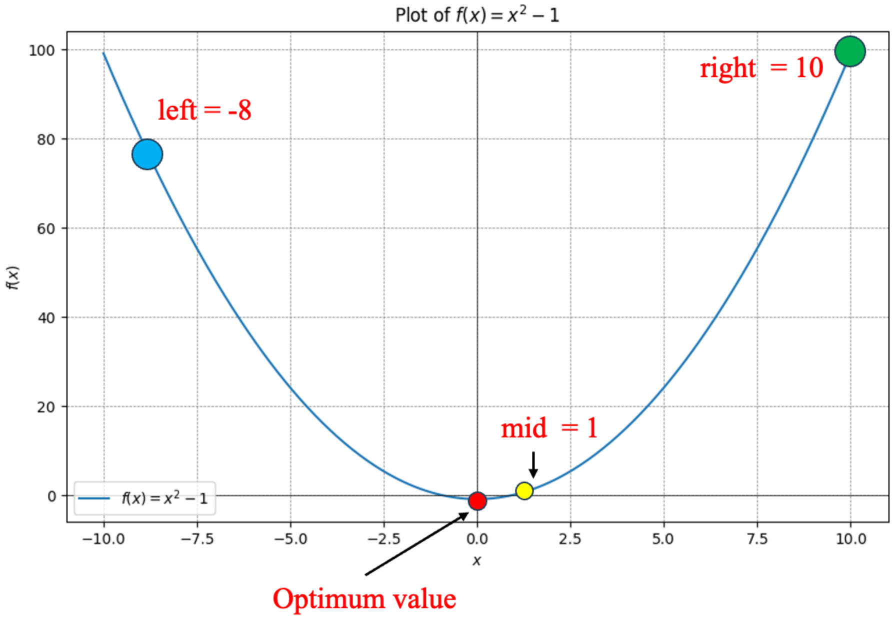
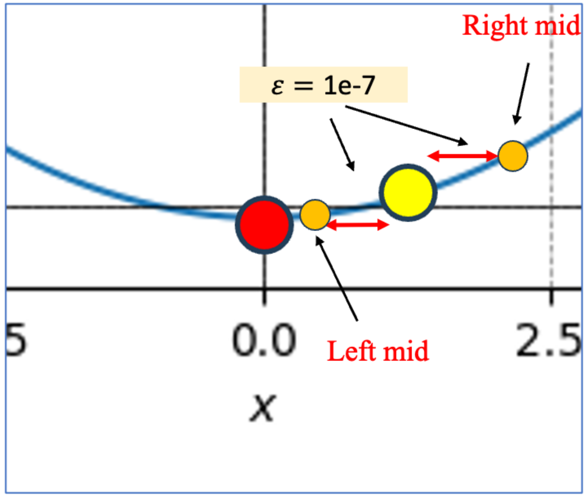
AI VIET NAM
@aivietnam.edu.vn

Algorithms on List

Sort and Search





Vinh Dinh Nguyen
PhD in Computer Science

Outline

- Built-in Function for List
- Bubble Sort Algorithm
- Binary Search Algorithm
- Quiz
- Optimization using Binary Search

Built-in Functions for List

len(), min(), and max()

data =

6	5	7	1	9	2

trả về số phần tử
len(data) = 6

trả về số phần tử có giá trị nhỏ nhất
min(data) = 1

trả về số phần tử có giá trị lớn nhất
max(data) = 9

```
data = [6, 5, 7, 1, 9, 2]
print(data)
```

```
[6, 5, 7, 1, 9, 2]
```

```
# get a number of elements
length = len(data)
print(length)
```

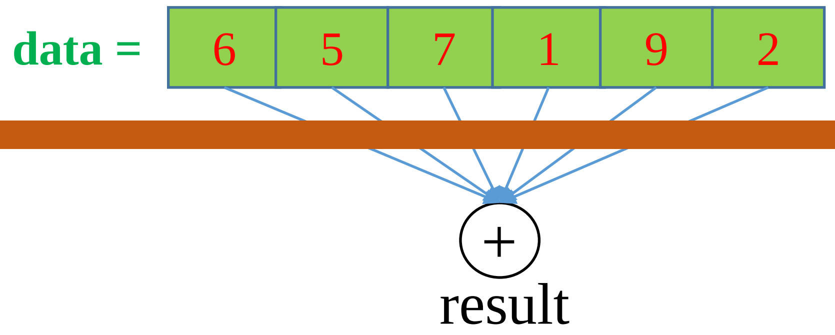
```
6
```

```
# get the min and max values
print(min(data))
print(max(data))
```

```
1
9
```

AI VIET NAM
@aivietnam.edu.vn

Built-in Functions



sum()

$$summation = \sum_{i=0}^{n} data_i$$

data =

6	5	7	1	9	2

tính tổng

sum(data) = 30

```python
data = [6, 5, 7, 1, 9, 2]
print(data)

summation = sum(data)
print(summation)
```

```
[6, 5, 7, 1, 9, 2]
30
```

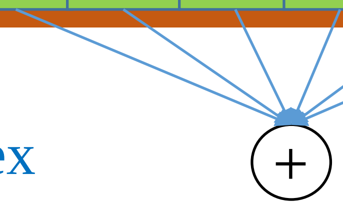
```python
# custom summation - way 1
def computeSummation(data):
    result = 0

    for value in data:
        result = result + value

    return result

# test
data = [6, 5, 7, 1, 9, 2]
summation = computeSummation(data)
print(summation)
```

```
30
```

AI VIET NAM
@aivietnam.edu.vn

Built-in Functions

index	0	1	2	3	4	5
data =	6	5	7	1	9	2



result

sum()

$$summation = \sum_{i=0}^{n} data_i$$

data =	6	5	7	1	9	2

tính tổng
sum(data) = 30

```python
data = [6, 5, 7, 1, 9, 2]
print(data)

summation = sum(data)
print(summation)
```

```
[6, 5, 7, 1, 9, 2]
30
```

Using index

```python
# custom summation - way 2
def computeSummation(data):
    result = 0

    length = len(data)
    for index in range(length):
        result = result + data[index]

    return result

# test
data = [6, 5, 7, 1, 9, 2]
summation = computeSummation(data)
print(summation)
```

```
30
```

AI VIET NAM
@aivietnam.edu.vn

Examples

Sum of even numbers

data =

6	5	7	1	9	2

```python
# sum of even number
def sum1(data):
    result = 0

    for value in data:
        if value%2 == 0:
            result = result + value

    return result

# test
data = [6, 5, 7, 1, 9, 2]
summation = sum1(data)
print(summation)
```

Sum of elements with even indices

data =

6	5	7	1	9	2

```python
# sum of numbers with even indices
def sum2(data):
    result = 0

    length = len(data)
    for index in range(length):
        if index%2 == 0:
            result = result + data[index]

    return result

# test
data = [6, 5, 7, 1, 9, 2]
summation = sum2(data)
print(summation)
```

Examples

square(aList)

data =	6	5	7	1	9	2
square(data) =	36	25	49	1	81	4

```python
# square function
def square(data):
    result = []

    for value in data:
        result.append(value*value)

    return result

# test
data = [6, 5, 7, 1, 9, 2]
print(data)
data_s = square(data)
print(data_s)
```

```
[6, 5, 7, 1, 9, 2]
[36, 25, 49, 1, 81, 4]
```

AI VIET NAM
@aivietnam.edu.vn

List Sorting

```python
data = [6, 5, 7, 1, 9, 2]
print(data)
data.sort()
print(data)
```

```
[6, 5, 7, 1, 9, 2]
[1, 2, 5, 6, 7, 9]
```

```python
data = [6, 5, 7, 1, 9, 2]
print(data)
data.sort(reverse = True)
print(data)
```

```
[6, 5, 7, 1, 9, 2]
[9, 7, 6, 5, 2, 1]
```

```python
# sorted
data = [6, 5, 7, 1, 9, 2]
print(data)

sorted_data = sorted(data)
print(sorted_data)
```

```
[6, 5, 7, 1, 9, 2]
[1, 2, 5, 6, 7, 9]
```

```python
# sorted
data = [6, 5, 7, 1, 9, 2]
print(data)

sorted_data = sorted(data, reverse=True)
print(sorted_data)
```

```
[6, 5, 7, 1, 9, 2]
[9, 7, 6, 5, 2, 1]
```

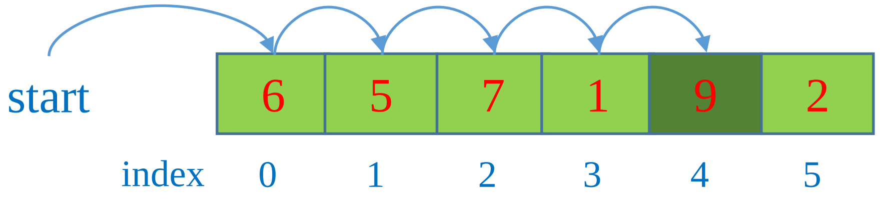
Algorithms on List

❖ Linear searching

data =

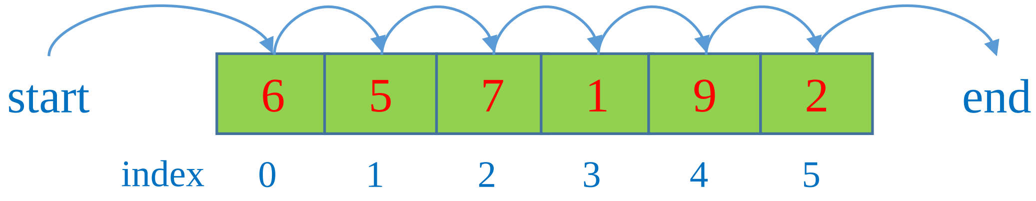
	6	5	7	1	9	2
index	0	1	2	3	4	5

Searching for 9

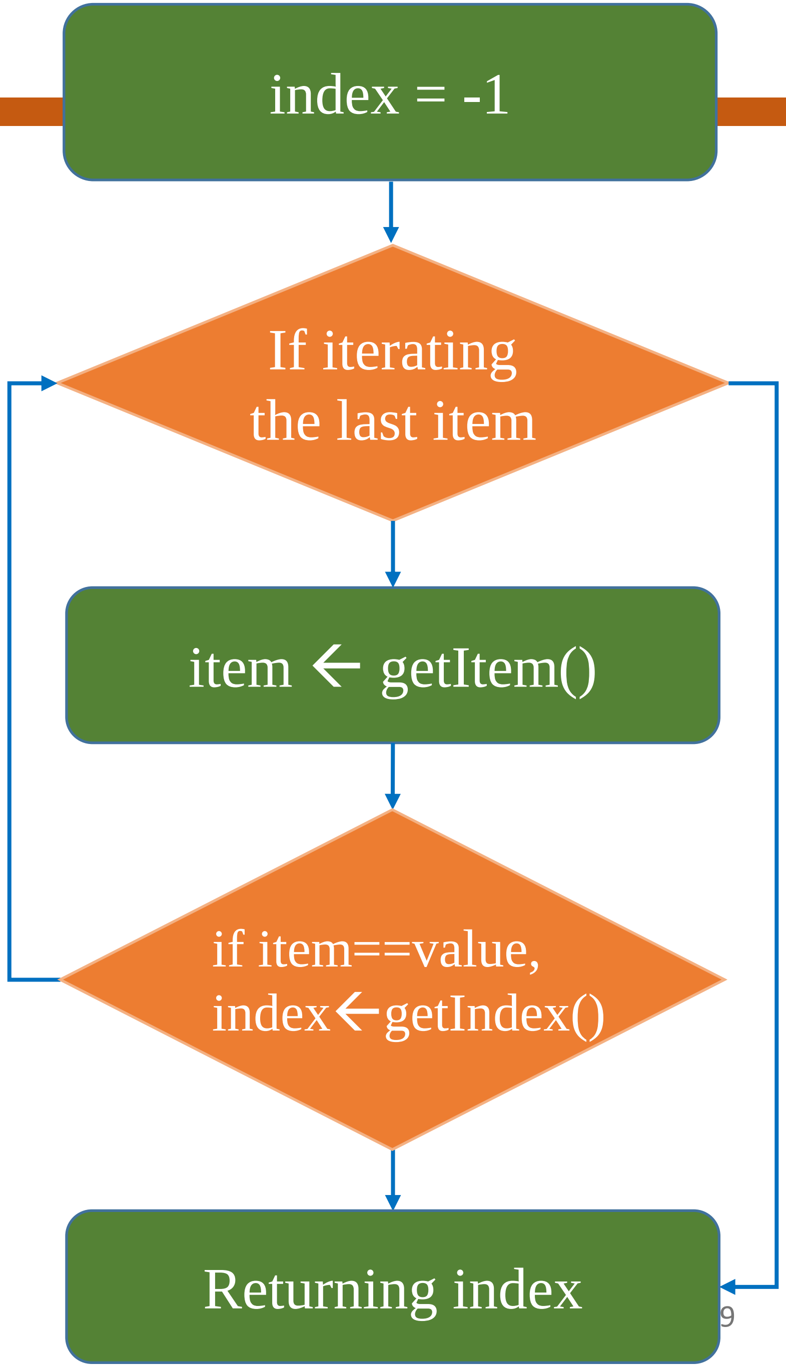


Returning 4

Searching for 8



Returning ?



AI VIET NAM
@aivietnam.edu.vn

Algorithms on List

❖ Linear searching

Element found

Element not found

Algorithms on List

❖ **Sorting using min(), remove(), and append()**

data = | 6 | 5 | 7 | 1 | 9 | 2 |

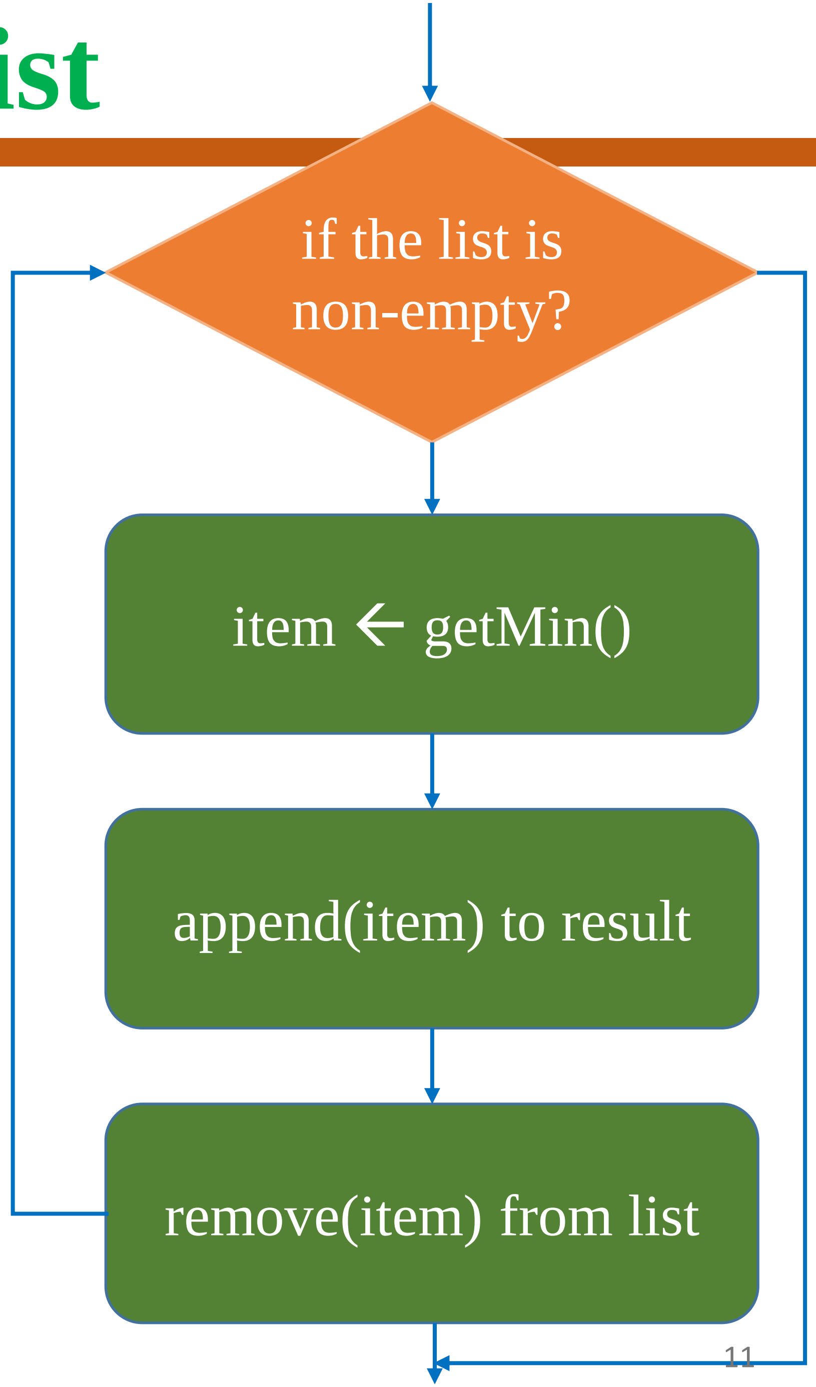
result = [ ]

min(data) = 1

result.append(1) = | 1 |

data.remove(1) = | 6 | 5 | 7 | 9 | 2 |

. . .



Algorithms on List

❖ **Sorting using min(), remove(), and append()**

data = | 6 | 5 | 7 | 9 | 2 |

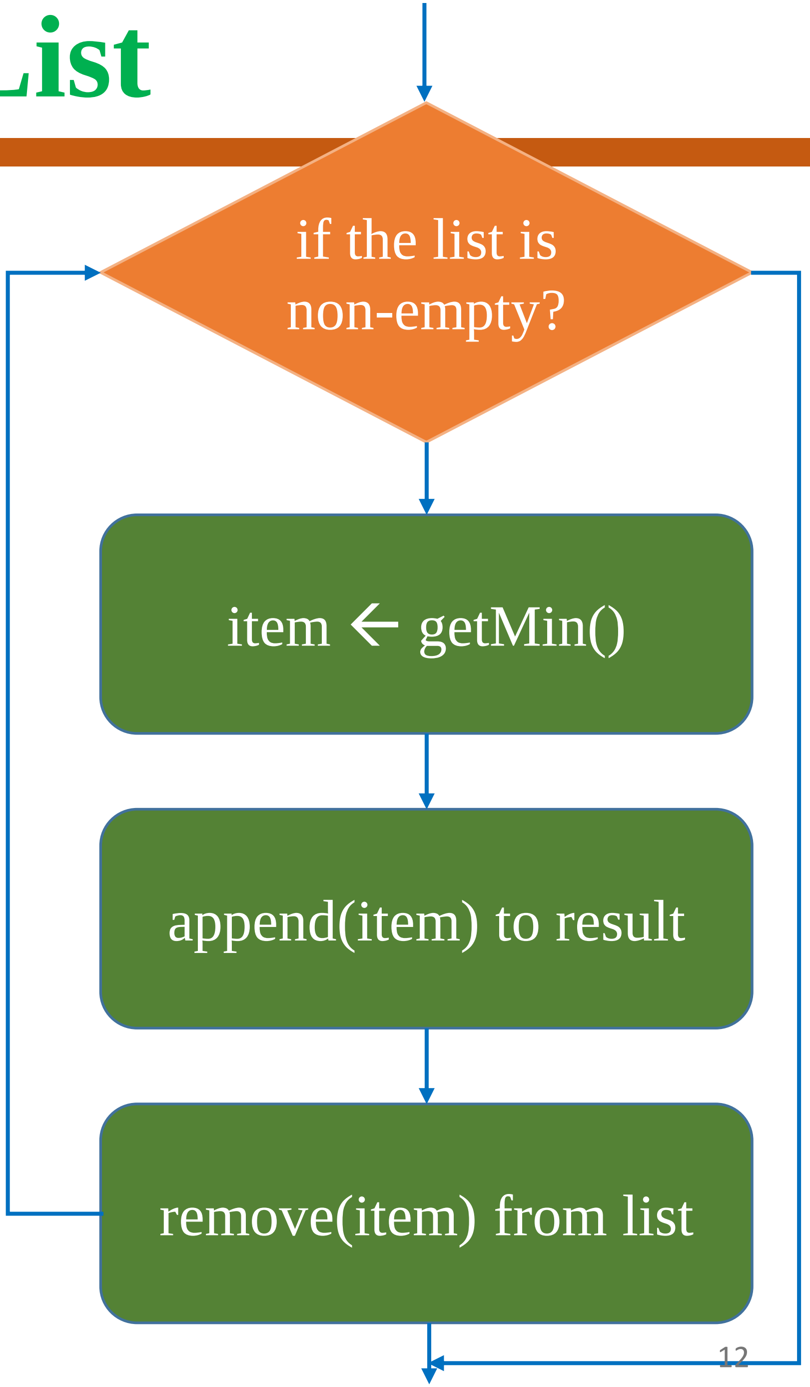
result = | 1 |

min(data) = 2

result.append(2) = | 1 | 2 |

data.remove(2) = | 6 | 5 | 7 | 9 |

. . .



Sorting

item=7

```
sorted(iterable, key=None, reverse=False)
```

```python
# create a list
aList = [1, 5, 3, 7, 4]
print(aList)

# sort
sortedList = sorted(aList)
print(sortedList)
```

```
[1, 5, 3, 7, 4]
[1, 3, 4, 5, 7]
```

```python
# create a list
aList = [1, 5, 3, 7, 4]
print(aList)

# function for sorting
def compare(item):
    return item

# sort
sortedList = sorted(aList, key=compare)
print(sortedList)
```

```
[1, 5, 3, 7, 4]
[1, 3, 4, 5, 7]
```

Sorting

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3)
print(list4)
```

```
[('a', 16), ('b', 15), ('e', 18), ('g', 13), ('h', 11)]
```

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# function for sorting
def compare(item):
    return item[0]

# create
list3 = list(zip(list1, list2))
print(list3)
```

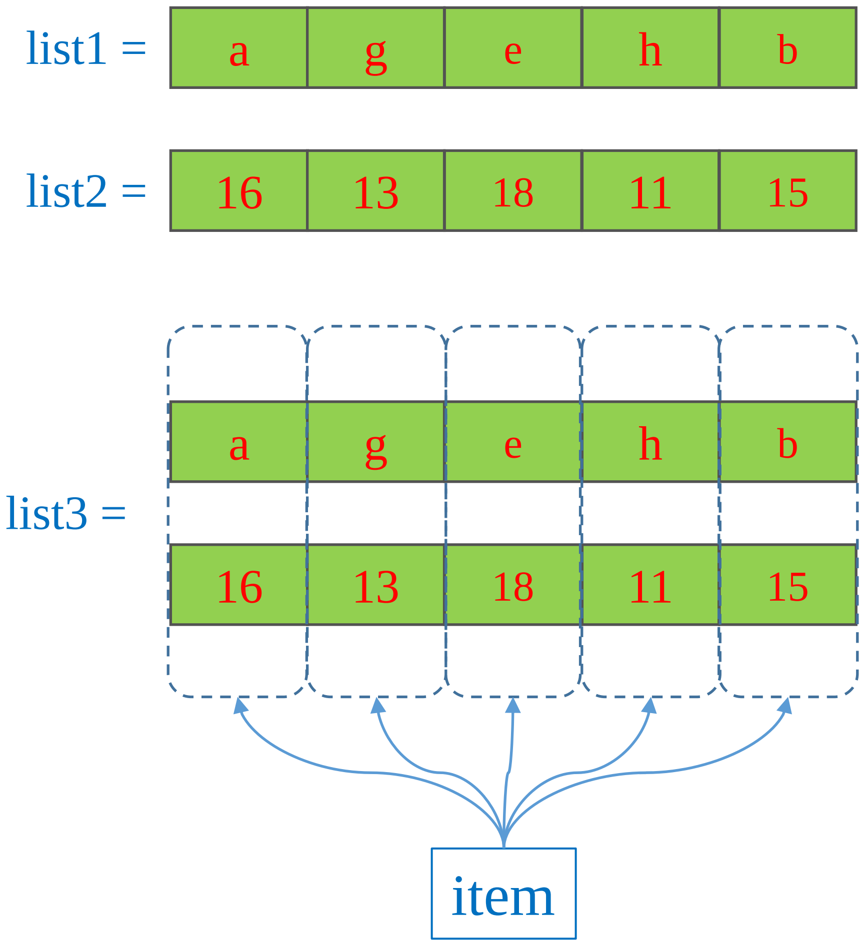
```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3, key=compare)
print(list4)
```

```
[('a', 16), ('b', 15), ('e', 18), ('g', 13), ('h', 11)]
```

AI VIET NAM
@aivietnam.edu.vn

Sorting




```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# function for sorting
def compare(item):
    return item[1]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3, key=compare)
print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```

Lambda function

- Take any number of arguments
- Can only have one expression

Syntax

lambda arguments : expression

```python
# lambda function
a_lfunction = lambda v: v + 10
print(a_lfunction(5))
```

```
15
```

```python
# lambda function
a_lfunction = lambda v1, v2: v1+v2
print(a_lfunction(3, 4))
```

```
7
```

Sorting

Using lambda function

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3, key=lambda item: item[1])
print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```

item=('g', 13)

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# function for sorting
def compare(item):
    return item[1]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

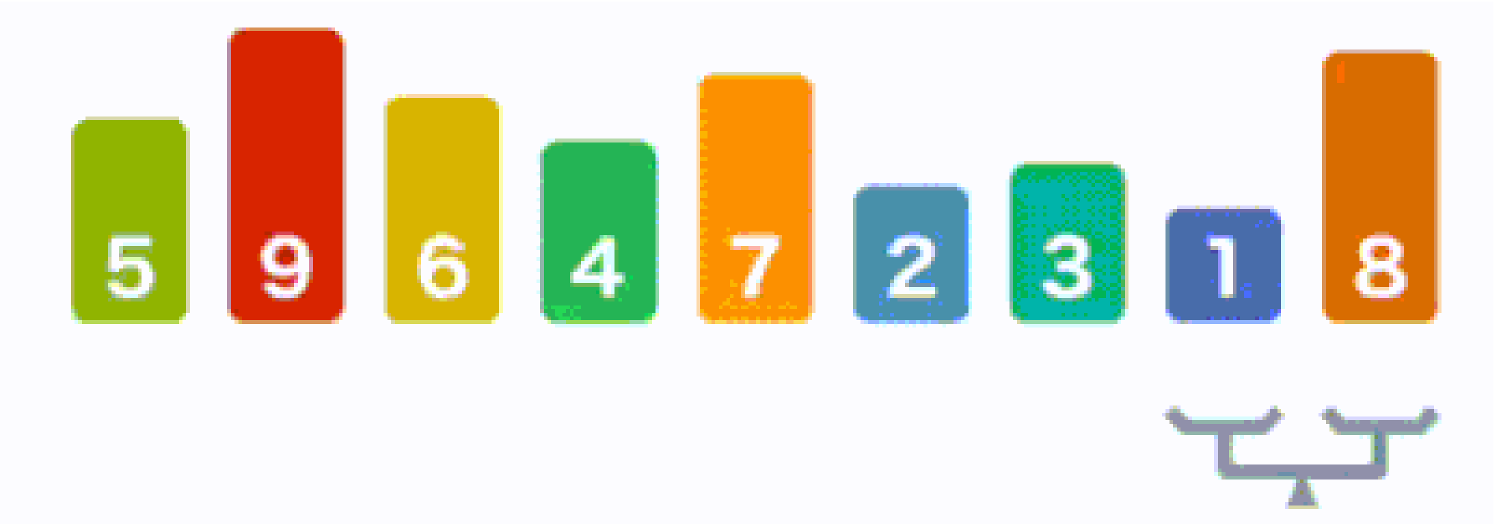
```python
list4 = sorted(list3, key=compare)
print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```

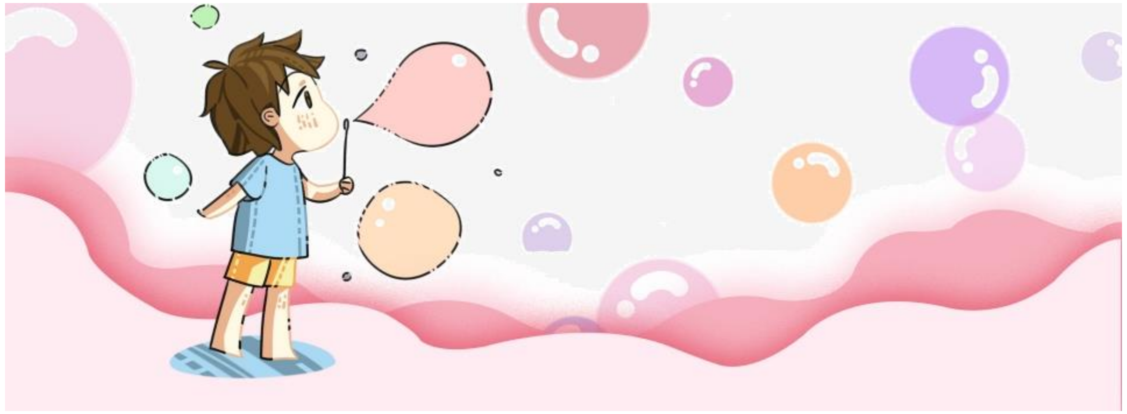
Outline

- Built-in Function for List
- Bubble Sort Algorithm
- Binary Search Algorithm
- Quiz
- Optimization using Binary Search

Bubble Sort Algorithm

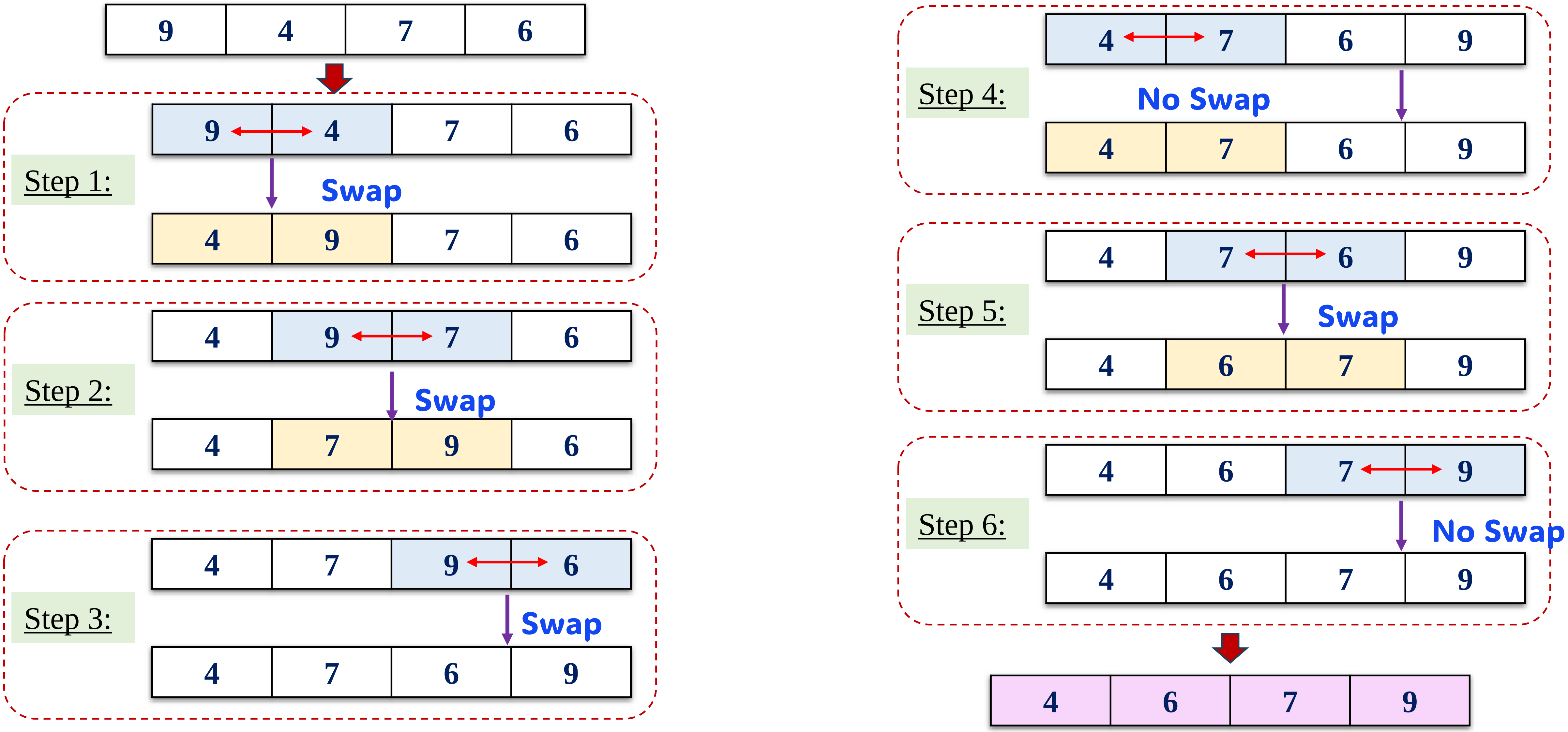


Bubble sort is a comparison-based sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order. As the algorithm progresses, smaller elements “bubble” to the top of the list or array, eventually resulting in a sorted sequence.

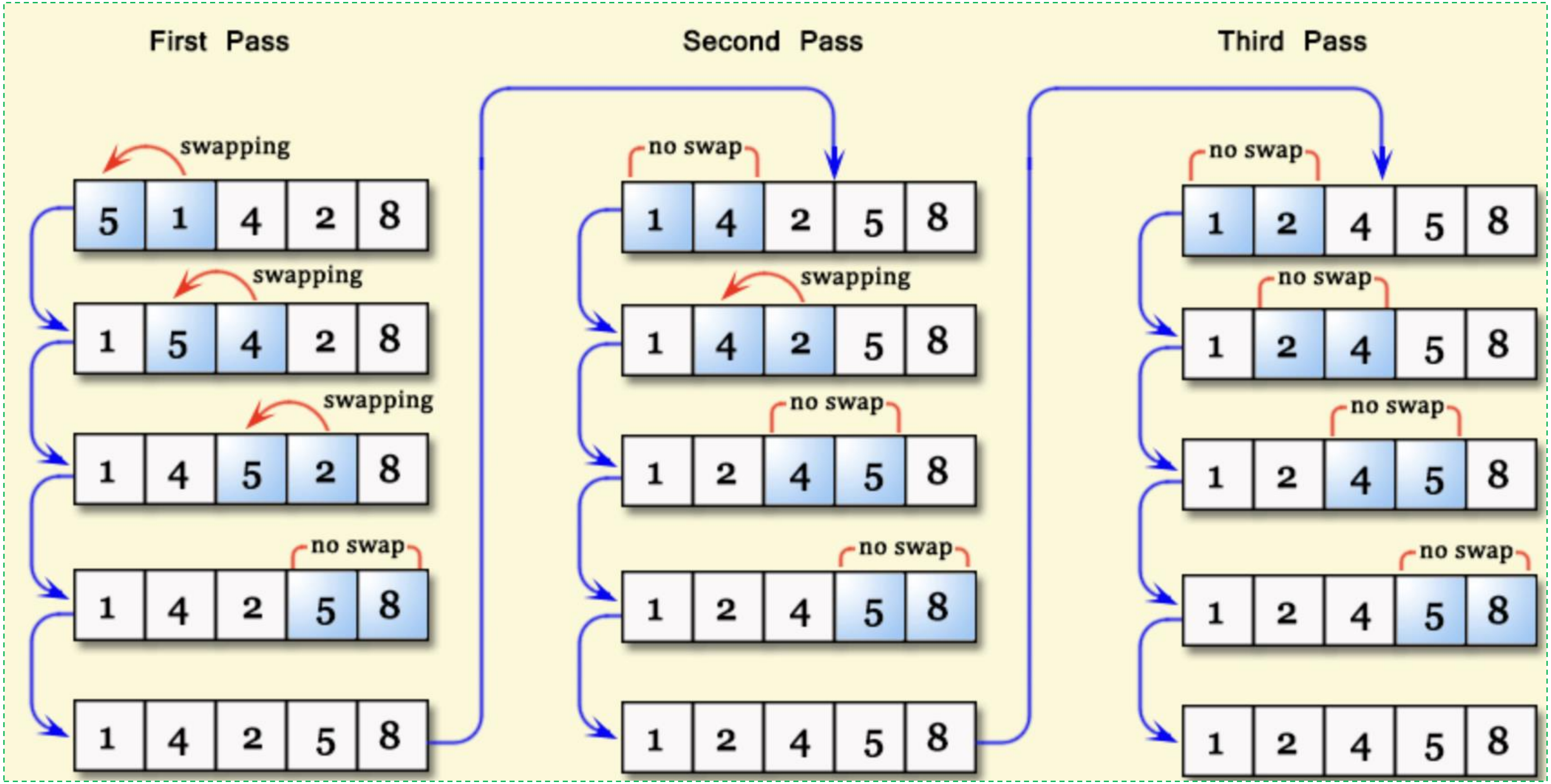


AI VIET NAM
@aivietnam.edu.vn

Bubble Sort Algorithm



Bubble Sort Algorithm



AI VIET NAM
@aivietnam.edu.vn

Bubble Sort Algorithm

❖This implementation is not optimal. Why?

Version 1

```
def bubbleSort(arr):
    n = len(arr)
    step = 1
    for i in range(n-1):
        for j in range(n-1):
            print("step: ", step)
            step = step + 1
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

```
array = [4,6, 7, 9]
bubbleSort(array)
print(array)
```

```
step:  1
step:  2
step:  3
step:  4
step:  5
step:  6
[4, 6, 7, 9]
```

AI VIET NAM
@aivietnam.edu.vn

Bubble Sort Algorithm

❖This implementation is not optimal. Why?

Version 2

```
def bubbleSort(arr):
    n = len(arr)
    step = 1
    for i in range(n-1):
        for j in range(0, n-i-1):
            print("step: ", step)
            step = step + 1
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

```
array = [4,6, 7, 9]
bubbleSort(array)
print(array)
```

```
step:  1
step:  2
step:  3
step:  4
step:  5
step:  6
[4, 6, 7, 9]
```

Bubble Sort Algorithm

❖ **This is a better solution. Why?**

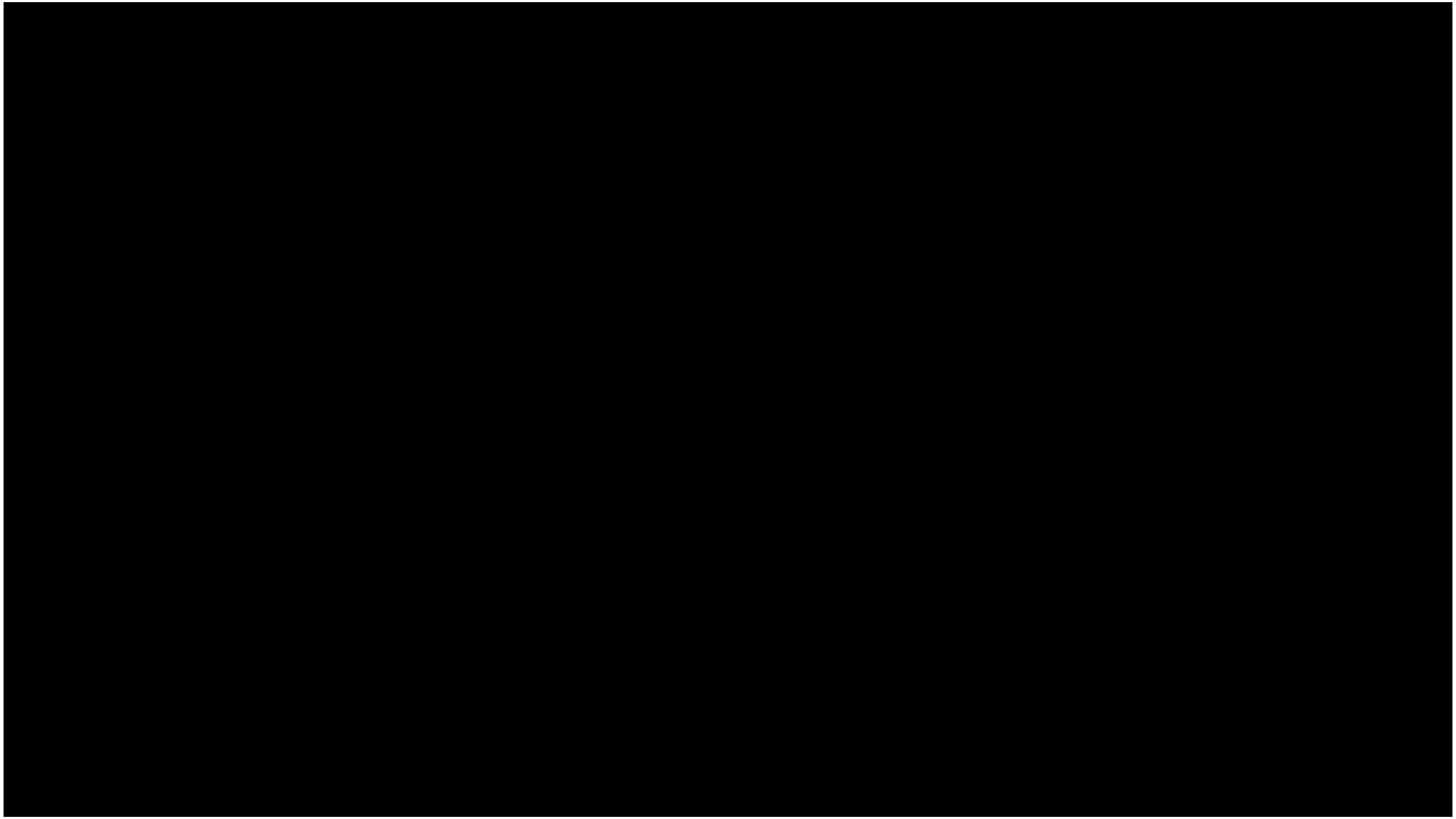
Version 3

```
def bubbleSort_optimize(arr):
    n = len(arr)
    step = 1
    for i in range(n-1):
        swapped = False
        for j in range(0, n-i-1):
            print("step: ", step)
            step = step + 1
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if not swapped:
            return
```

```
array = [4,6, 7, 9]
bubbleSort_optimize(array)
print(array)
```

```
step:  1
step:  2
step:  3
[4, 6, 7, 9]
```

Bubble Sort: Example



AI VIET NAM
@aivietnam.edu.vn

Bubble Sort Vs. Others

https://www.toptal.com/developers/sorting-algorithms

AI VIET NAM
@aivietnam.edu.vn

Time Complexity

Sorting algorithm	Time Worst case	Time Best case	Space complexity
Bubble sort	O(N^2)	O(N)	O(1)
Insertion sort	O(N^2)	O(N)	O(1)
Selection sort	O(N^2)	O(N^2)	O(1)
Merge sort	O(N log N)	O(N log N)	O(N)
Heap sort	O(N log N)	O(N log N)	O(1)
Quick sort	O(N^2)	O(N log N)	O(N log N)
Counting sort	O(N^2)	O(N)	O(N)
Radix sort	O(N)	O(N)	O(N)
Bucket sort	O(N^2)	O(N)	O(N)

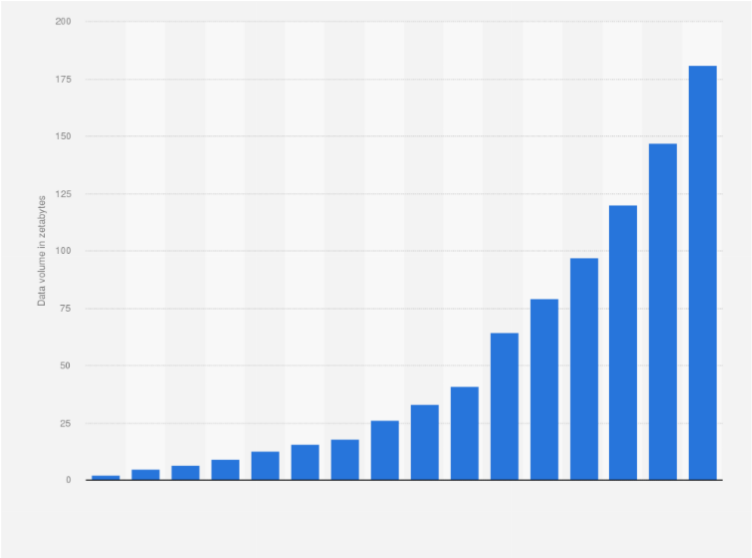
What is big ”O” notation?

AI VIET NAM
@aivietnam.edu.vn

What is Big O Notation

How code slows as data grows

1. Describe the performance of an algorithm as the amount of data increases
2. Machine independent (# of steps to completion)
3. Ignore smaller operations: O(n+1) => O(n)

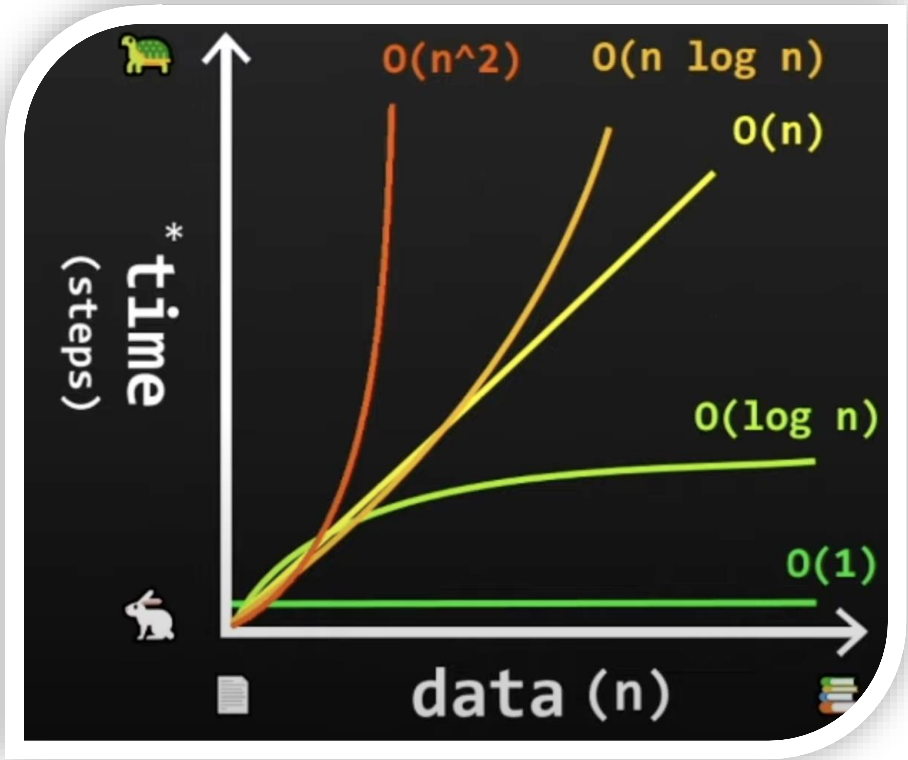


O(1)
O(n)
O(log n)
O(n^2)

n = amount of data

AI VIET NAM
@aivietnam.edu.vn

What is Big O Notation



O(1) - constant time:
An example of an operation that runs in constant time is random access of an element in an array.

O(log n) - logarithmic time:
An example of an operation that runs in logarithmic time is binary search.

O(n) - linear time:
An example of an operation that runs in linear time is looping through elements in an array.

O(n log n) - quasilinear time:
Examples of operations that run in quasilinear time include quicksort, mergesort, and heapsort.

O(n^2) - quadratic time:
Examples of operations that run in quadratic time include insertion sort, selection sort, and bubble sort.

What is Big O Notation

O(n) linear time

```python
def addUp(n):
    sum = 0
    for i in range(1, n+1):
        print("step ", i)
        sum = sum + i
    return sum
```

```python
n = 10
print(addUp(n))
```

```
step  1
step  2
step  3
step  4
step  5
step  6
step  7
step  8
step  9
step  10
55
```

n = 100000 => 100000 steps

sum = 1 + 2 + 3 + …+ (n-1) + n

O(1) constant time

```python
def addUp(n):
    sum = n + 1
    sum = sum  * n
    sum = sum / 2
    return sum
    # return n * (n+1) / 2
```

```python
n = 10
print(addUp(n))
```

n = 100000 => 3 steps

sum = n(n+1)/2

AI VIET NAM
@aivietnam.edu.vn

Time Complexity Bubble Sort

Best Case Time Complexity Analysis of Bubble Sort: O(N)

```
def bubbleSort_optimize(arr):
    n = len(arr)
    step = 1
    for i in range(n-1):
        swapped = False
        for j in range(0, n-i-1):
            print("step: ", step)
            step = step + 1
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if not swapped:
            return
```

Outer loop

Inner loop 6

- The best case occurs when the array is already sorted.
- So the number of comparisons required is N-1 and the number of swaps required = 0.
- Hence the best case complexity is **O(N)**.

3	4	5	6

3	4	5	6

3	4	5	6

3	4	5	6

Number of comparisons = N – 1

AI VIET NAM
@aivietnam.edu.vn

Time Complexity Bubble Sort

Worst Case Time Complexity Analysis of Bubble Sort: O(N^2)

The worst-case condition for bubble sort occurs when elements of the array are arranged in decreasing order. In the worst case, the total number of iterations or passes required to sort a given array is **(N-1).** where 'N' is the number of elements present in the array.

```
def bubbleSort_optimize(arr):
    n = len(arr)
    step = 1
    for i in range(n-1):
        swapped = False
        for j in range(0, n-i-1):
            print("step: ", step)
            step = step + 1
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if not swapped:
            return
```

Outer loop

Inner loop

- **At pass 1:** (N-1) Number of swaps and comparisons
- **At pass 2:** (N-2) Number of swaps and comparisons
- **At pass N-1:** 1 Number of swaps and comparisons

Total [swaps + comparison]: (N-1) + (N-2) + (N-3) + . . . 2 + 1

Total [comparisons + swap] = N(N-1)/2

$$1 + 2 + 3 + 4 + \ldots\ldots\ldots + n = \frac{n(n+1)}{2}$$

https://byjus.com/maths/arithmetic-series-sum-n-terms/

Time Complexity Bubble Sort

At 1st iteration

5	4	3	2
5	4	3	2
4	5	3	2
4	3	5	2
4	3	2	5

3 (comparisons + swaps)

At 2nd iteration

4	3	2	5
4	3	2	5
3	4	2	5
3	2	4	5

2 (comparisons + swaps-

At 3nd iteration

3	2	4	5
3	2	4	5
2	3	4	5

1 (comparison + swap)

At 1st iteration	4	3	2	5
At 2nd iteration	3	2	4	5
At 3nd iteration 1	2	3	4	5

6 (comparisons + swaps) = 4 (elements) x 3 (iterations) x 1/2
= n x (n-1) x 1/2 = $\frac{1}{2}(n^2 - n)$

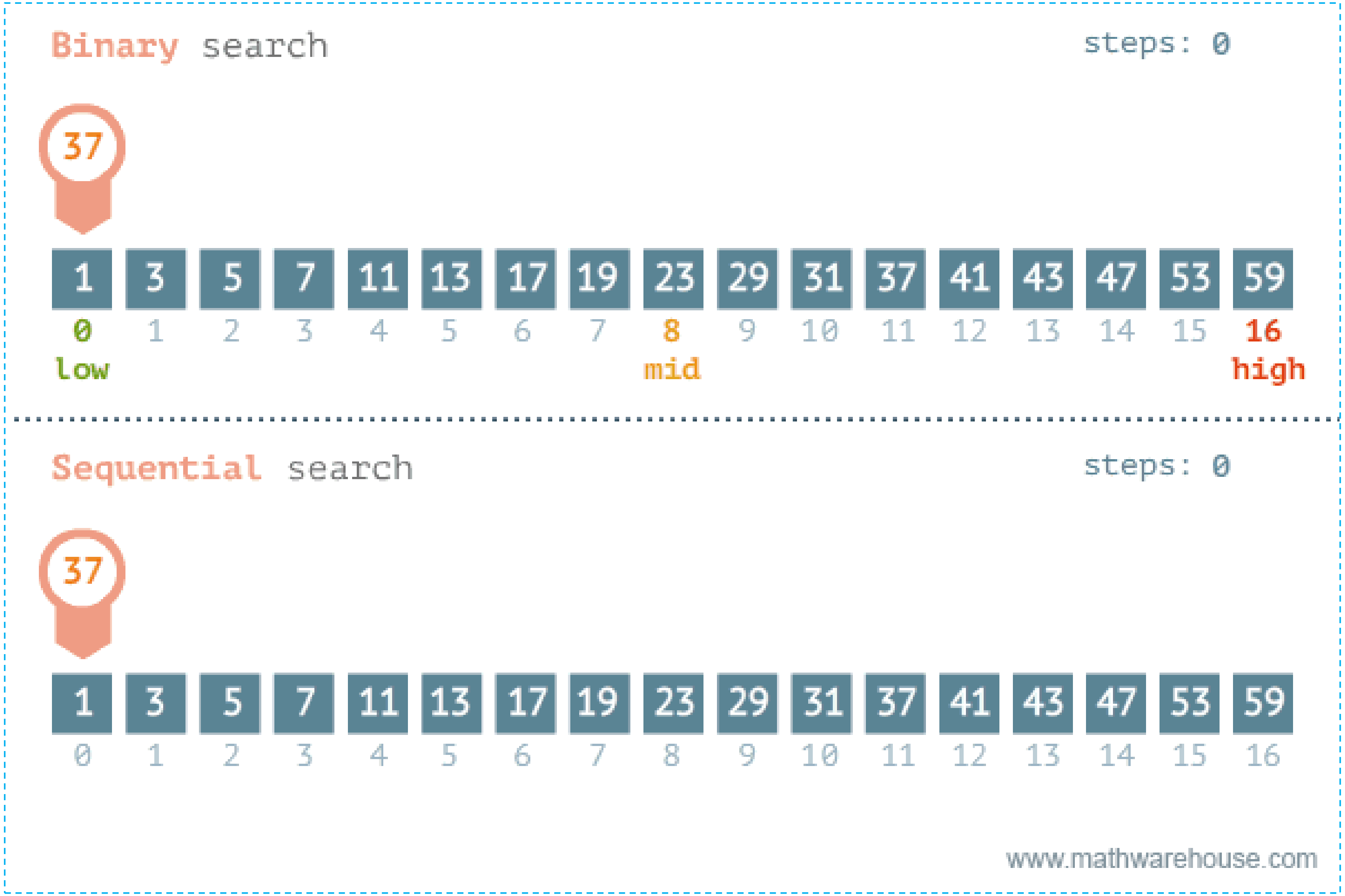
we focus on the term that grows the fastest as n increases and ignore constant factors and lower-order terms

Outline

- Built-in Function for List
- Bubble Sort Algorithm
- Binary Search Algorithm
- Quiz
- Optimization using Binary Search

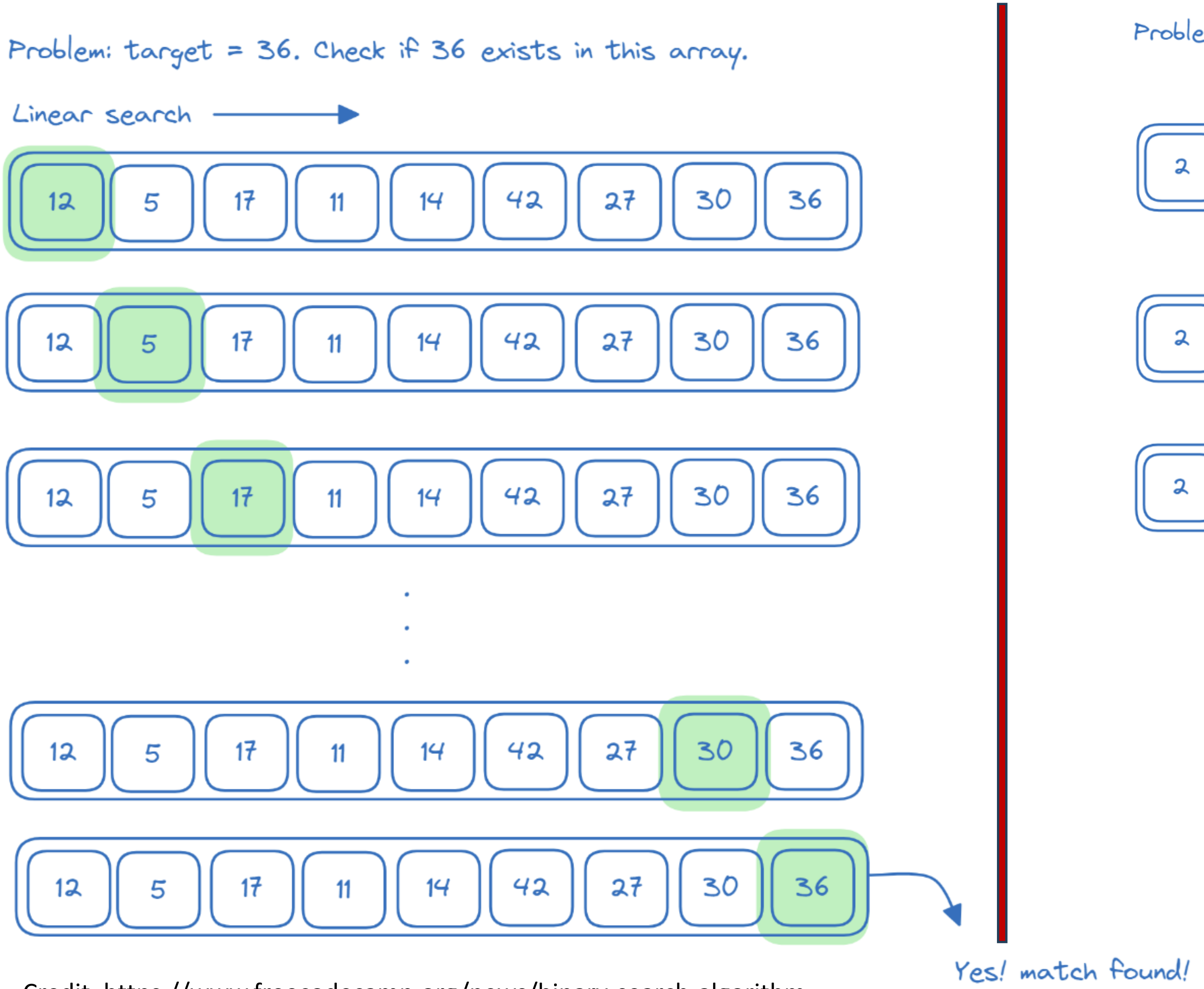
AI VIET NAM
@aivietnam.edu.vn

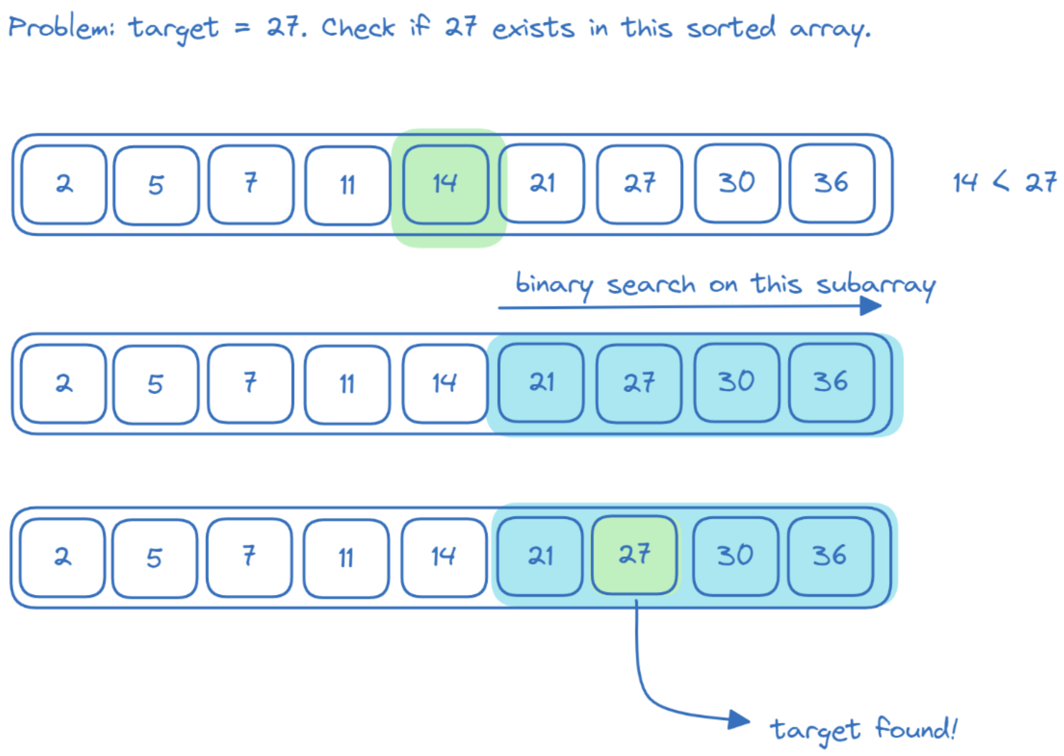
Binary Vs. Linear Search Algorithm

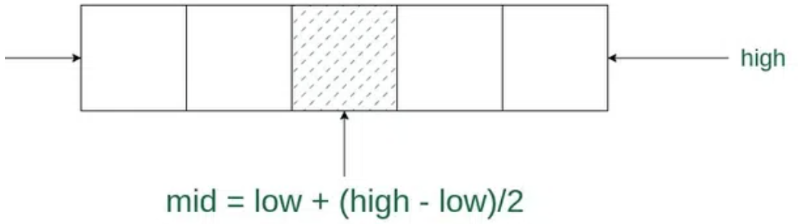


AI VIET NAM
@aivietnam.edu.vn

Binary Vs. Linear Search Algorithm







Credit: https://www.freecodecamp.org/news/binary-search-algorithm-and-time-complexity-explained/

Divide the search space into two halves by finding the middle index "mid".

Binary Search Algorithm

Assuming that the list was sorted!

Given a list of numbers, write a program to check if a specific number exists in the list or not. Search value = 9

Looking for the middle

1	2	3	4	5	6	7	8	9	10	11

Is it equal to "9"?

1	2	3	4	5	6	7	8	9	10	11

Is is "less than 9"?

1	2	3	4	5	6	7	8	9	10	11

Yes, we just need to search in this direction

7	8	9	10	11

Looking for the middle

Is it equal to "9"?

7	8	9	10	11

The index location is 8

AI VIET NAM
@aivietnam.edu.vn

Binary Search Algorithm

```
def binary_search(list_data, search_value):
  low = 0
  high = len(list_data) - 1
  while low <= high:
    mid = (low + high) // 2
    if list_data[mid] == search_value:
      return mid
    elif list_data[mid] < search_value:
      low = mid + 1
    else:
      high = mid - 1
  return -1

numbers = [2, 3, 4, 10, 40]
search_value = 10
result = binary_search(numbers, search_value)
if result != -1:
  print("Element is present at index", result)
else:
  print("Element is not present in array")
```

```
Element is present at index 3
```

Division (floor): divides the first operand by the second

Iterative solution

Search value = 10

low =0 high = 4

2	3	4	10	40

middle = 2 high = 4

2	3	4	10	40

low = 3

			10	40

middle = 3

			10	40

The index location is 3

AI VIET NAM
@aivietnam.edu.vn

Binary Search Algorithm

Recursive solution

```
def binarySearchRecursive(list_data, low, high, search_value):
    if high >= low:

        mid = low + (high - low) // 2

        if list_data[mid] == search_value:
            return mid

        elif list_data[mid] > search_value:
            return binarySearchRecursive(list_data, low, mid-1, search_value)

        else:
            return binarySearchRecursive(list_data, mid + 1, high, search_value)

    else:
        return -1
```

Search value = 10

low =0 high = 4

2	3	4	10	40

middle = 2 high = 4

2	3	4	10	40

low = 3

			10	40

middle = 3

			10	40

The index location is 3

Time Complexity

Search Algorithm	Worst Case	Best Case	Space complexity
Linear Search	O(N)	O(1)	O(1)
Binary Search	O(log N)	O(1)	O(1)
Jump Search	O(√N)	O(1)	O(1)
Interpolation Search	O(N)	O(1)	O(1)
Exponential Search	O(log N)	O(1)	O(1)
Fibonacci Search	O(log N)	O(1)	O(1)

Time Complexity

After k iterations, the size of the array becomes 1 (narrowed down to the first element or last element only).

Length of array = $\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

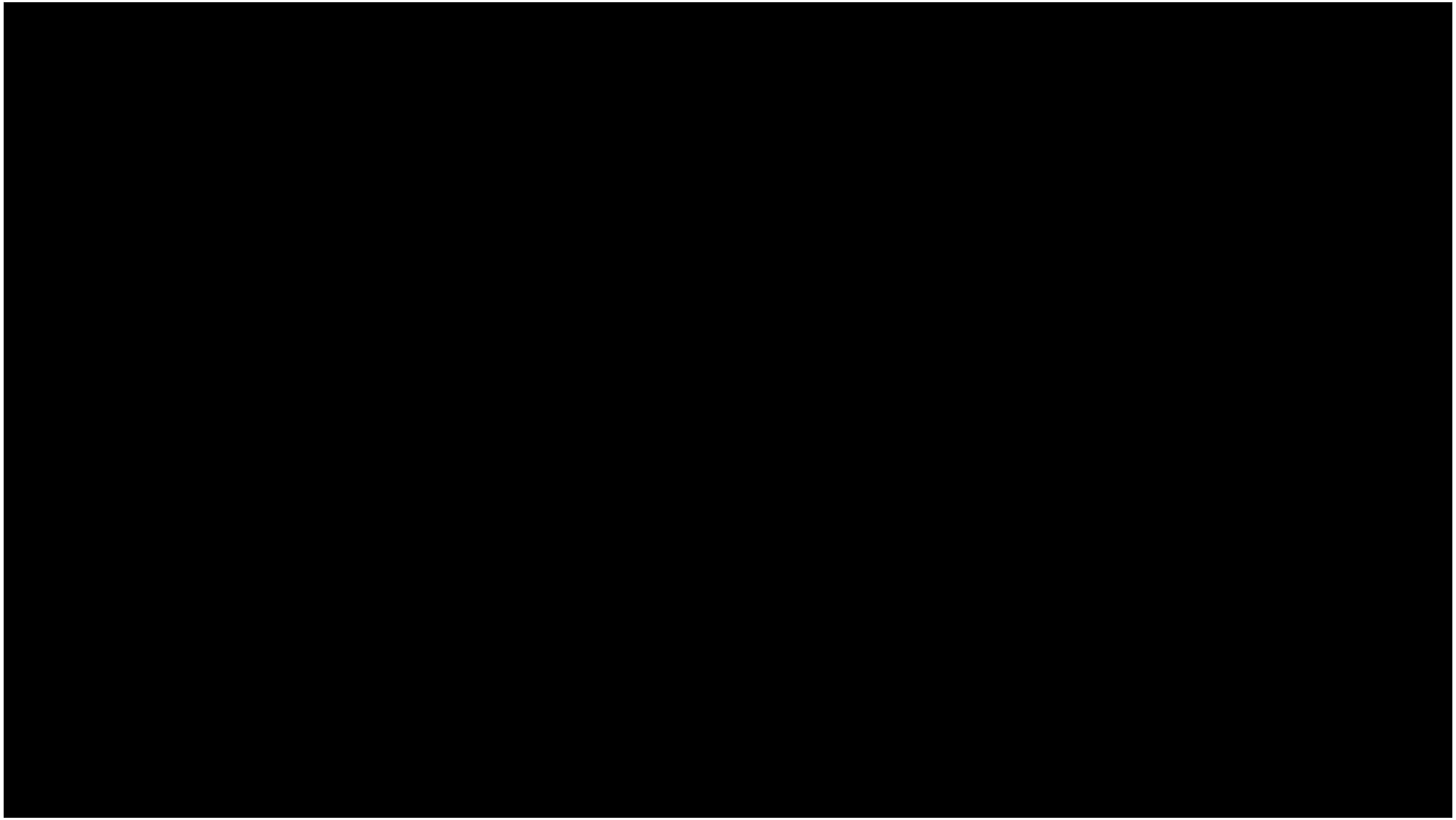
Applying log function on both sides:

$\Rightarrow \log_2(n) = \log_2(2^k)$

$\Rightarrow \log_2(n) = k * \log_2(2) = k$

$\Rightarrow k = \log_2(n)$

Binary Search: Demo



Outline

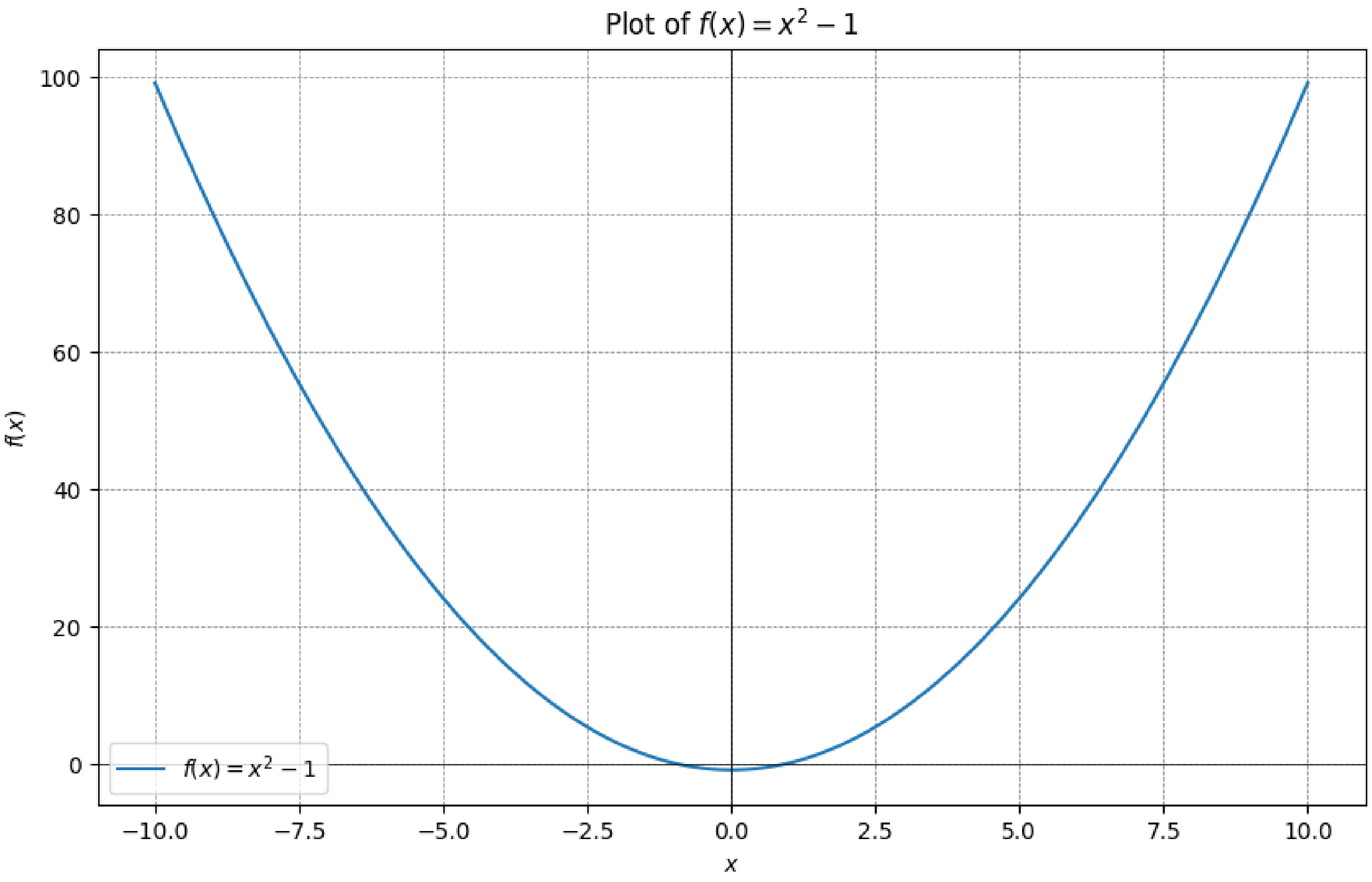
- Built-in Function for List
- Bubble Sort Algorithm
- Binary Search Algorithm
- Quiz
- Optimization using Binary Search

Outline

- Built-in Function for List
- Bubble Sort Algorithm
- Binary Search Algorithm
- Quiz

- Optimization using Binary Search

Optimization using Binary Search

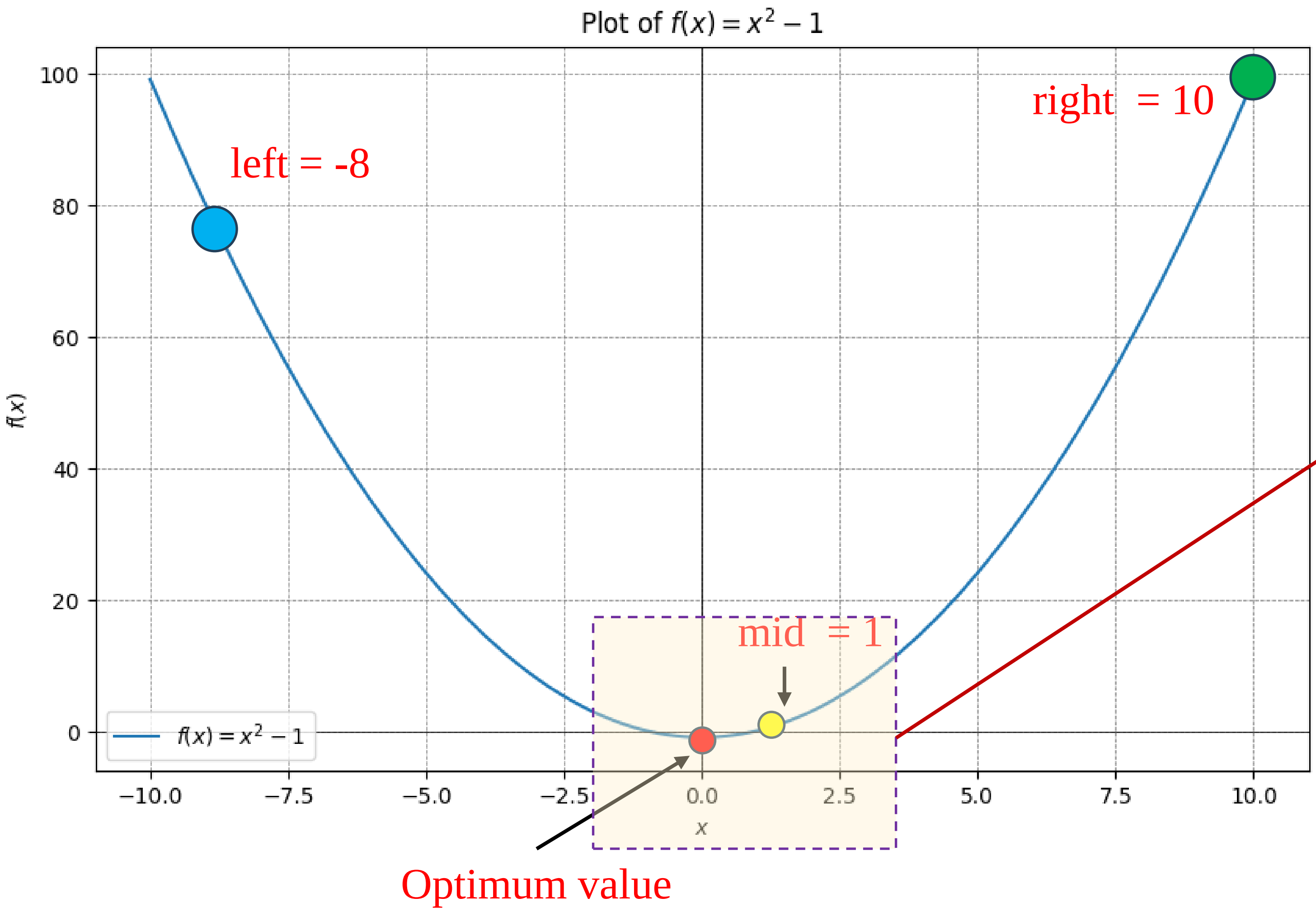


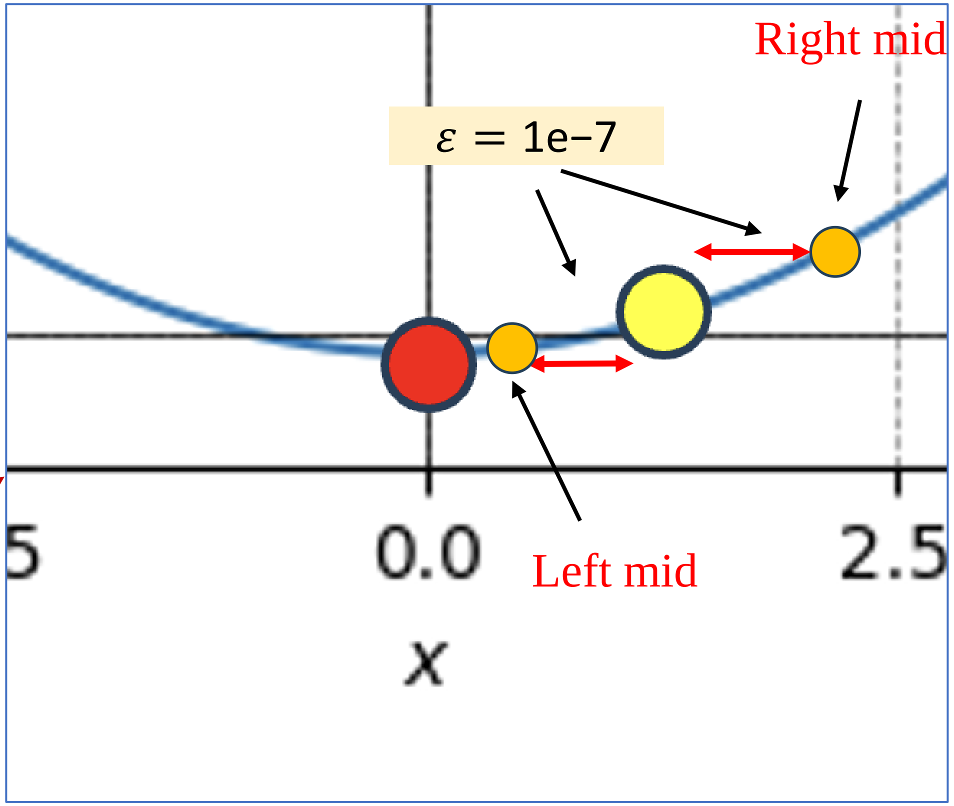
Find the minimum of the function

$f(x) = x^2 - 1$ Given $x \in [-8,10]$

AI VIET NAM
@aivietnam.edu.vn

Optimization using Binary Search



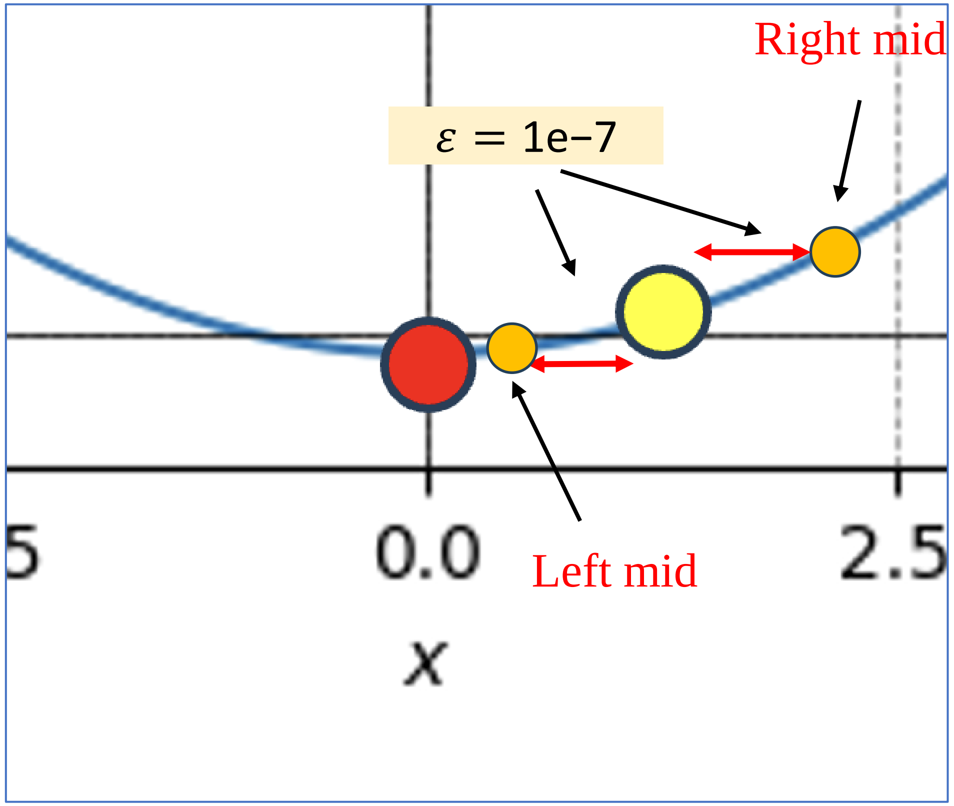



```
f_left_mid = f(left_mid)
f_right_mid = f(right_mid)
if f_left_mid < f_right_mid:
      right = mid
else:
      left = mid
```

AI VIET NAM
@aivietnam.edu.vn

Optimization using Binary Search

```python
def binary_search_min(f, a, b, tol=1e-7):

    print("left \t\t right \t\t middle")
    while b - a > tol:
        mid = (a + b) / 2
        print("%.10f \t %.10f \t %.10f" %  (a,  b, mid))
        left_mid = mid - tol
        right_mid = mid + tol

        f_left_mid = f(left_mid)
        f_right_mid = f(right_mid)

        if f_left_mid < f_right_mid:
            b = mid
        else:
            a = mid

    return (a + b) / 2

# Example usage
a = -8  # starting point of interval
b = 10   # ending point of interval
minimum_x = binary_search_min(f, a, b)
minimum_f = f(minimum_x)

print("The minimum value of f(x) = x^2 - 1 is at x =", minimum_x)
print("The minimum value of the function is f(x) =", minimum_f)
```

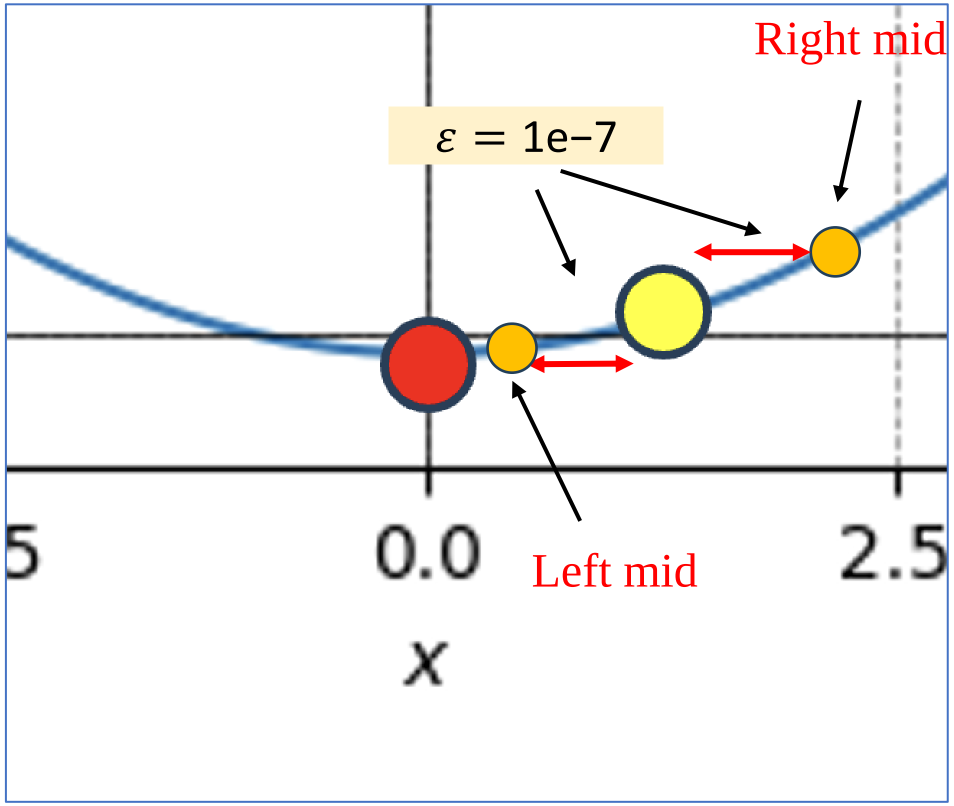



```python
f_left_mid = f(left_mid)
f_right_mid = f(right_mid)
if f_left_mid < f_right_mid:
      right = mid
else:
      left = mid
```

AI VIET NAM
@aivietnam.edu.vn

Optimization using Binary Search

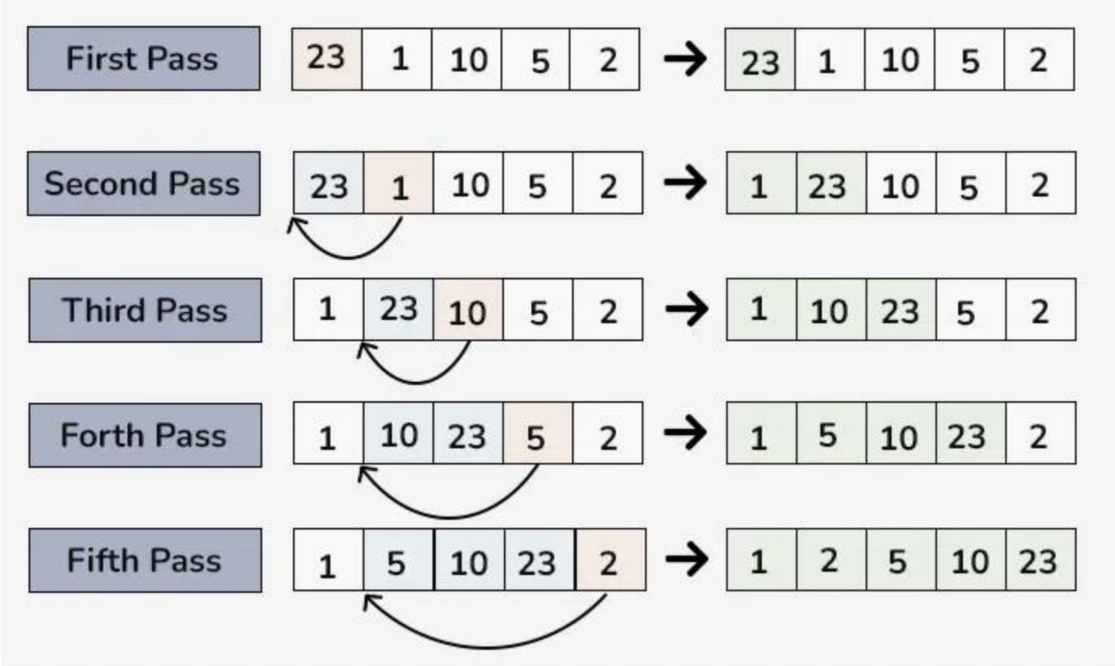
```
left                right               middle
-8.0000000000       10.0000000000       1.0000000000
-8.0000000000       1.0000000000        -3.5000000000
-3.5000000000       1.0000000000        -1.2500000000
-1.2500000000       1.0000000000        -0.1250000000
-0.1250000000       1.0000000000        0.4375000000
-0.1250000000       0.4375000000        0.1562500000
-0.1250000000       0.1562500000        0.0156250000
-0.1250000000       0.0156250000        -0.0546875000
-0.0546875000       0.0156250000        -0.0195312500
-0.0195312500       0.0156250000        -0.0019531250
-0.0019531250       0.0156250000        0.0068359375
-0.0019531250       0.0068359375        0.0024414062
-0.0019531250       0.0024414062        0.0002441406
-0.0019531250       0.0002441406        -0.0008544922
-0.0008544922       0.0002441406        -0.0003051758
-0.0003051758       0.0002441406        -0.0000305176
-0.0000305176       0.0002441406        0.0001068115
-0.0000305176       0.0001068115        0.0000381470
-0.0000305176       0.0000381470        0.0000038147
-0.0000305176       0.0000038147        -0.0000133514
-0.0000133514       0.0000038147        -0.0000047684
-0.0000047684       0.0000038147        -0.0000004768
-0.0000004768       0.0000038147        0.0000016689
-0.0000004768       0.0000016689        0.0000005960
-0.0000004768       0.0000005960        0.0000000596
-0.0000004768       0.0000000596        -0.0000002086
-0.0000002086       0.0000000596        -0.0000000745
-0.0000000745       0.0000000596        -0.0000000075
The minimum value of f(x) = x^2 - 1 is at x = 2.60770320892334e-08
The minimum value of the function is f(x) = -0.9999999999999993
```




```
f_left_mid = f(left_mid)
f_right_mid = f(right_mid)
if f_left_mid < f_right_mid:
        right = mid
else:
        left = mid
```

Other Sort Methods

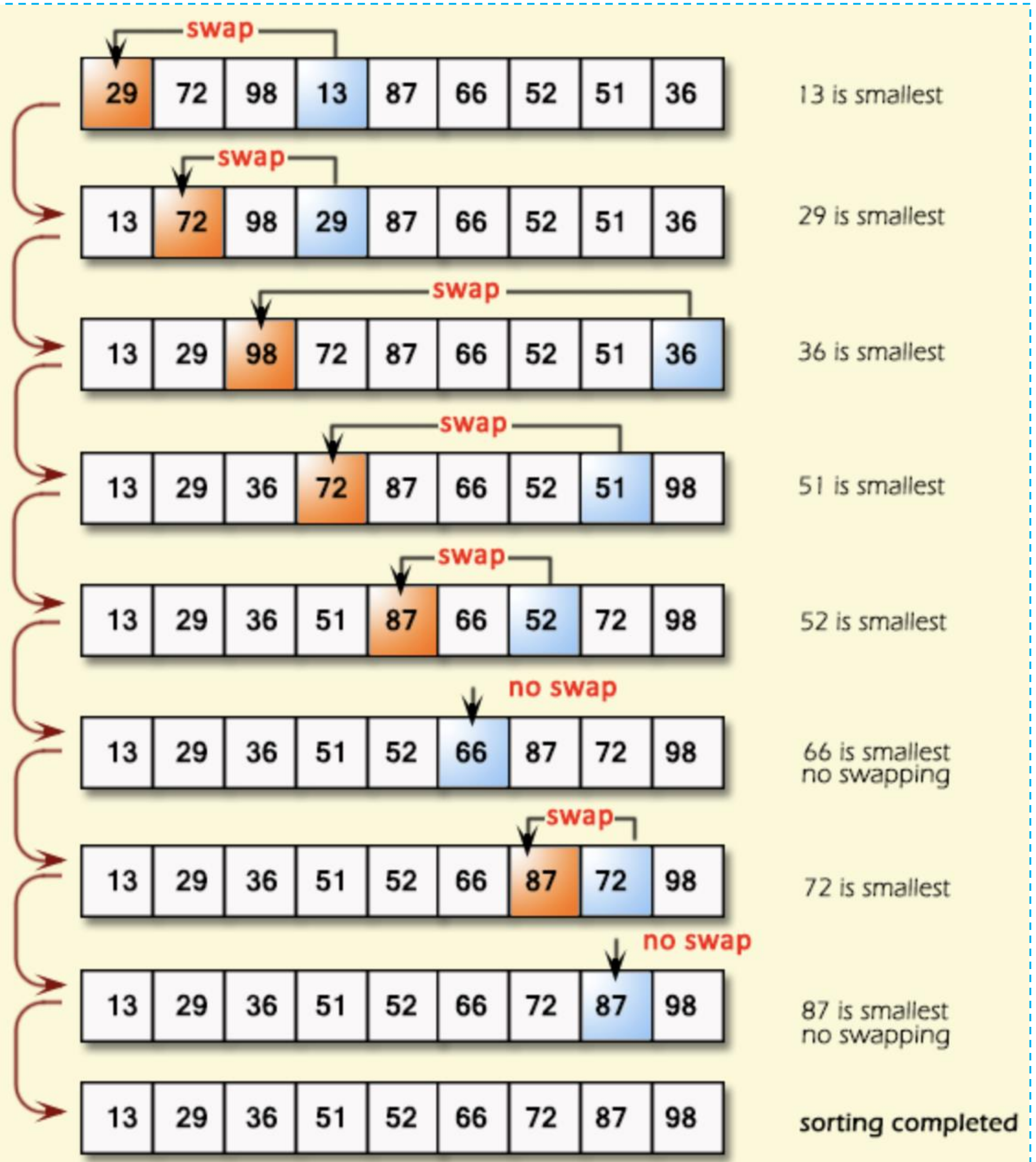
- Insert sort



AI VIET NAM
@aivietnam.edu.vn

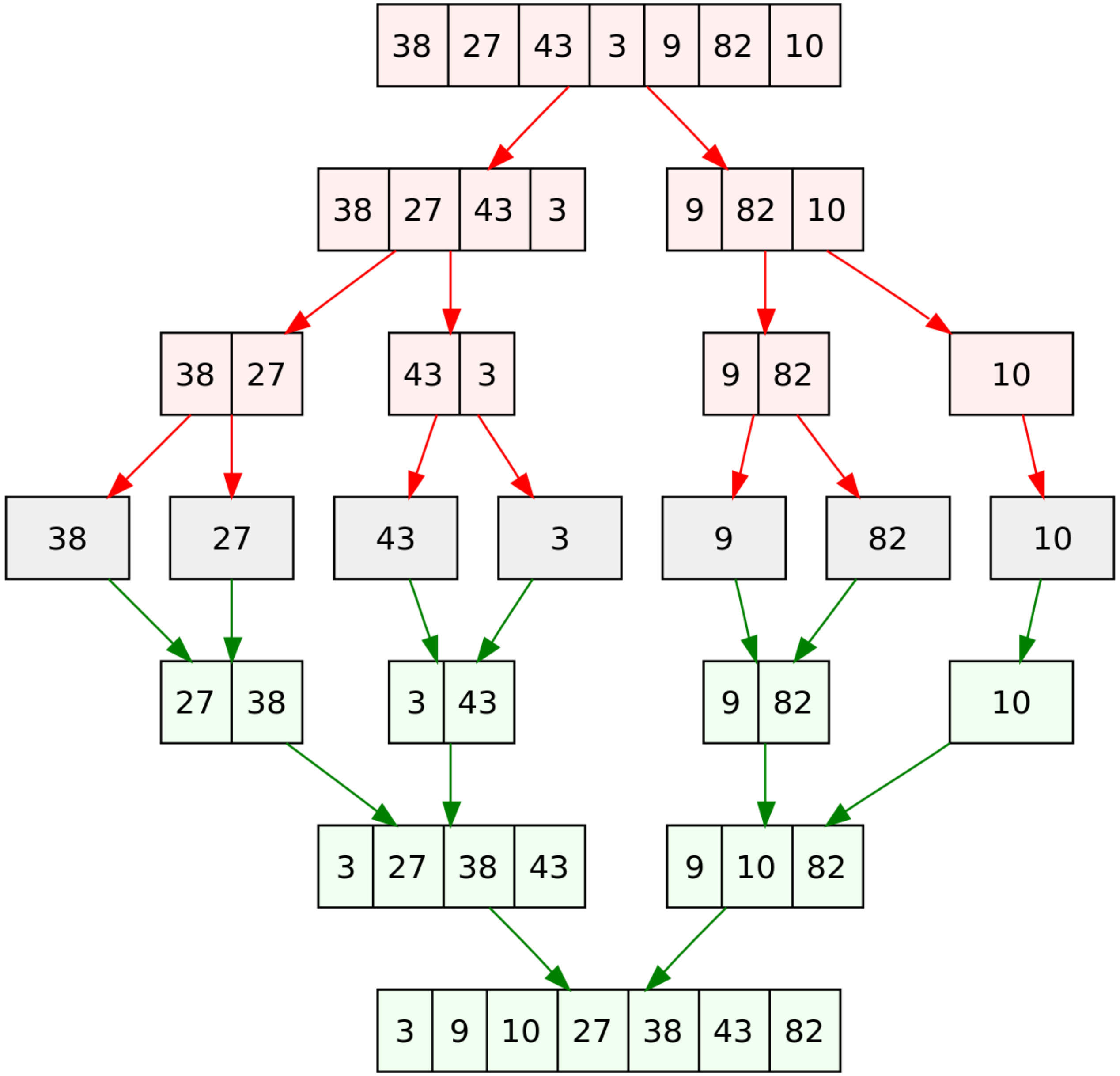
Other Sort Methods

❖ **Selection sort**



Other Sort Methods

❖ **Merge sort**



References

Problem Solving with Algorithms and Data Structures

Release 3.0

Brad Miller, David Ranum

September 22, 2013