COMPARISION OF DIFFERENT SORTING ALGORITHMS IN RISCV

SOURABH.T PES1UG20EC339
PRUTHVIK KASHYAP S PES1UG20EC326

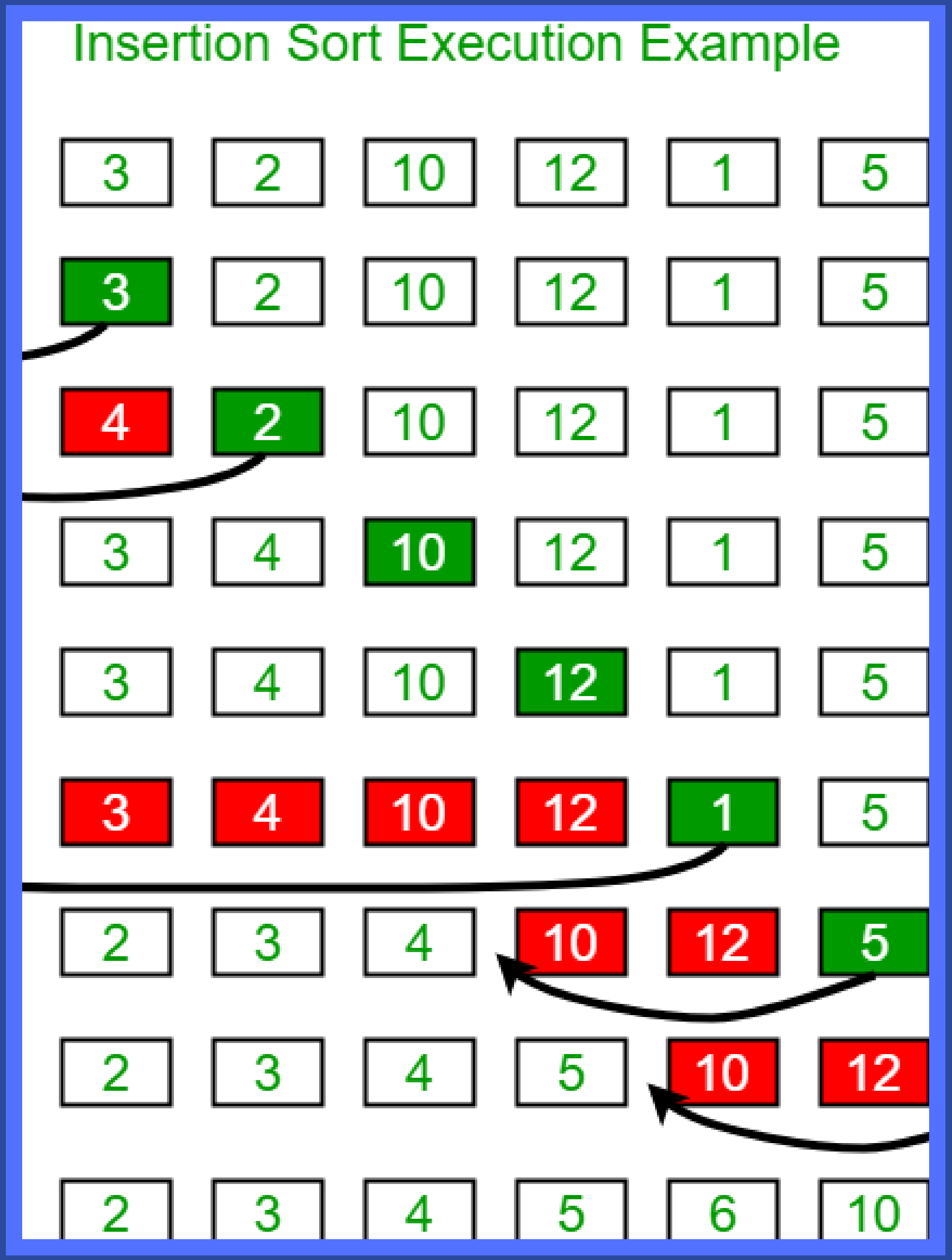
INTRODUCTION

1. INSERTION SORT
2. SELECTION SORT
3. BINARY SEARCH
4. QUICK SORT
5. MERGE SORT

1

INSERTION SORT

- THIS ALGORITHM IS ONE OF THE SIMPLEST ALGORITHM WITH SIMPLE IMPLEMENTATION
- ASICALLY, INSERTION SORT IS EFFICIENT FOR SMALL DATA VALUES
- INSERTION SORT IS ADAPTIVE IN NATURE, I.E. IT IS APPROPRIATE FOR DATA SETS WHICH ARE ALREADY PARTIALLY SORTED
- TIME COMPLEXITY $O(N^2)$



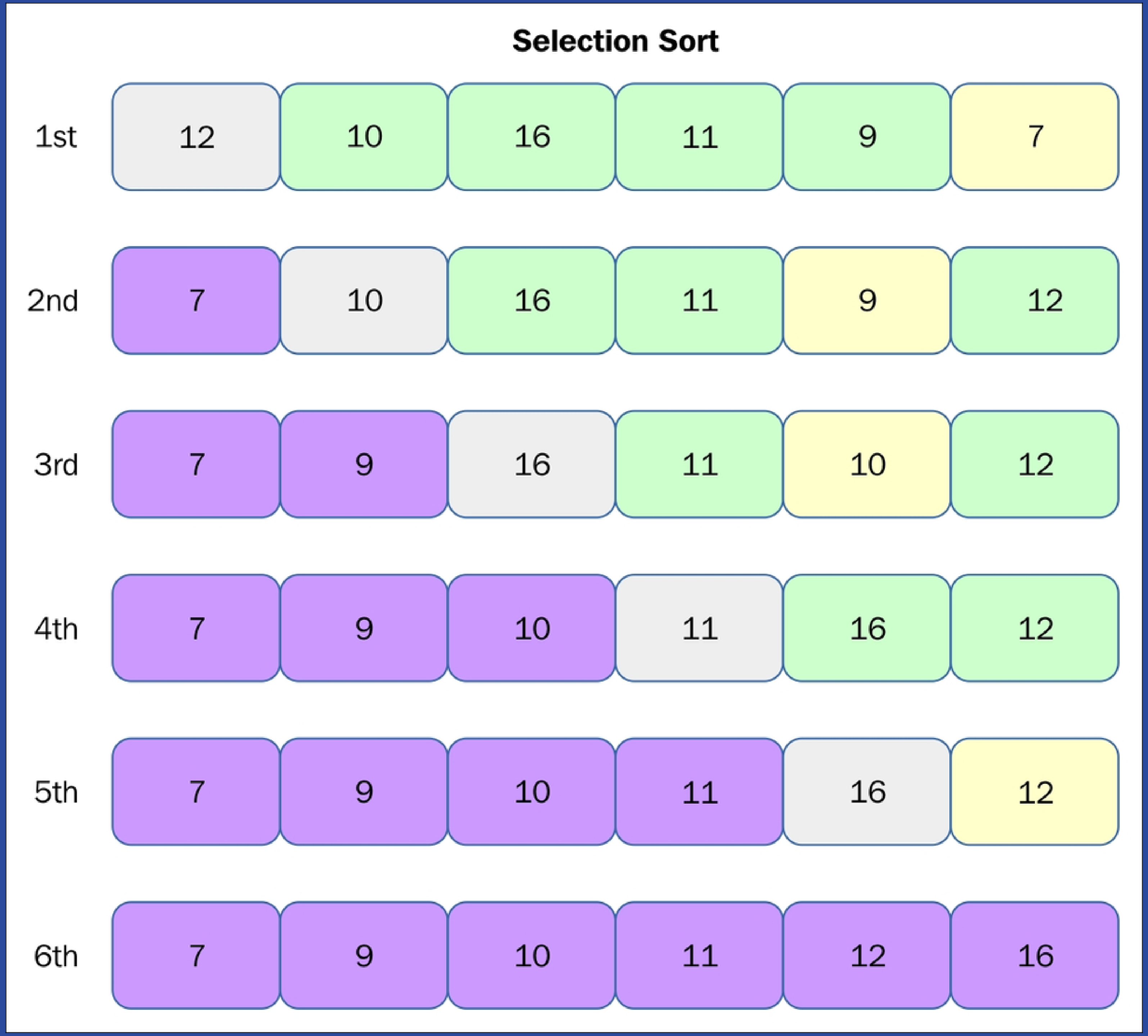
ALGORITHM

- *Iterate from arr[1] to arr[N] over the array.*
- *Compare the current element (key) to its predecessor.*
- *If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.*

CODE

2
SELECTION SORT

- THE SELECTION SORT ALGORITHM SORTS AN ARRAY BY REPEATEDLY FINDING THE MINIMUM ELEMENT (CONSIDERING ASCENDING ORDER) FROM THE UNSORTED PART AND PUTTING IT AT THE BEGINNING.
- TIME COMPLEXITY $O(N^2)$



ALGORITHM

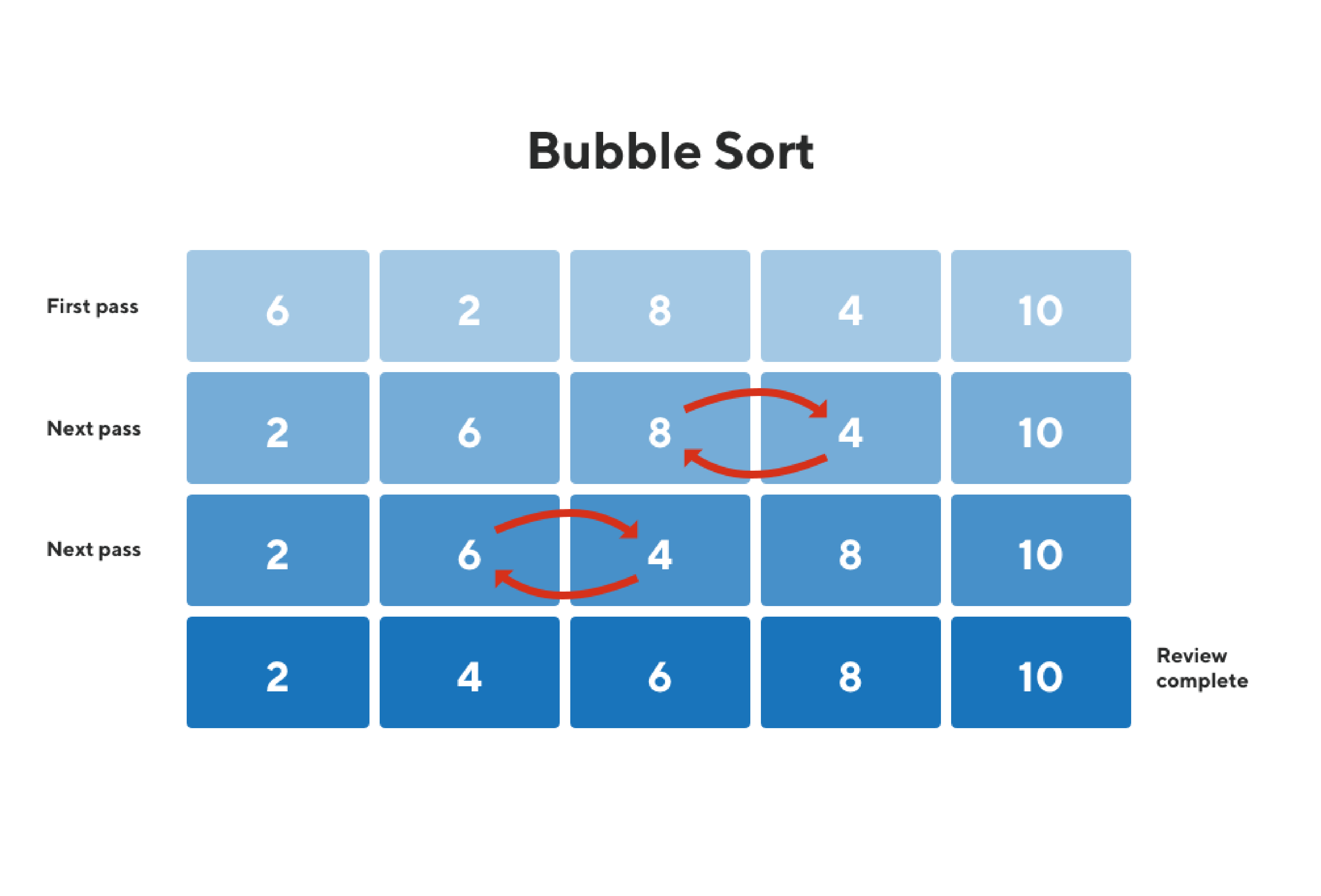
- *Initialize minimum value(min_idx) to location 0.*
- *Traverse the array to find the minimum element in the array.*
- *While traversing if any element smaller than min_idx is found then swap both the values.*
- *Then, increment min_idx to point to the next element.*
- *Repeat until the array is sorted.*

CODE

3

BUBBLE SORT

- *BUBBLE SORT IS THE SIMPLEST SORTING ALGORITHM THAT WORKS BY REPEATEDLY SWAPPING THE ADJACENT ELEMENTS IF THEY ARE IN THE WRONG ORDER.*
- *THIS ALGORITHM IS NOT SUITABLE FOR LARGE DATA SETS AS ITS AVERAGE AND WORST-CASE TIME COMPLEXITY IS QUITE HIGH.*
- *TIME COMPLEXITY $O(N^2)$*



ALGORITHM

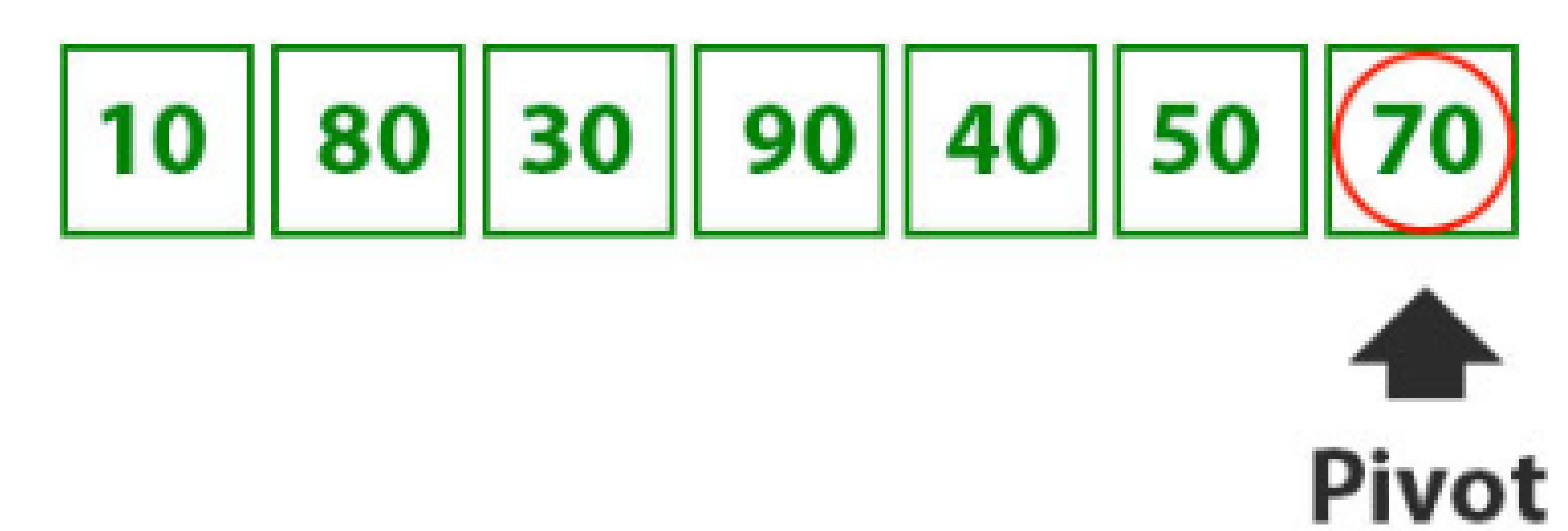
- Run a nested for loop to traverse the input array using two variables i and j, such that $0 \leq i < n-1$ and $0 \leq j < n-i-1$
- If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on
- Print the sorted array

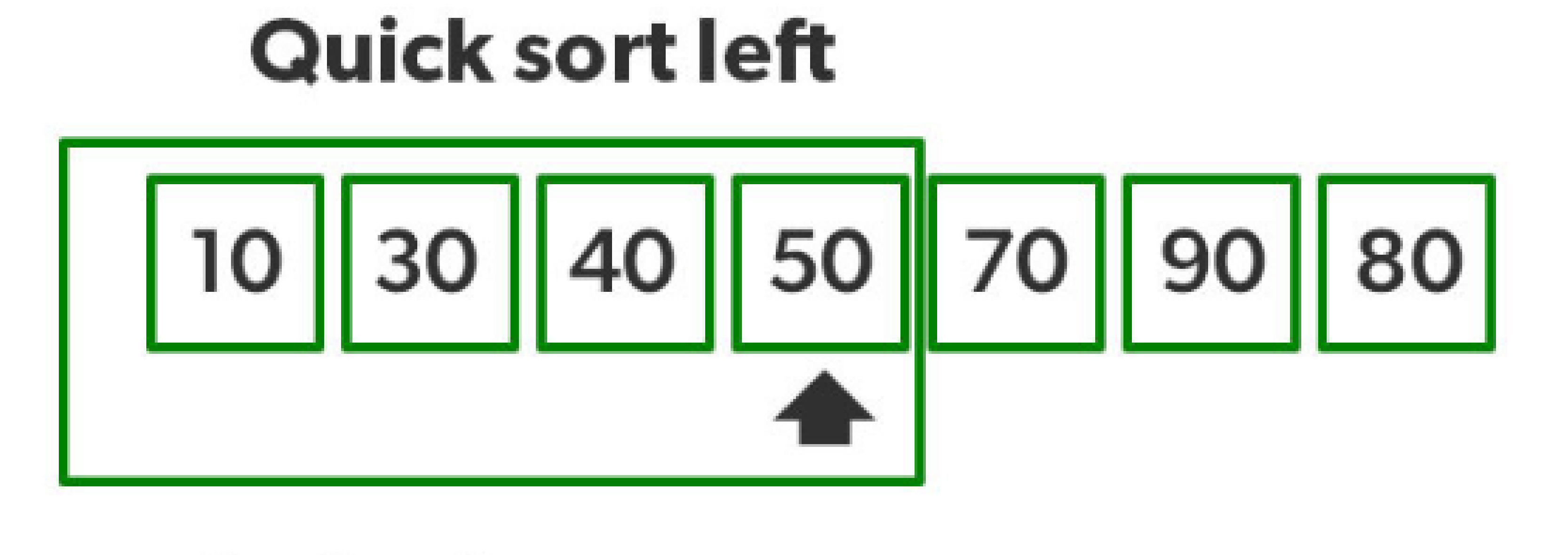
CODE

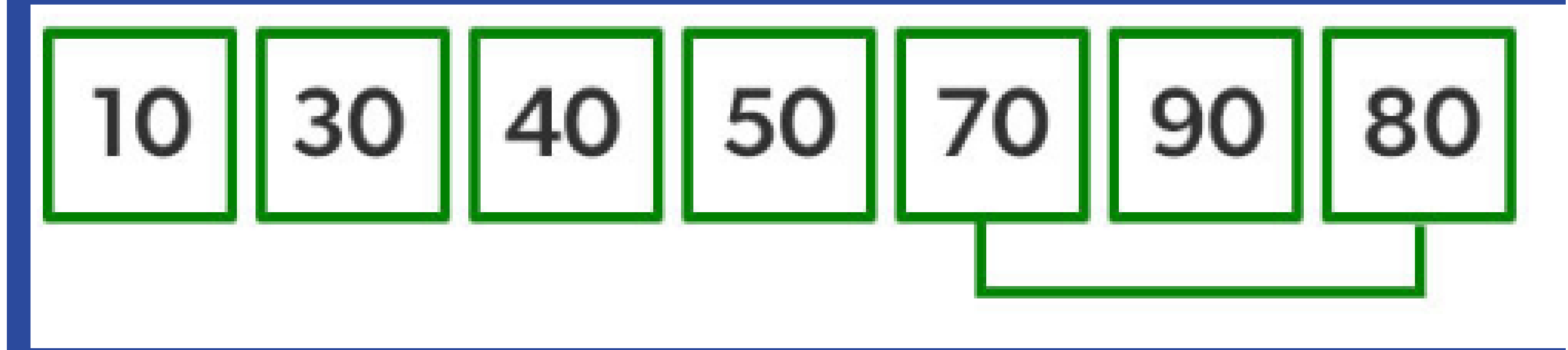
4

QUICK SORT

- QUICKSORT IS A DIVIDE AND CONQUER ALGORITHM.
- IT PICKS AN ELEMENT AS A PIVOT AND PARTITIONS THE GIVEN ARRAY AROUND THE PICKED PIVOT.
- THERE ARE MANY DIFFERENT VERSIONS OF QUICKSORT THAT PICK PIVOT IN DIFFERENT WAYS.
- TIMING COMPLEXITY O(N*LOGN)







PARTITION ALGORITHM

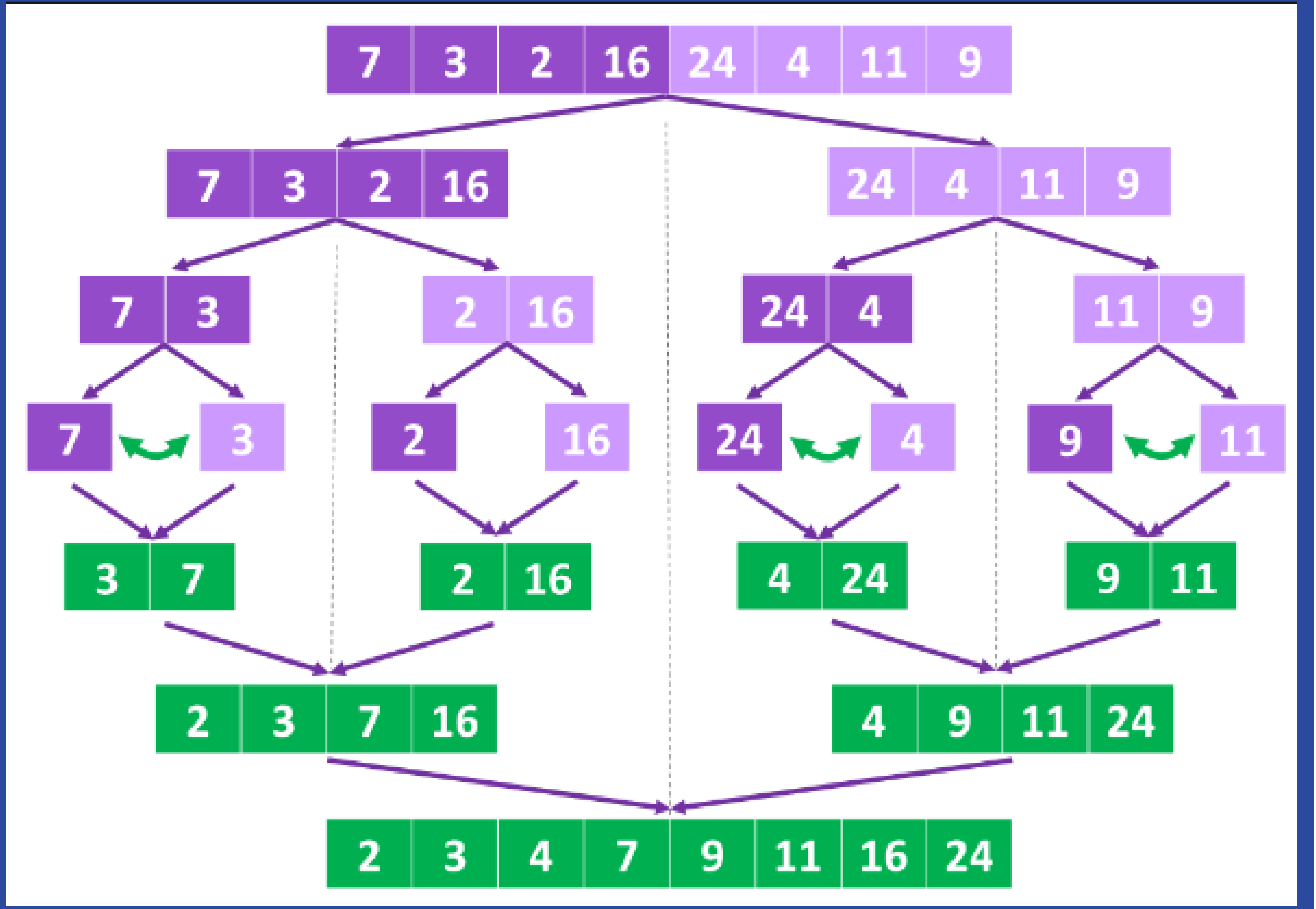
- *The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal to) elements as i.*
- *While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.*

CODE

5

MERGE SORT

- THE MERGE SORT ALGORITHM IS A SORTING ALGORITHM THAT IS BASED ON THE DIVIDE AND CONQUER PARADIGM.
- IN THIS ALGORITHM, THE ARRAY IS INITIALLY DIVIDED INTO TWO EQUAL HALVES AND THEN THEY ARE COMBINED IN A SORTED MANNER.
- TIMING COMPLEXITY O(N*LOGN)



ALGORITHM

- *It as a recursive algorithm continuously splits the array in half until it cannot be further divided.*
- *If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves.*
- *Finally, when both halves are sorted, the merge operation is applied.*
- *Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.*

CODE

COMPARISION – SINGLE CYCLE

1.INSERTION SORT

Execution info

Cycles:	99
Instrs. retired:	99
CPI:	1
IPC:	1
Clock rate:	7.46 Hz

2.SELECTION SORT

Execution info

Cycles:	127
Instrs. retired:	127
CPI:	1
IPC:	1
Clock rate:	10.75 Hz

3. BUBBLE SORT

Execution info

Cycles:	174
Instrs. retired:	174
CPI:	1
IPC:	1
Clock rate:	50.58 Hz

4.QUICK SORT

Execution info

Cycles:	176
Instrs. retired:	176
CPI:	1
IPC:	1
Clock rate:	7.87 Hz

5.MERGE SORT

Execution info

Cycles:	263
Instrs. retired:	263
CPI:	1
IPC:	1
Clock rate:	9.35 Hz

COMPARISION – 5 CYCLE(S)

1.INSERTION SORT

Execution info

Cycles:	133
Instrs. retired:	99
CPI:	1.34
IPC:	0.744
Clock rate:	10.31 Hz

2.SELECTION SORT

Execution info

Cycles:	197
Instrs. retired:	127
CPI:	1.55
IPC:	0.645
Clock rate:	9.26 Hz

3.BUBBLE SORT

Execution info

Cycles:	242
Instrs. retired:	174
CPI:	1.39
IPC:	0.719
Clock rate:	124.61 Hz

4.QUICK SORT

Execution info

Cycles:	249
Instrs. retired:	176
CPI:	1.41
IPC:	0.707
Clock rate:	9.62 Hz

5.MERGE SORT

Execution info

Cycles:	373
Instrs. retired:	263
CPI:	1.42
IPC:	0.705
Clock rate:	271.74 Hz

BUBBLE VS QUICK SORT FOR LARGE DATA SET

BUBBLE SORT

- Compares the current element with the next on n-1 time.
- Relatively less efficient.
- Timing complexity O(n2)

Execution info

Cycles:	760
Instrs. retired:	610
CPI:	1.25
IPC:	0.803
Clock rate:	983.12 Hz

QUICK SORT

- *Compares the pivot with the rest of the elements n-1 times*
- *Relatively more efficient.*
- *Timing complexity O(n*logn)*

Execution info

Cycles:	1206
Instrs. retired:	839
CPI:	1.44
IPC:	0.696
Clock rate:	3.36 KHz

THANK YOU