
WELCOME
TechPro

[Nasos Lentzas]

[Intro to Algorithms]

[23/10/2024]

TechPro

In this Course

1. Sorting
2. Searching
3. Hashing

TechPro

Sort Algorithms

TechPro

Bubble sort

In this algorithm:

- traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so on until the data is sorted.

6 5 3 1 8 7 2 4

TechPro

Time complexity: O(N^2)
Space complexity: O(1)

Bubble sort

Pros	Cons
Easy to understand & implement	Very slow for large data sets
Does not require additional memory	Requires comparison that may reduce efficiency
Stable algorithm*	

TechPro

*elements with the same key value maintain their relative order in the sorted output.

Insertion sort

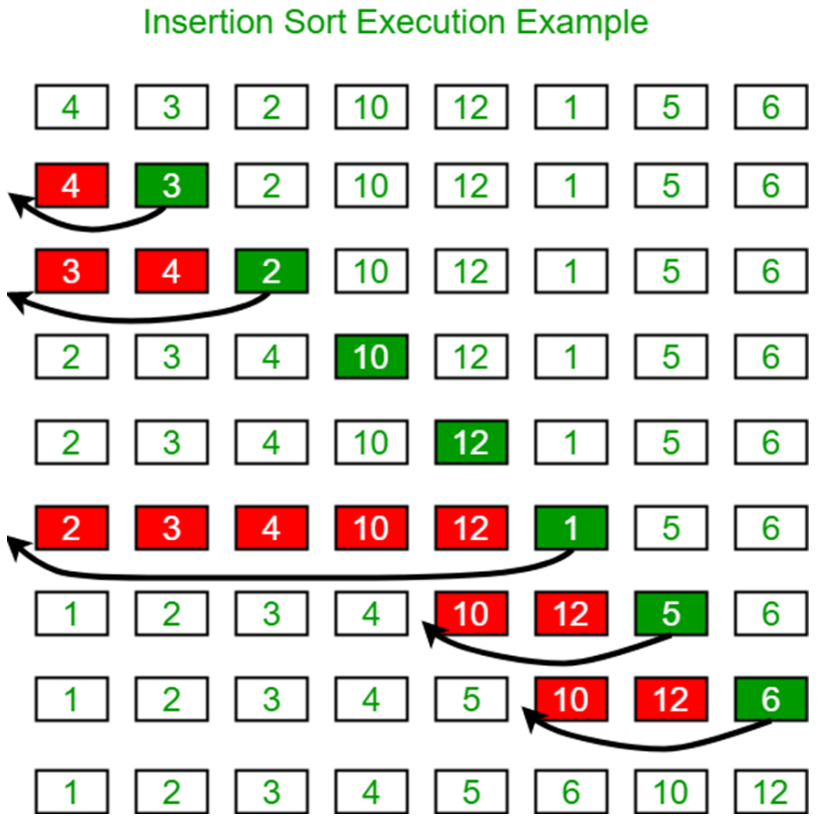
To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

6 5 3 1 8 7 2 4

TechPro

Time complexity: O(N^2)
Space complexity: O(1)

Insertion sort



TechPro

Merge sort

Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array

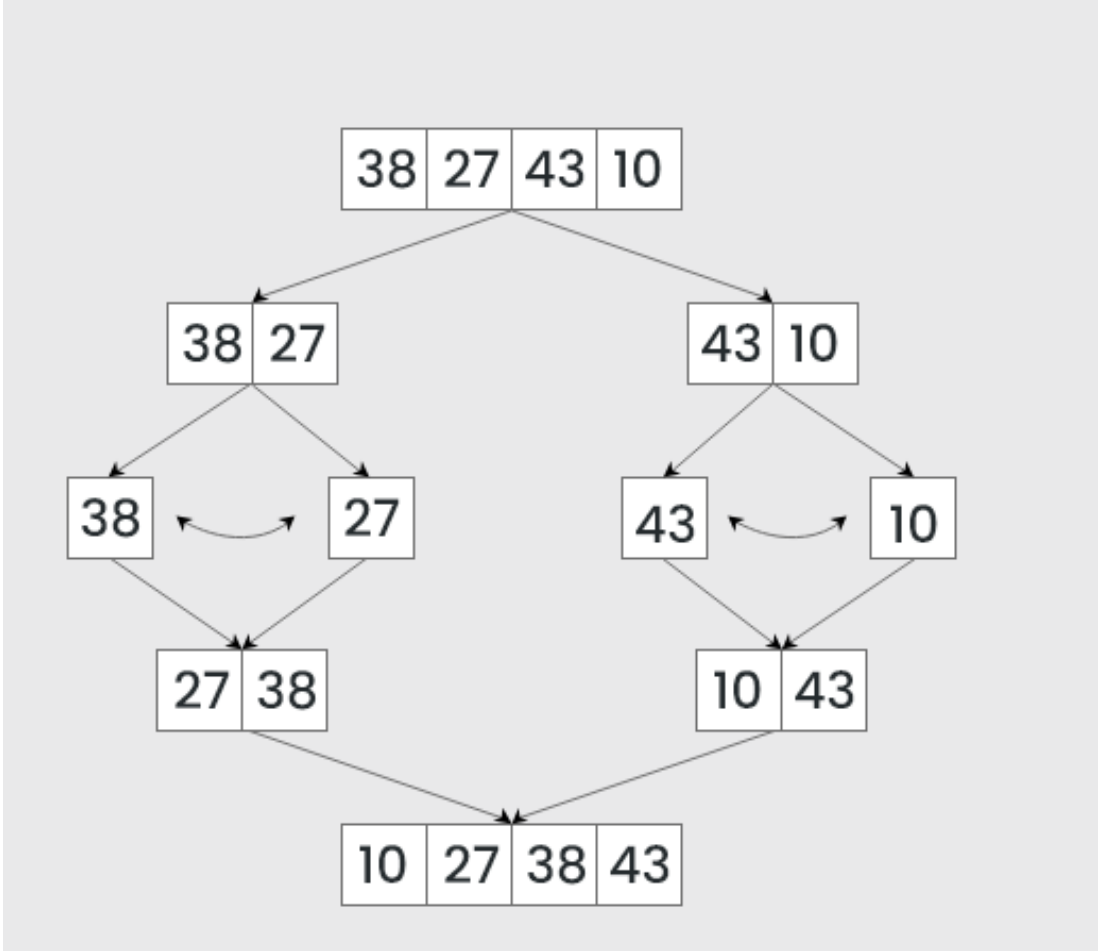
- Sorting large datasets: Merge sort is particularly well-suited for sorting large datasets due to its guaranteed worst-case time complexity of O(n log n).
- External sorting: Merge sort is commonly used in external sorting, where the data to be sorted is too large to fit into memory.
- Custom sorting: Merge sort can be adapted to handle different input distributions, such as partially sorted, nearly sorted, or completely unsorted data.

6 5 3 1 8 7 2 4

TechPro

Time complexity: O(N*log(N))
Space complexity: O(N)

Merge sort



Time complexity: O(N*log(N))
Space complexity: O(N)

TechPro

Merge sort

Pros	Cons
Stable algorithm	Space complexity
Guaranteed worst-case performance	Not in-place
Parallelizable	Not optimal for small datasets

TechPro

Time complexity: O(N*log(N))
Space complexity: O(N)

Quick sort

QuickSort is a sorting algorithm based on the Divide and Conquer that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

- Choose a Pivot: Select an element from the array as the pivot. The choice of pivot can vary (e.g., first element, last element, random element, or median).
- Partition the Array: Rearrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right. The pivot is then in its correct position, and we obtain the index of the pivot.
- Recursively Call: Recursively apply the same process to the two partitioned sub-arrays (left and right of the pivot).
- Base Case: The recursion stops when there is only one element left in the sub-array, as a single element is already sorted.

6 5 3 1 8 7 2 4

TechPro

Time complexity: O(N^2))
Space complexity: O(1)

Quick sort

Pros	Cons
divide-and-conquer algorithm that makes it easier to solve problems.	worst-case time complexity of O(N^2),
efficient on large data sets	not a stable sort
low overhead, as it only requires a small amount of memory to function Fastest general-purpose algorithm for large data	Not optimal for small datasets

TechPro

Time complexity: O(N*log(N))
Space complexity: O(1)

Search Algorithms

TechPro

Linear Search

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

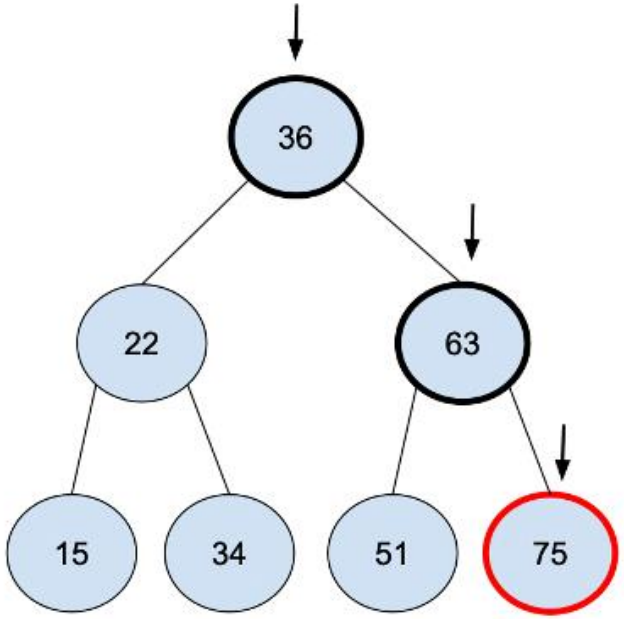
Pros	Cons
Can be used regardless of sorted array or not	High time complexity
No additional memory	Not suitable for large datasets
Well suited for small data sets	

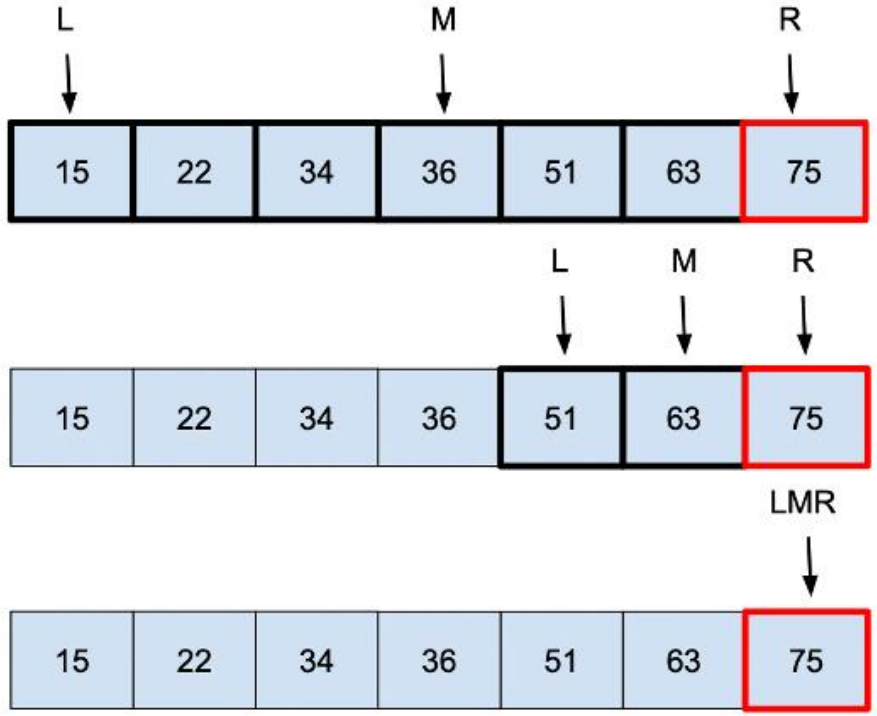
TechPro

Time complexity: O(N))
Space complexity: O(1)

Binary Search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted.





TechPro

Time complexity: O(log2(N))
Space complexity: O(1)

Binary Search

Pros	Cons
Fast especially for large sets	Array should be sorted
More efficient than other search algorithms	Data structure must be stored in contiguous memory locations
Well suited for large data sets	Requires comparable items

TechPro

Used on:

- Database search
- Graphics (ray tracing, texture mapping etc)
- Used on complex algorithms (i.e. machine learning)

Ternary Search

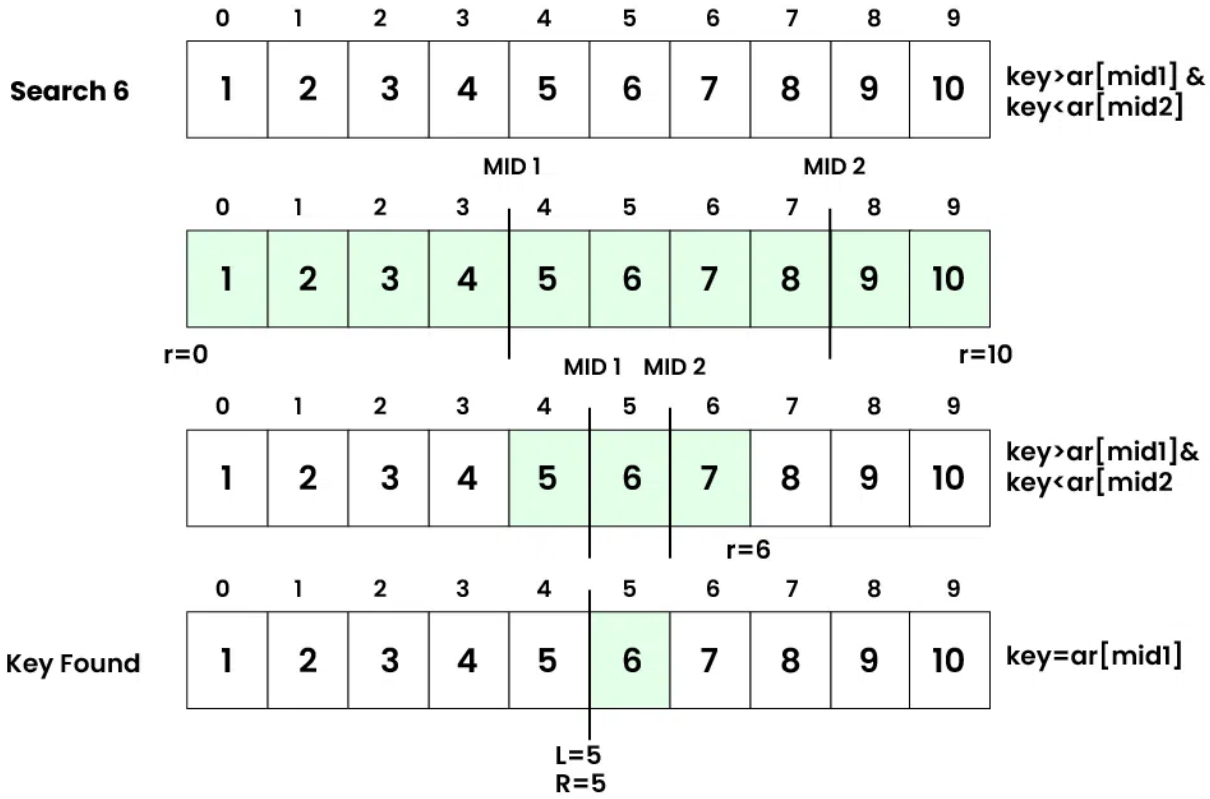
It operates on the principle of dividing the array into three parts instead of two, as in binary search. The basic idea is to narrow down the search space by comparing the target value with elements at two points that divide the array into three equal parts

- Set two pointers, left and right, initially pointing to the first and last elements of our search space.
- Calculate two midpoints, mid1 and mid2, dividing the current search space into three roughly equal parts:
  - mid1 = left + (right – left) / 3
  - mid2 = right – (right – left) / 3
- Compare with search key
  - If the target is equal to the element at mid1 or mid2, the search is successful, and the index is returned
  - If the target is less than the element at mid1, update the right pointer to mid1 – 1.
  - If the target is greater than the element at mid2, update the left pointer to mid2 + 1.
  - If the target is between the elements at mid1 and mid2, update the left pointer to mid1 + 1 and the right pointer to mid2 – 1.
- Repeat the process with the reduced search space until the target is found or the search space becomes empty.

TechPro

Time complexity: O(log3(N))
Space complexity: O(1)

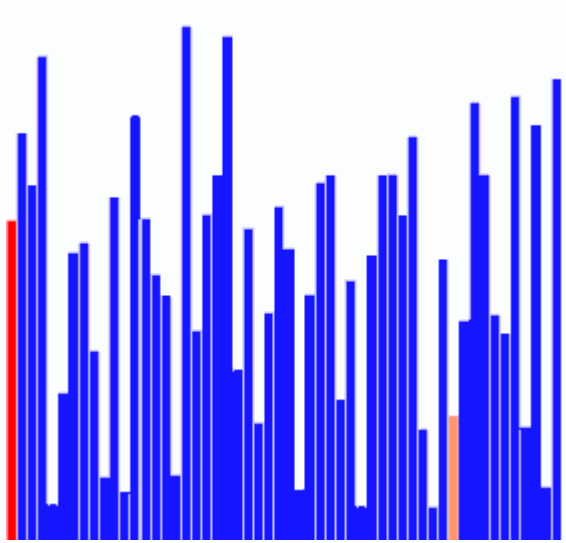
Ternary Search



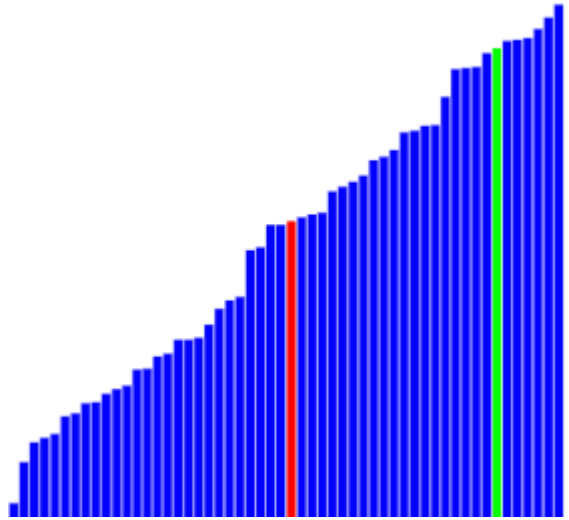
TechPro

Time complexity: O(log3(N))
Space complexity: O(1)

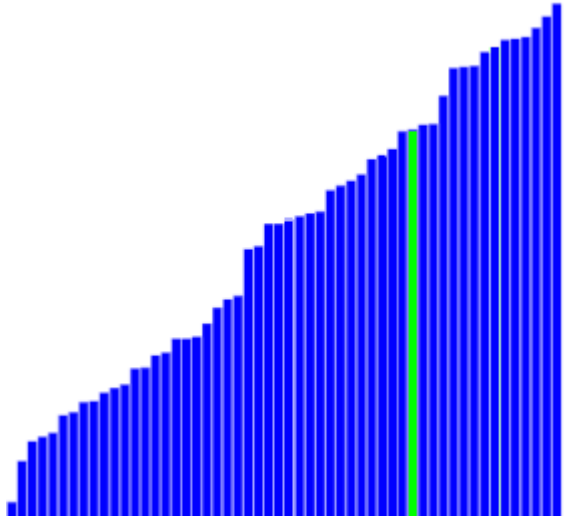
Search comparison



Linear Search



Binary Search



Ternary Search

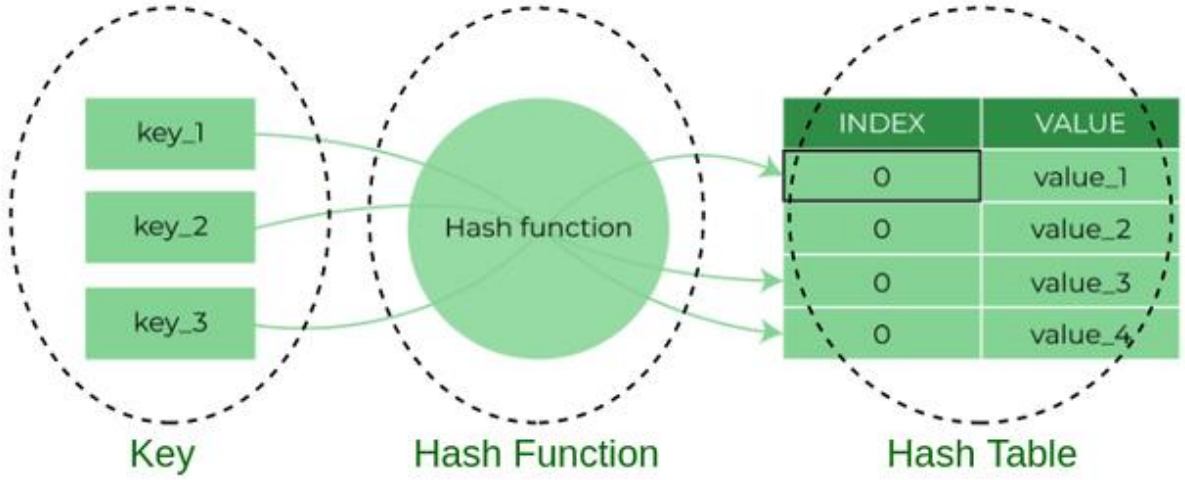
TechPro

Hashing

TechPro

Hashing

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.



TechPro

How hashing works

Suppose we have a set of strings {“ab”, “cd”, “efg”} and we would like to store it in a table:

- Choose a hash function to use (e.g. sum of character mod Table size)
- We assign each letter to a number (“a” = 1, “b” =2 etc)
- We calculate the numerical value of the string. e.g. “ab” = 1+2=3
- We have a table with a size of 7 so we calculate the hash value (“ab” mod 7 =3)

0	1	2	3	4	5	6
cd			ab	efg		

TechPro

Hashing

Pros	Cons
Better synch that other structures	Inefficient when we have many collisions
More efficient that trees	Collisions are practically not avoided on large sets
Constant time for search, insert, delete	Does not allow null values

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used. Hashing is used in:

- Database indexing
- Disk-based data structures
- Cache mapping
- Password verification
- Cryptography

TechPro

Hashing

A hash function that maps every item into its own unique slot is known as a perfect hash function. We can construct a perfect hash function if we know the items and the collection will never change

A good hash function must have the following:

- Efficiently Computable
- Uniformly distribute the keys
- Minimize collisions
- Low load factor (number of items in table divided by the size of table)

Hash collisions

It is possible for a hash function to generate the same hash value for two keys

Separate Chaining:
The idea is to make each cell of the hash table point to a linked list of records that have the same hash function value. Chaining is simple but requires additional memory outside the table.

Anagram problem

Given a table with all English words how can I find all anagrams of a word quickly?

Input: dog
Output: dog, god

Words
dog
god
listen
enlist
silent
tinsel

TechPro

Feedback Discussion

TechPro

Thank you

TechPro