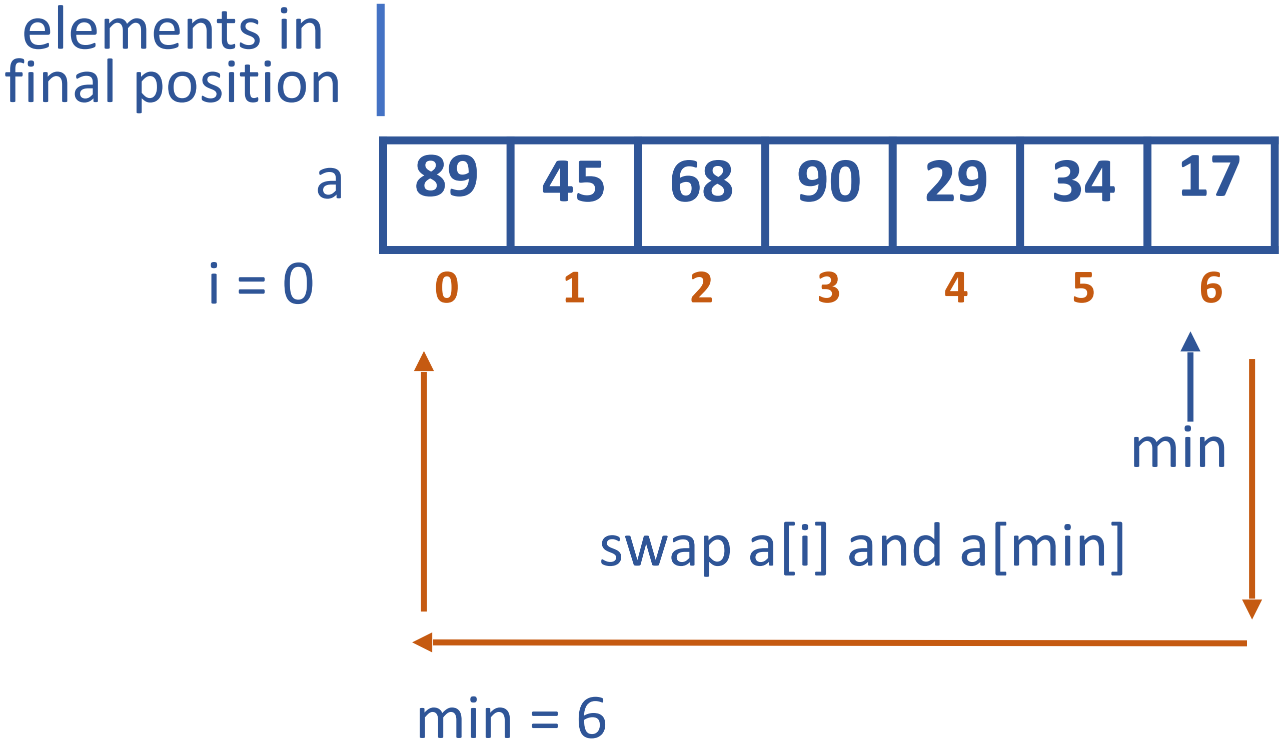


17 < 29 ?? ☑

min = 6

j = 6

Selection Sort



elements in final position

a

89	45	68	90	29	34	17
0	1	2	3	4	5	6

i = 0

min

swap a[i] and a[min]

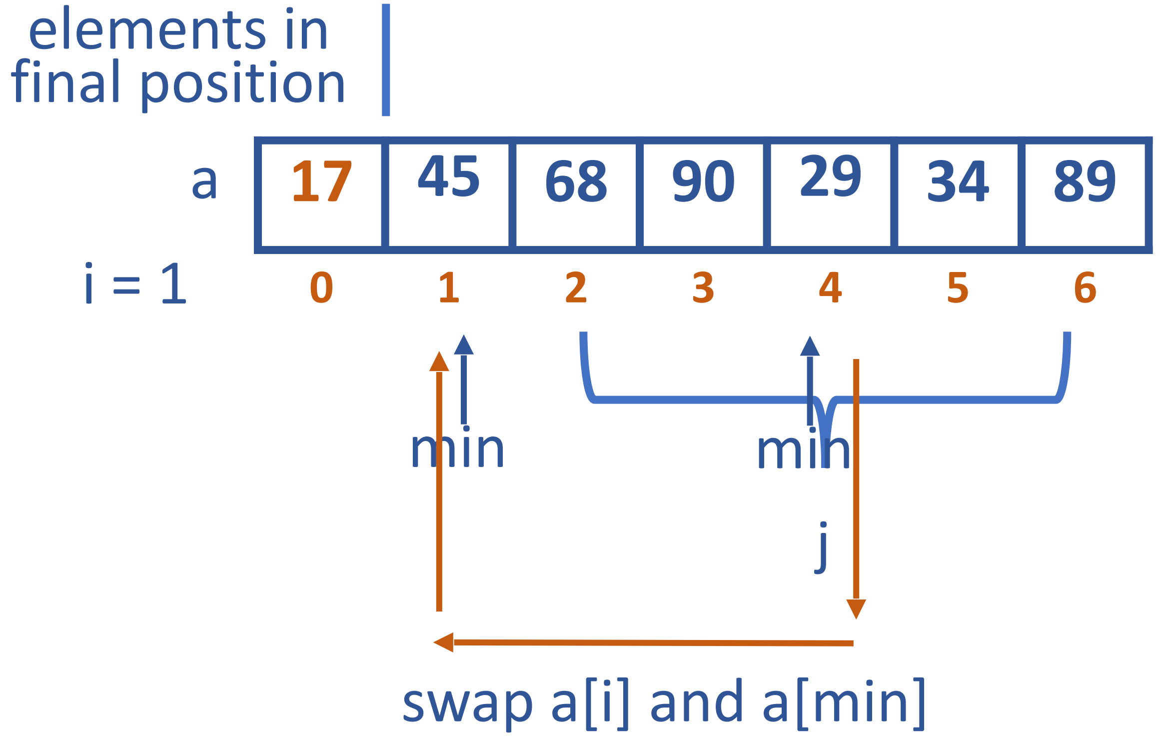
min = 6

Selection Sort

elements in
final position

a	17	45	68	90	29	34	89
i = 0	0	1	2	3	4	5	6

Selection Sort



elements in final position

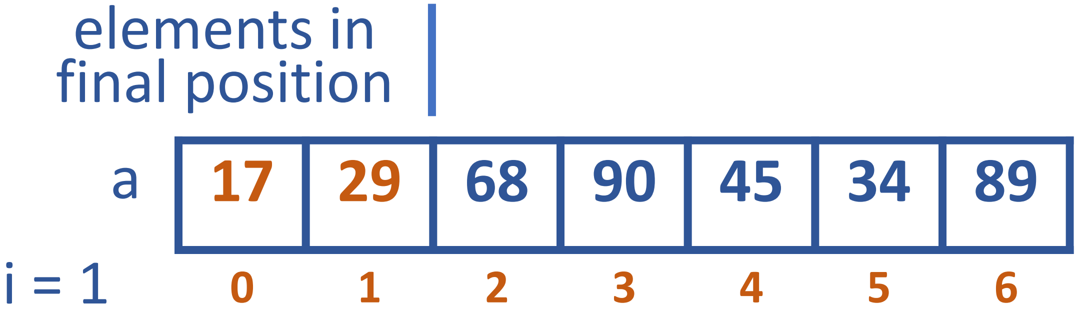
a	17	45	68	90	29	34	89
	0	1	2	3	4	5	6

Pass 1 i = 1

swap a[i] and a[min]

min = 4

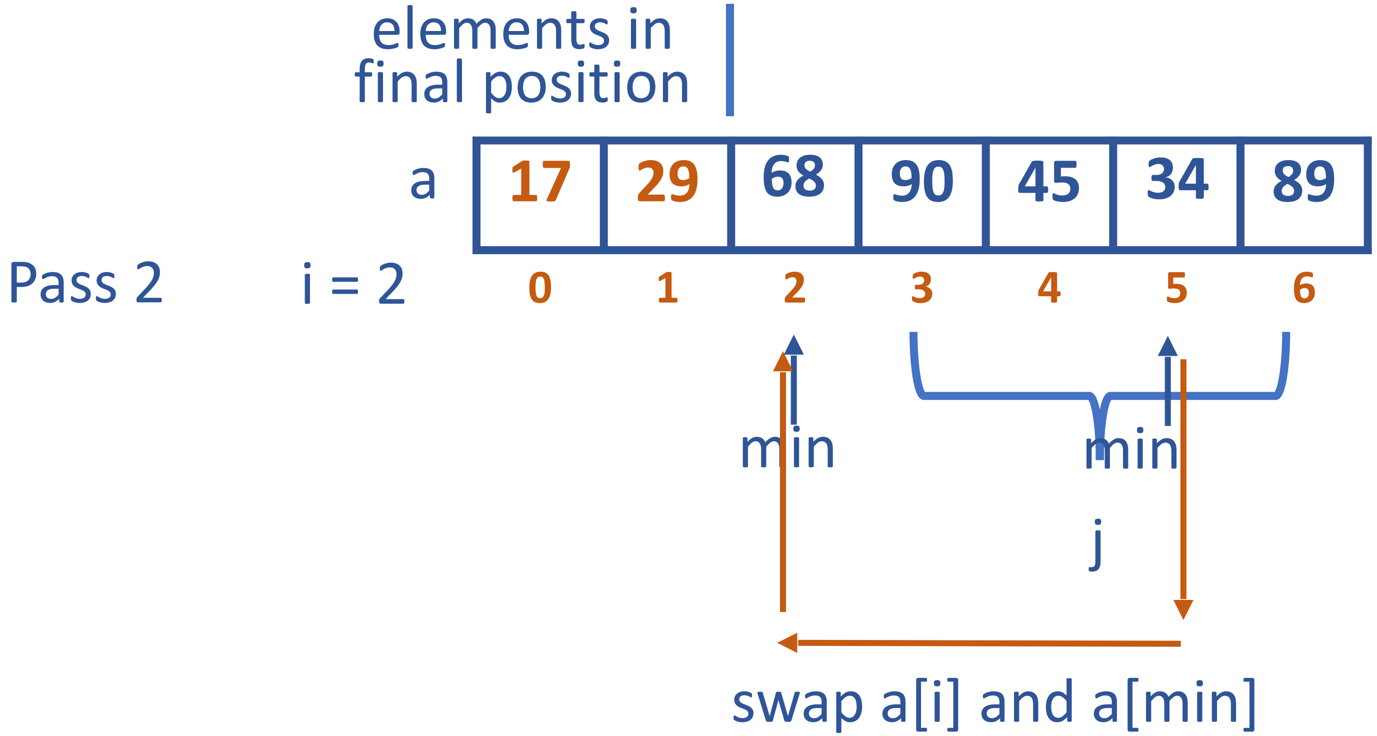
Selection Sort



elements in final position

a	**17**	**29**	68	90	45	34	89
i = 1	0	1	2	3	4	5	6

Selection Sort



elements in final position

a	17	29	68	90	45	34	89
	0	1	2	3	4	5	6

Pass 2 i = 2

min

min

j

swap a[i] and a[min]

min = 5

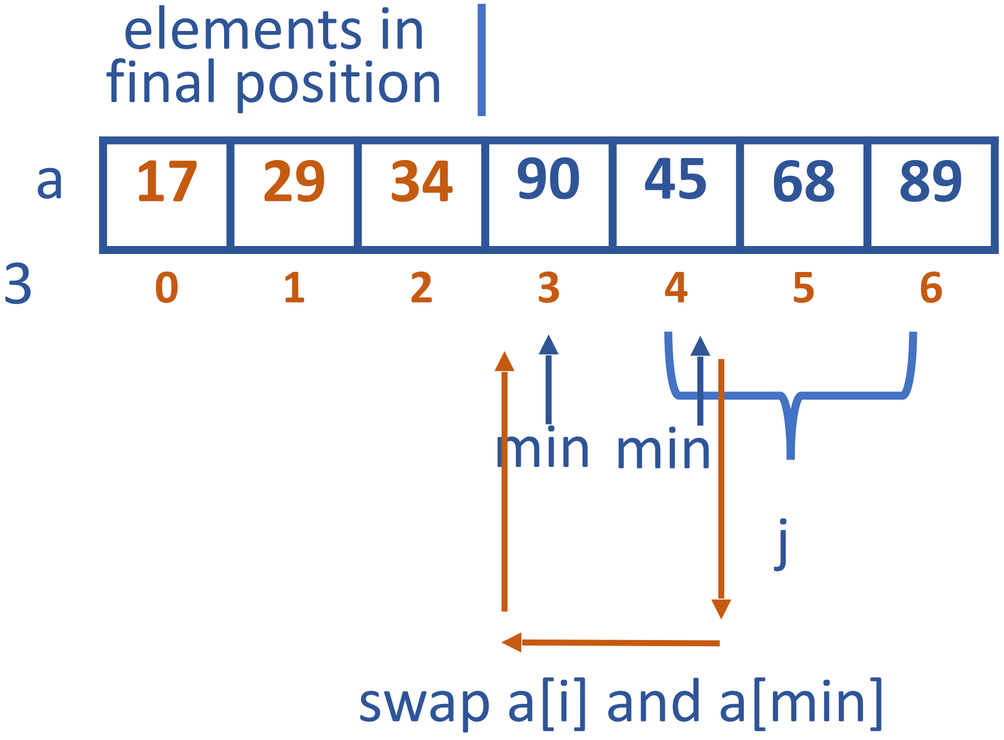
DESIGN AND ANALYSIS OF ALGORITHMS

Selection Sort

elements in final position

a	17	29	34	90	45	68	89
i = 2	0	1	2	3	4	5	6

Selection Sort



elements in
final position

a

17	**29**	**34**	**90**	**45**	**68**	**89**
0	1	2	3	4	5	6

Pass 3 i = 3

swap a[i] and a[min]

min = 4

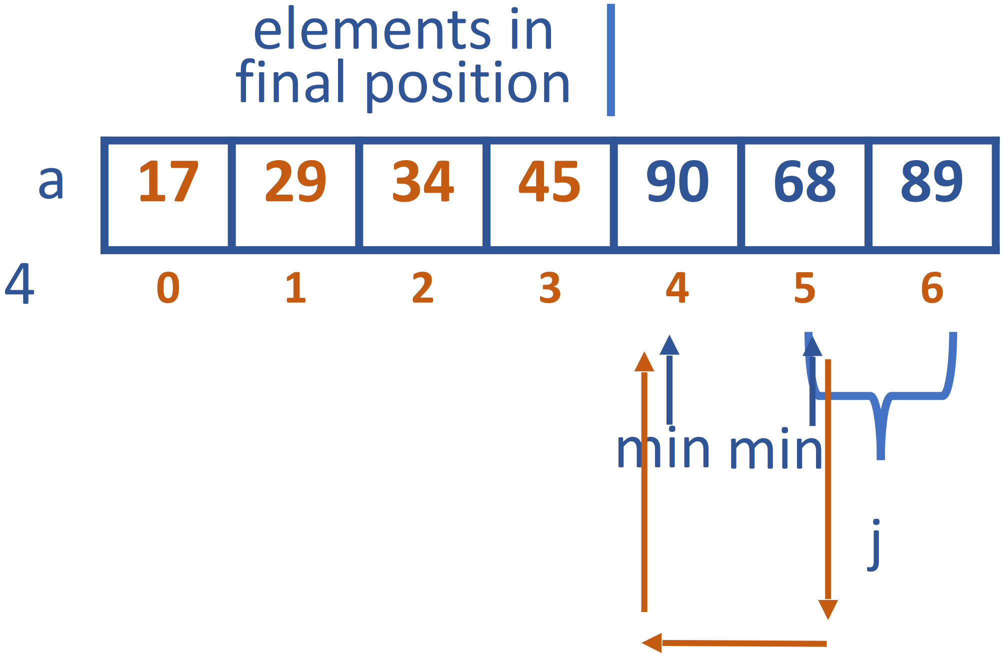
Selection Sort

elements in
final position

a	**17**	**29**	**34**	**45**	**90**	**68**	**89**
i = 3	**0**	**1**	**2**	**3**	**4**	**5**	**6**

DESIGN AND ANALYSIS OF ALGORITHMS

Selection Sort



elements in
final position

a	17	29	34	45	90	68	89
	0	1	2	3	4	5	6

Pass 4 i = 4

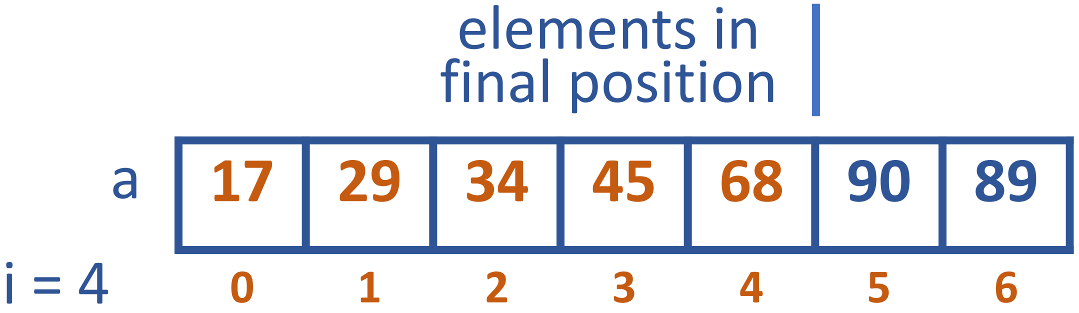
min min

j

swap a[i] and a[min]

min = 5

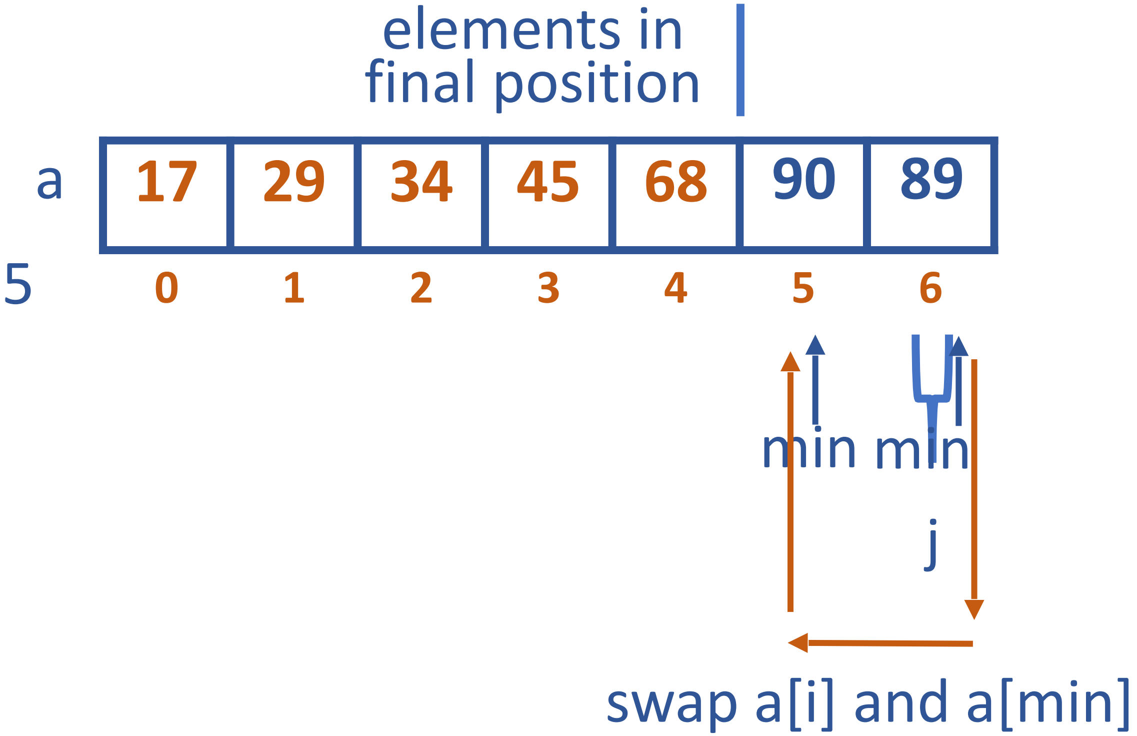
Selection Sort



elements in
final position

a	17	29	34	45	68	90	89
i = 4	0	1	2	3	4	5	6

Selection Sort



elements in final position

a

17	29	34	45	68	90	89
0	1	2	3	4	5	6

Pass 5 i = 5

min min

j

swap a[i] and a[min]

min = 6

DESIGN AND ANALYSIS OF ALGORITHMS

Selection Sort

elements in
final position

a	17	29	34	45	68	89	90
i = 5	0	1	2	3	4	5	6

Original array	89	45	68	90	29	34	17
	0	1	2	3	4	5	6

Array after sorting	17	29	34	45	68	89	90
	0	1	2	3	4	5	6

Selection Sort

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

Selection Sort Analysis

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

Selection Sort is a $\Theta(n^2)$ algorithm

THANK YOU

Shylaja S S

Department of Computer Science
& Engineering

shylaja.sharath@pes.edu