BINARY SEARCH VISUALIZATION ON UNSORTED ARRAY

Faculty of Computer Science and Artificial Intelligence, Damietta University

Meet The Team

Amr Ahmad Gomaa

Abdullah Mahmoud Farha

Ibrahim Mohamed Rizk

Mohamed Tamer El Masry

Mahmoud Naser El Sharawy

Mustafa Mohamed Refaei

Raafat Mohamed El Menayar

Seif Basel Abo-Taleb

Wesam Ahmad El Sharawy

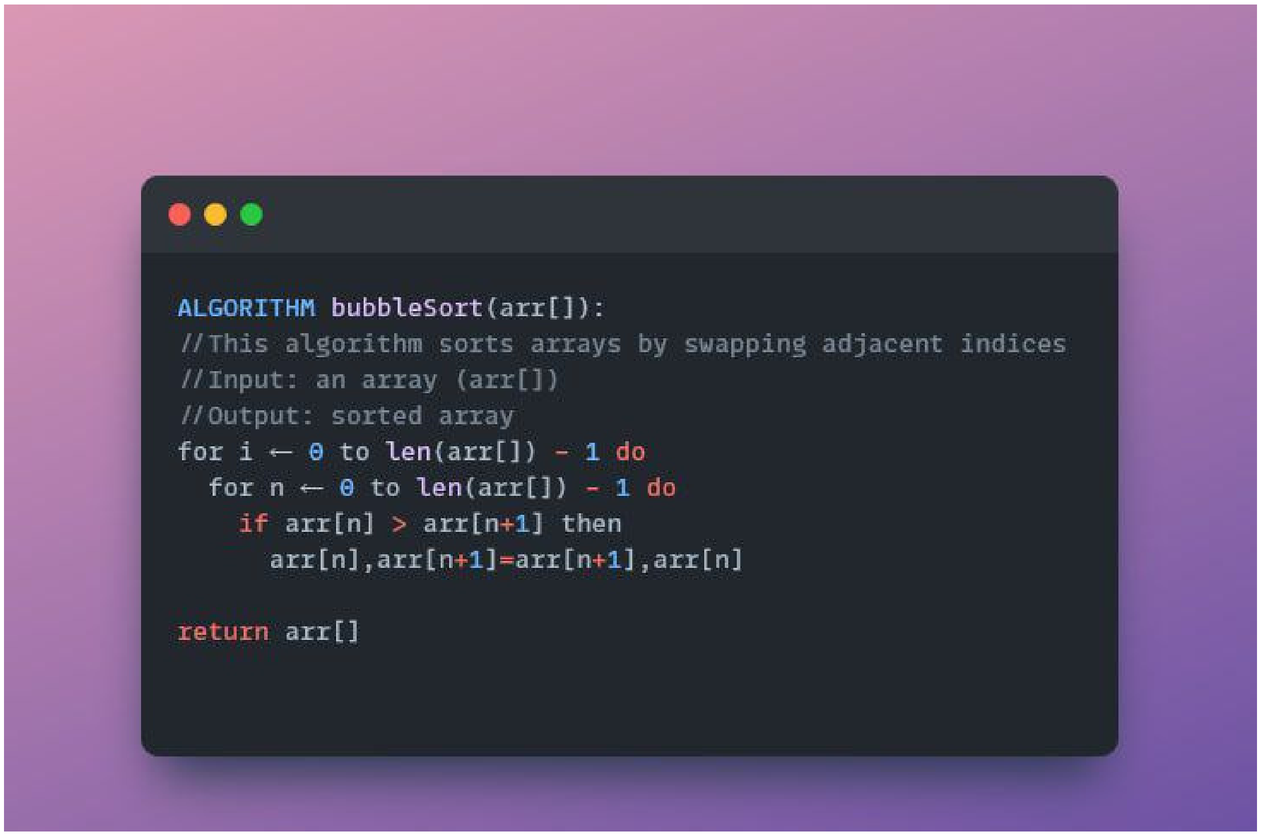
Wael Sherif Ahmad

Problem Description

Assuming we have an unsorted array, we seek to search for an element in such an array using binary search. Firstly, we need to sort this array using bubble sort, and only then can we apply our binary search algorithm for such a purpose.

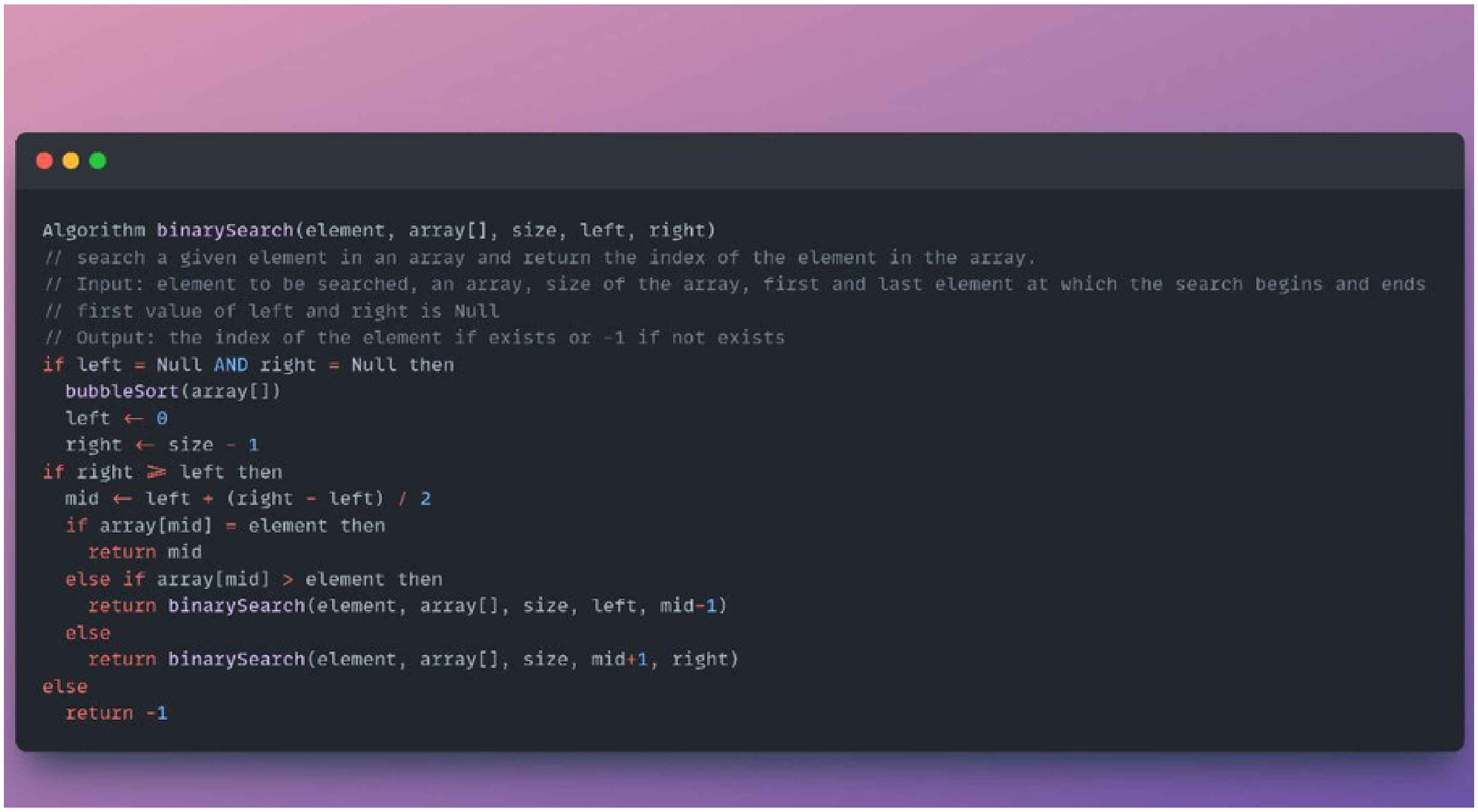
Algorithm

The bubble sort algorithm and binary search algorithm are presented in the first and second images, respectively.
The bubble sort algorithm sorts an array by repeatedly swapping the adjacent elements if they are in the wrong order, it assumes an array as an input, and finally returns a sorted array as an output.
The binary search algorithm searches for an element in a given array by comparing the target value to the middle element of the array. If they are not equal, then it narrows the search to the other half, and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this process until the target value is found, it assumes a sorted (in ascending order) integer array nums of n elements and a target value, the size of the given array, the first and last element at which the search begins and ends as an input, and finally returns the index of the target value as an output.



```
ALGORITHM bubbleSort(arr[]):
//This algorithm sorts arrays by swapping adjacent indices
//Input: an array (arr[])
//Output: sorted array
for i ← 0 to len(arr[]) - 1 do
  for n ← 0 to len(arr[]) - 1 do
    if arr[n] > arr[n+1] then
      arr[n],arr[n+1]=arr[n+1],arr[n]

return arr[]
```



```
Algorithm binarySearch(element, array[], size, left, right)
// search a given element in an array and return the index of the element in the array.
// Input: element to be searched, an array, size of the array, first and last element at which the search begins and ends
// first value of left and right is Null
// Output: the index of the element if exists or -1 if not exists
if left = Null AND right = Null then
  bubbleSort(array[])
  left ← 0
  right ← size - 1
if right ≥ left then
  mid ← left + (right - left) / 2
  if array[mid] = element then
    return mid
  else if array[mid] > element then
    return binarySearch(element, array[], size, left, mid-1)
  else
    return binarySearch(element, array[], size, mid+1, right)
else
  return -1
```

Analysis

The mathematical analysis of the two discussed algorithms is derived and presented in the following images.

Bubble Sort:
The Best and worst case have the same time complexity. Because in either case, it must go through the two loops.

Bubble sort

$$\boldsymbol{Cavg} = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} 1 = \sum_{i=0}^{n-1}(n-1-i+1) = \sum_{i=0}^{n-1}(n-i) = \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i$$

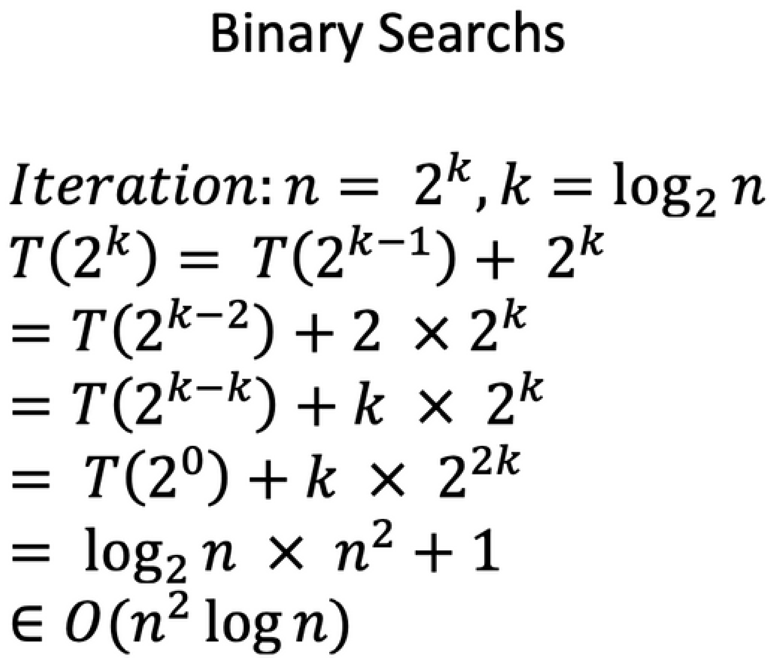
$$= n\sum_{i=0}^{n-1} 1 - \sum_{i=0}^{n-1} i = n(n-1+1) = n^2 - \frac{(n-1)n}{2} = n^2 - \frac{n^2-n}{2}$$

$$= n^2 - \frac{n^2}{2} + \frac{n}{2} = \frac{n^2}{2} + \frac{n}{2}$$

$$\therefore \theta(n^2)$$

Binary Search:
If the given element is present in the middle of the array, the time complexity in the best case yields 1, and the worst case yields the following recurrence relation, as explained in the following picture.

$$T(n) = T(n/2) + n^2 \qquad T(1) = 1$$



Binary Searchs

$$Iteration\colon n = 2^k, k = \log_2 n$$
$$T(2^k) = T(2^{k-1}) + 2^k$$
$$= T(2^{k-2}) + 2 \times 2^k$$
$$= T(2^{k-k}) + k \times 2^k$$
$$= T(2^0) + k \times 2^{2k}$$
$$= \log_2 n \times n^2 + 1$$
$$\in O(n^2 \log n)$$

Code & Output

Here is a GitHub repository in the link below that demonstrates the output and presents the implementation of the explained algorithms in Python.

https://github.com/MoDev2002/Algoramers

Verdict

In this project, we analyzed the prementioned algorithms mathematically. And implemented those algorithms in python, and thus, we can apply the bubble sort algorithm on an unsorted array and afterward apply the binary search algorithm on the output (sorted array in ascending order) to find a given value. Furthermore, we implemented a visualization for the bubble sort and binary search algorithm using the Tkinter library.