CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Searching Algorithms

Housekeeping

Scheduling Updates

- A3: due date pushed back to:
- Review [MEC] Project due date pushed back to:

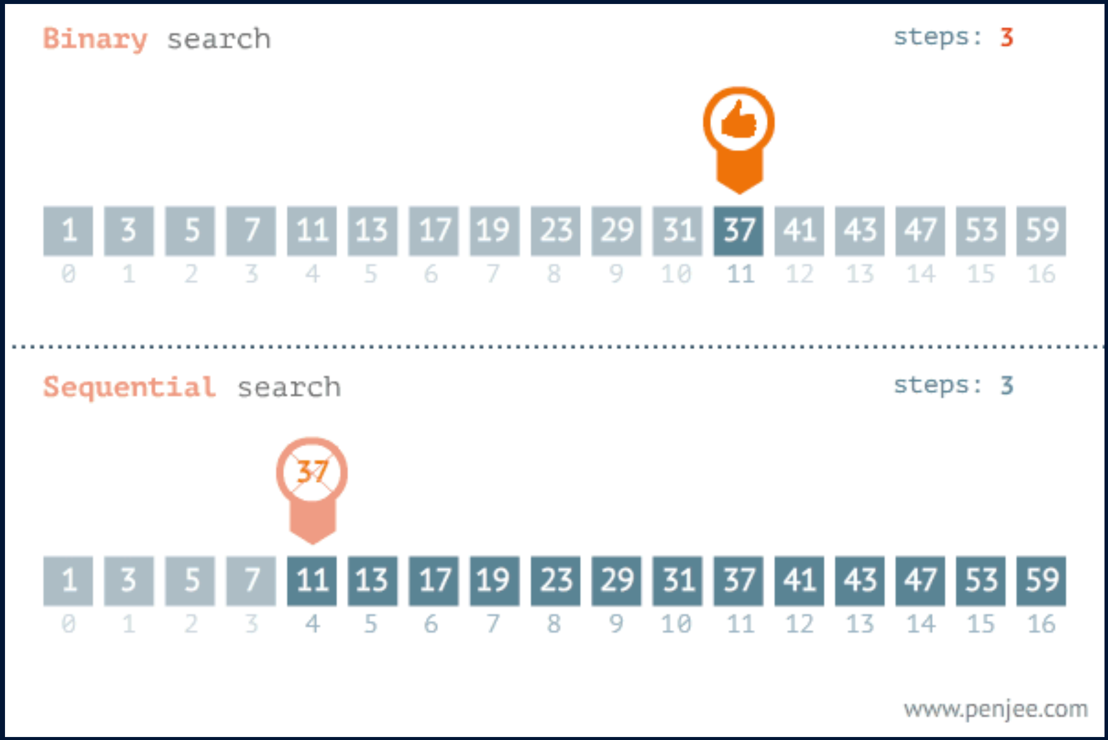
Searching Algorithms

Interval Search

- repeatedly target the center of the search structure and divide the search space in half.
- ex. binary search
- *note: specifically designed for searching in sorted data-structures…*

Sequential Search

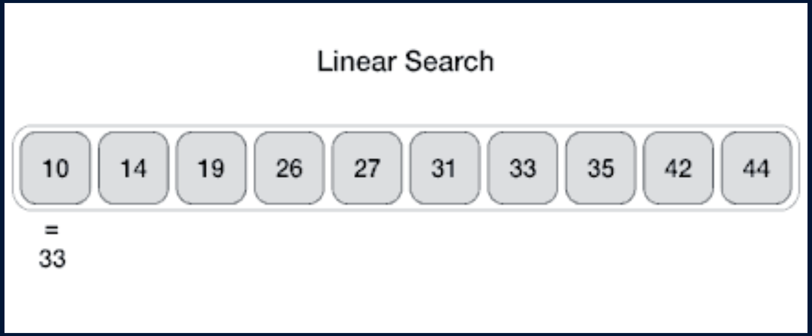
- the list or array is traversed sequentially and every element is checked.
- ex. linear search



Linear Search

Linear Search: Implementation

```
// Pseudocode
// ----------
// Iterate from 0 to N-1,
// compare the value of every index with x
// if they match, return index

int linearSearch(int array[], int n, int x) {
    // Going through array sequencially
    for (int i = 0; i < n; i++)
        if (array[i] == x)
            return i;
    return -1;
}
```



https://www.tutorialspoint.com/data_structures_algorithms/images/linear_search.gif

Linear Search: Analysis

Rules

- Consider all possible cases.
- Find the number of comparisons for each case.
- Add the number of comparisons and divide by the number of cases.

Best-case => $T(n) = O(1)$

$$target = A[0] = 1 \; comparison$$

Worst-case => $T(n) = O(n)$

$$target = A[n-1] = n \; comparisons$$

Average-case *(in a successful search)* => $T(n) = O(n)$

$$\frac{1 + 2 + \ldots + n}{n} = \frac{1}{n} * \frac{n(n-1)}{2} = \frac{n-1}{n}$$

Binary Search

Binary Search: Pseudocode

Iterative Approach

- Consider start index to be at 0 and last index to be $n-1$th index at starting $//n->length$
- Find middle $index(mid)$ of the array
- If key is found to be less than $mid\ index\ element$ then update last index of the array to $mid-1$
- Else if key is found to be greater than $mid index element$ then update start index of the array to $mid+1$
- Else check for $mid\ index\ element$ with key if not match repeat the above steps til start index is less than end index

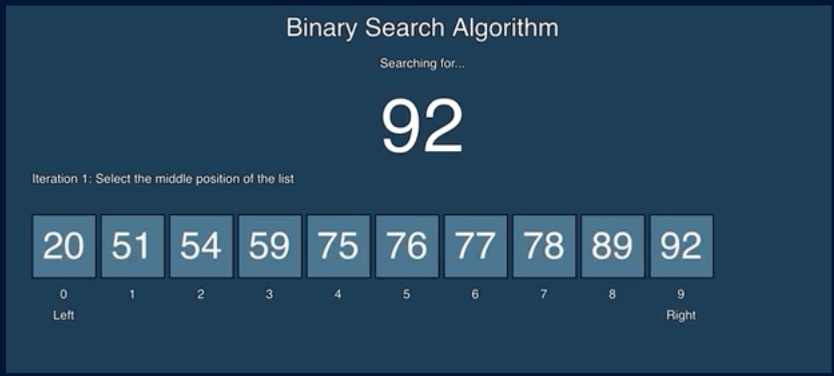
Recursive Approach

- If $start$ is less than end perform Binary search else terminate the algorithm.
- If the element at the $middle\ index$ is equal to the key then return the index as it found the key
- Else if the key is less than the element at the $middleindex$ then call the function by passing end as $mid-1$ (as the key will be less than mid element)
- Else if the key is greater than the element at the $middle\ index$ then call the function by passing start as $mid+1$ (as the key will be greater than mid)

https://takeuforward.org/data-structure/binary-search-explained/

Binary Search: Iterative

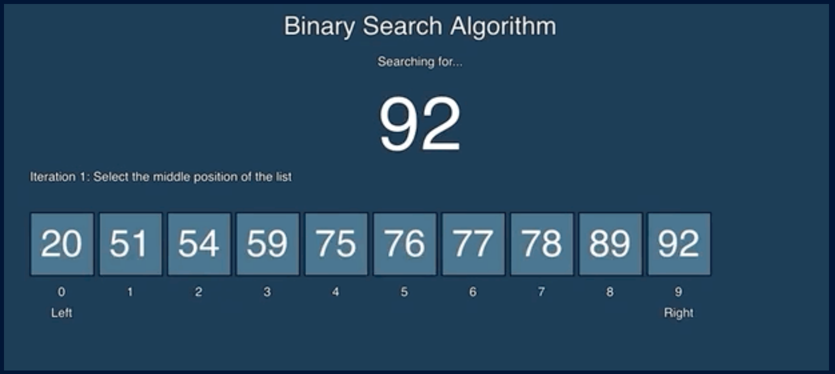
```
int binarySearch(int array[], int x, int low, int high) {

  // Repeat until the pointers low and high meet each other
    while (low <= high) {

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

        if (array[mid] == x)
            return mid;

        if (array[mid] < x)
            low = mid + 1;

        else
            high = mid - 1;
    }
    return -1;
}
```



https://www.codecademy.com/resources/blog/content/images/2018/10/binary-search-small.gif

Binary Search: Recursive

```
int binarySearch(int arr[], int start, int end, int k) {

  if (start > end) {
    return -1;
  }
  int mid = (start + end) / 2;

  if (k == arr[mid]) {
    return mid;

  else if (k < arr[mid])
    return binarySearch(arr, start, mid - 1, k);

  else
    return binarySearch(arr, mid + 1, end, k);
  }
}
```



https://www.codecademy.com/resources/blog/content/images/2018/10/binary-search-small.gif

Binary Search: Analysis

Rules

- Break down the problem into subproblems
- Solve the sub problems
- Merge the sub problems to get desired Output
- *Note: Must be sorted*

Best-case => $T(n) = O(1)$

$$target\ is\ first\ comparison$$

Worst-case => $T(n) = O(log\ n)$

$$target\ is\ last\ comparison$$

Average-case *(in a successful search)* => $T(n) = O(log\ n)$

$$target\ is\ neither first\ nor\ last\ comparison$$

Linear v. Binary

Linear	Binary
Input data need not to be in sorted.	Input data need to be in sorted order.
Also called sequential search.	Also called half-interval search.
$T(n) = O(n)$	$T(n) = O(log\ n)$
Multidimensional array can be used	Only single dimensional array is used
Performs equality comparisons	Performs ordering comparisons
Less complex.	More complex.
Very slow process.	Very fast process.