T2A1-A Workbook

Submitted by: Chen Zhang

Q1 Identify and explain the workings of TWO sorting algorithms and discuss and compare their performance/efficiency (i.e., Big O)

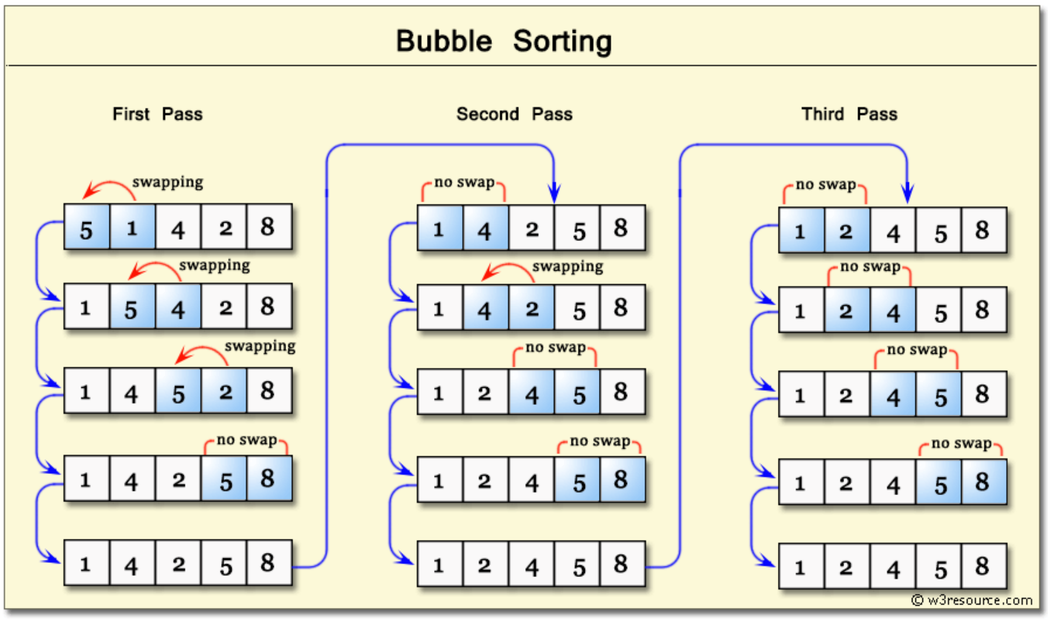
Bubble Sort

The bubble sort is a simple sorting algorithm. It repeatedly visits the array to be sorted, comparing two elements at a time and swapping them over if they are in the wrong order. The visit is repeated until no more exchange is needed, which means that the array is sorted. The name of this algorithm comes from the fact that the smaller elements are slowly "floated" to the top of the array by swapping.

Algorithm description

1. Compare adjacent elements. If the first is larger than the second, swap them both.
2. Do the same for each pair of adjacent elements, from the first pair at the beginning to the last pair at the end, so that the element at the end should be the largest number.
3. Repeat the above steps for all elements except the last one.

Repeat steps 1 to 3 until the sorting is complete. (As shown as gif)



Algorithm Implementation

```ruby
def bubble_sort(array)
   n = array.length
   swapped = true
   while swapped do
     #Created a variable check so we don’t run into infinite loop.
    swapped = false
     (n - 1).times do |i|
       if array[i] > array[i + 1]
         array[i], array[i + 1] = array[i + 1], array[i]
     swapped = true
       end
     end
   end
   array
end
```

Applicable scenarios

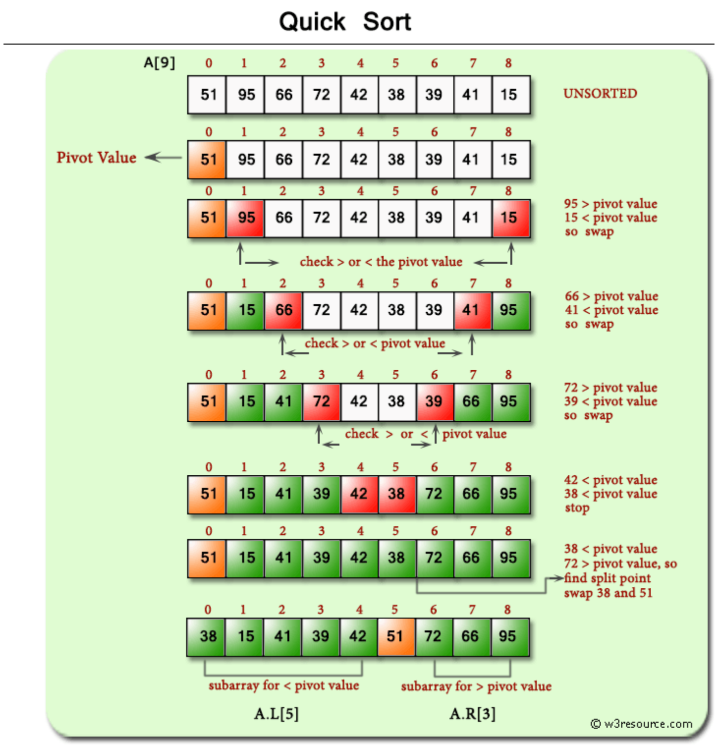
The bubble sort is simple in idea and simple in code, and is especially suitable for sorting small data. However, due to the high complexity of the algorithm, it is not suitable for use when there is a large amount of data.

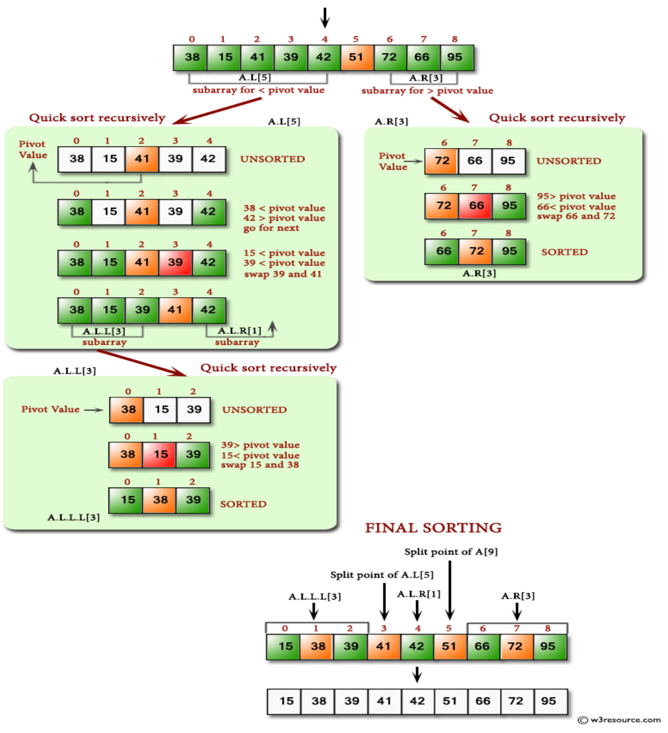
Quick Sort

Quick sort is a highly recognizable sorting algorithm, and its excellent sorting performance for large data and relatively simple implementation among algorithms of the same complexity make it destined to receive more favor than other algorithms.

Algorithm description

1. Pick an element from the array, called the "benchmark" (pivot), and
2. Reorder the array so that all elements smaller than the benchmark are placed in front of the benchmark and all elements larger than the benchmark are placed behind it (the same number can go to either side). At the end of this partitioning, the benchmark is in the middle of the array. This is called a partition operation.
3. Recursively sort the sub-array of elements less than the base and the sub-array of elements greater than the base.





Applicable Scenarios

Quick sorting is suitable in most cases, especially when the data volume is large. However, when necessary, optimization needs to be considered to improve its performance in the worst case.

Compare these two sorting algorithms

Performance and efficiency

Bubble sort - The optimal time complexity, O(n), is exhibited when the data is perfectly ordered. In other cases, it is almost always O(n^2). Thus, the algorithm performs best when the most of data is ordered. Memory is O(1)

To make the algorithm have O(n) complexity in the best case, some improvements need to be made by adding a Boolean swapped = true. When no swap is performed in the current round, it means that the array is already ordered and there is no need to perform the next round of loops and exit directly.

Quick sort - The optimal and average time complexity, O(nlogn). The worst scenario is O(n^2). Memory is O(logn)

Stability

Bubble Sort - When adjacent elements are equal, they do not exchange positions, so bubble sort is stable.

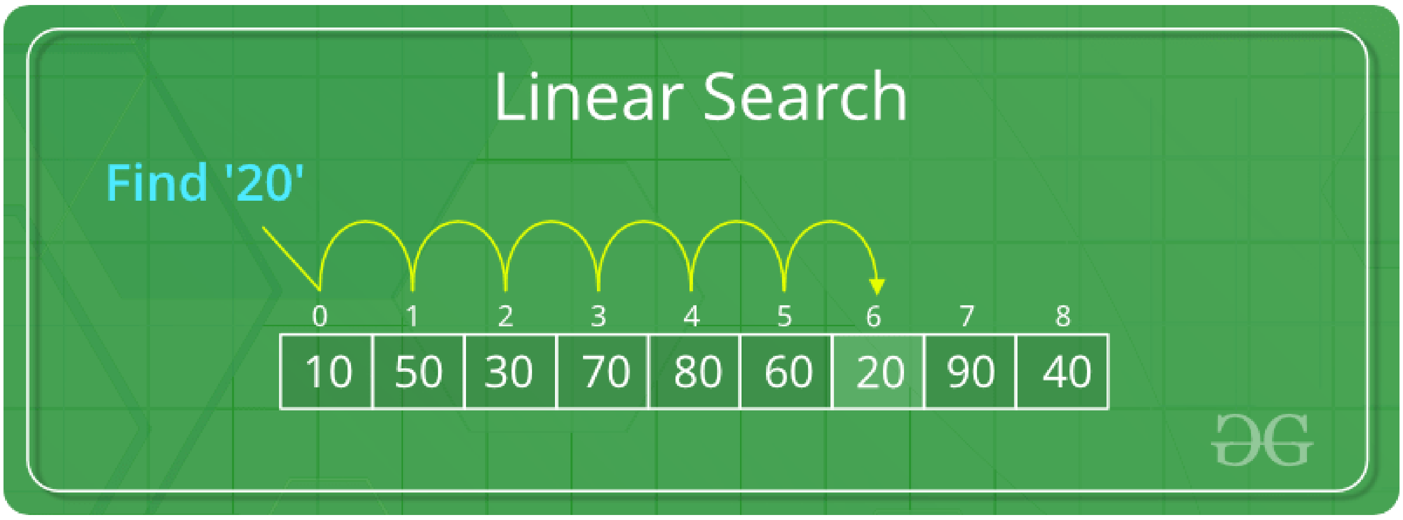
Quick Sort - Quick sort is not stable. This is because we cannot guarantee that equal data will be scanned in order and stored in order.

Q2 Identify and explain the workings of TWO search algorithms and discuss and compare their performance/efficiency (i.e., Big O)

Linear Search

Linear search is suitable for linear tables those storage structure is linear storage or linked storage.

Basic idea: Linear search belongs to the unordered search algorithm. Start from one end of the linear table, scan sequentially, and compare the keywords of the scanned nodes with the given value K. If they are equal, the search is successful; if the node with keyword equal to K is not found at the end of the scan, the search failed.



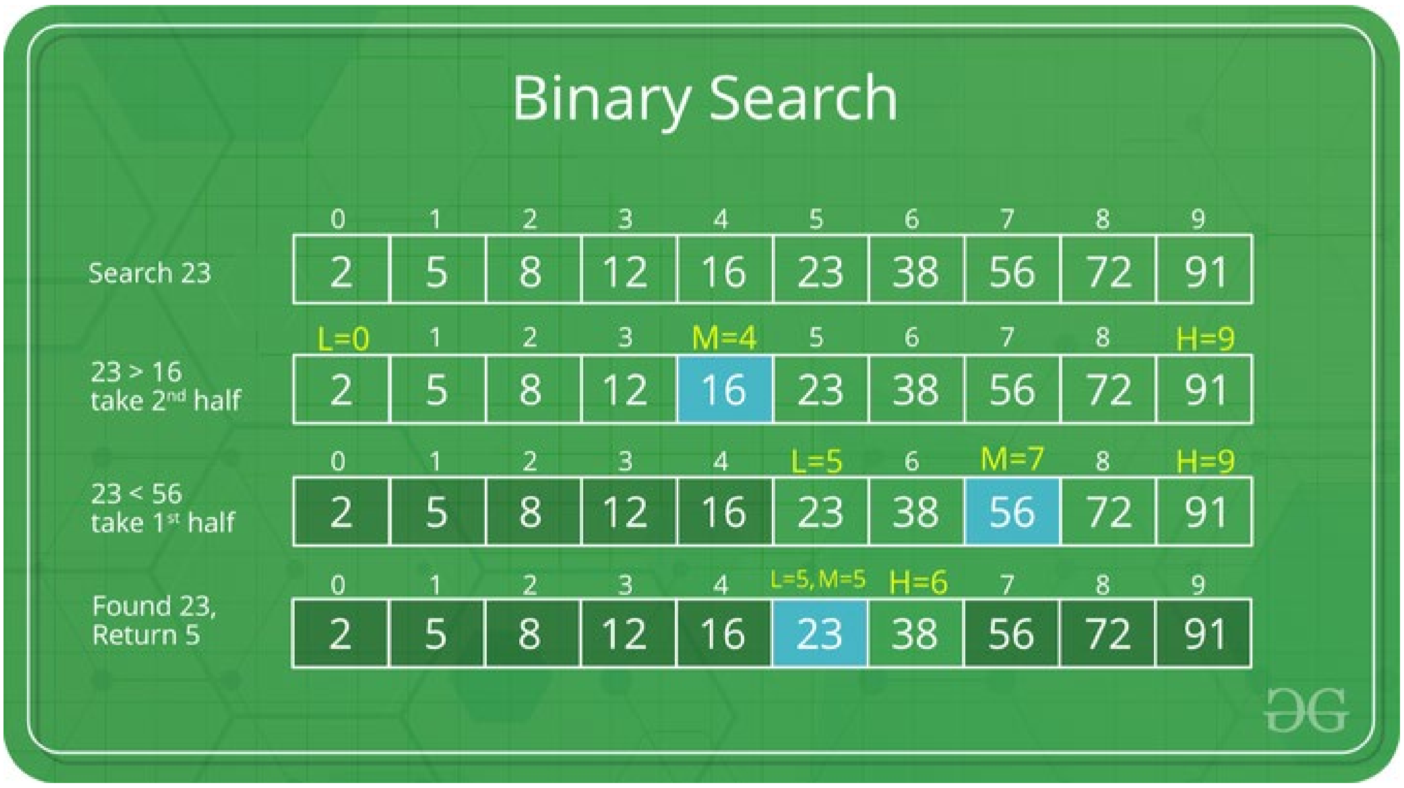
Algorithm Implementation

```ruby
def linear_search(array, key)
  i = 0
  while i < array.length
      if array[i] == key
        return "#{key} is found in array at index #{array.index(key)}"
      end
      i+=1
    end
    return "#{key} is not found"
end
```

Binary Search

Binary search is suitable for the elements which must be ordered, if they are unordered then the sorting operation should be performs first.

Basic idea: Binary belongs to the ordered search algorithm. With a given value K, first compared with the keyword of the intermediate node, the intermediate node of the linear table into two sub-tables, if equal, the search is successful; if not equal, and then determine which sub-table to look for next according to the comparison of K and the keyword of the intermediate node, so recursively, until the search to or the end of the search found no such node in the table.



Algorithm Implementation

```
def binary_search(array, key)
  max = array.length - 1
  min = 0
  while min <= max
    mid = (min + max) / 2
    if array[mid] == key
      return mid
    elsif array[mid] > key
      max = mid - 1
    else
      min = mid + 1
    end
  end
  return "#{key} is not found"
end
```

Compare these two search algorithms

Linear Search - he average search length for a successful search is: (assuming equal probability for each data element)

When the search is unsuccessful, n times comparisons are required and the time complexity is O(n);

Therefore, the time complexity of linear search is O(n).

Binary Search - In the worst case, the number of keyword comparisons is ($\log_2(n) + 1$) which is rounded down, and the expected time complexity is $O(\log_2 n)$

Compare Average Search Length (ASL)

Average Search Length: In the search operation, since the time spent on the comparison of keywords, the average number of keywords that need to be compared with the value to be searched is called Average Search Length.

Assume array.length = 10

Linear Search: ASL = (n + 1)/2, in this case, equals to 5.5

Binary Search: ASL = (1*1 + 2*2 + 3*4 + 4*3)/10 = 2.9