Searching Algorithms

Design & Analysis

Analysis of:

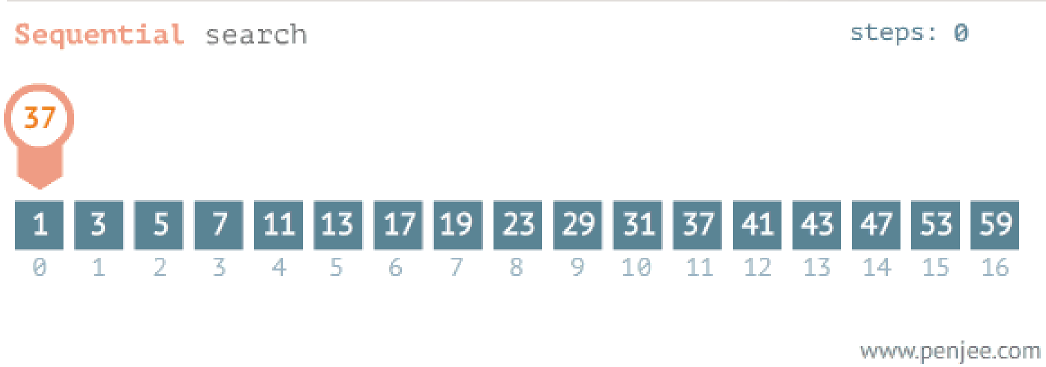
- Searching Algorithms
  - Linear Search
  - Binary Search
  - Interpolation Search

Search Problem

- Is the value K present in a collection A?
- Does the structure matter?
  - Array Vs List
- Does the organization of the information matter?
  - Values are sorted/unsorted

Linear Search(Unsorted Case)

```
function search(A,K)
  i = 0;

   while i < n and A[i] != K do
     i = i+1;

   if i < n
     return i;
   else
     return -1;
```



- **Worst case**
  - ❖Need to scan the entire sequence A
    - ✓O(n) for input sequence of A- takes linear time.
  - ❖Does not matter whether A is list or an array

Best case, Worst case, Average Case Analysis

- Best case time complexity
  - If an algorithm takes minimum amount of time to run to completion for a specific set of input
  - Linear searching - $C_{best}=1$
- Worst Case complexity
  - If an algorithm takes maximum amount of time to run to completion for a specific set of input.
  - Linear searching - $C_{worst}=n$
- Average case
  - Gives information about the behaviour of an algorithm on specific or random output
  - Linear searching - $C_{avg}=(n+1)/2$

Binary Search

- What if A is sorted?
  - Compare K with mid-point of A.
  - If mid-point of A is K, Value is found.
  - If K less than mid-point, search left half of A.
  - If K greater than mid-point, search right half of A.

Binary search

steps: 0

37

1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
low								mid								high

```
// binary search
bool BinarySearch(int key, int array[], int min, int max)
{
    if (min <= max)
    {
        int middle = (min + max)/2;

        if (key == array[middle])
            return true;
        else if (key < array[middle])
            BinarySearch(key, array, min, middle - 1);
        else if (key > array[middle])
            BinarySearch(key, array, middle + 1, max);
    }

    return false;

}
```

How long does this take?

- Each step halves the interval to search
- For an interval of size 1, the answer is immediate.
- T(n):time to search in a list of size n
  - Recurrence function
    - $T(1)=1$
    - $T(n)=1+T(n/2)$
  - Unwinding....
    - $T(n) \quad =1+T(n/2)$

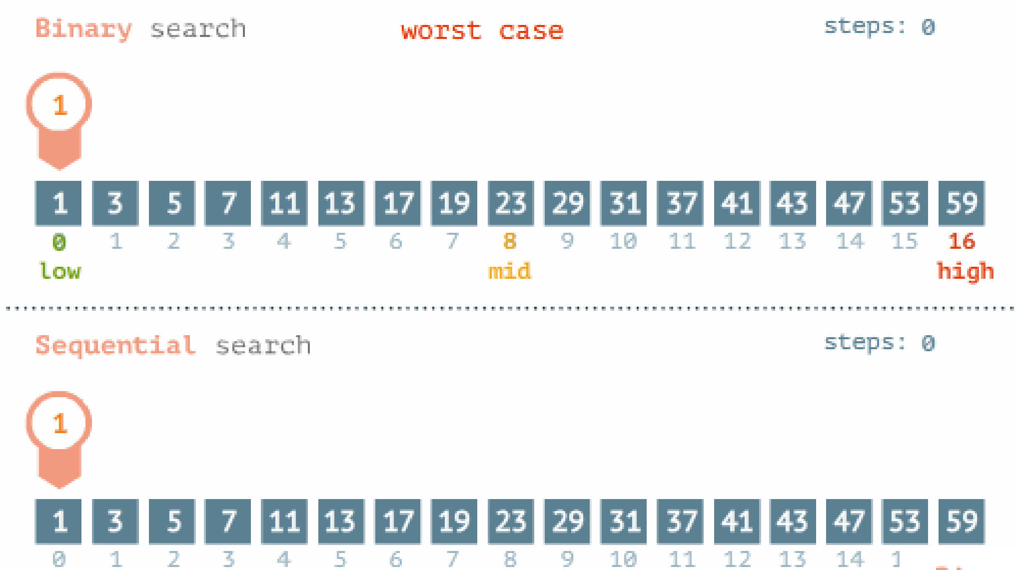
      $=1+1+T(n/4)=1+1+T(n/2^2)$

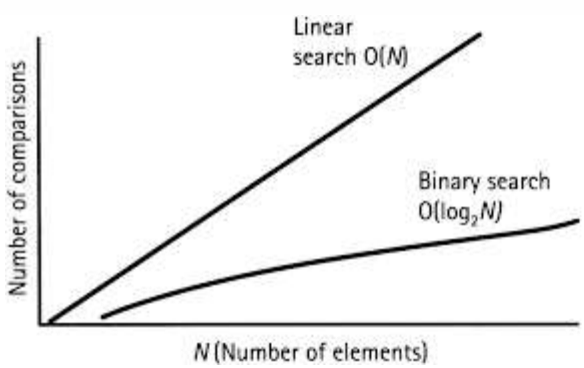
      $=1+1+1+T(n/2^3)$

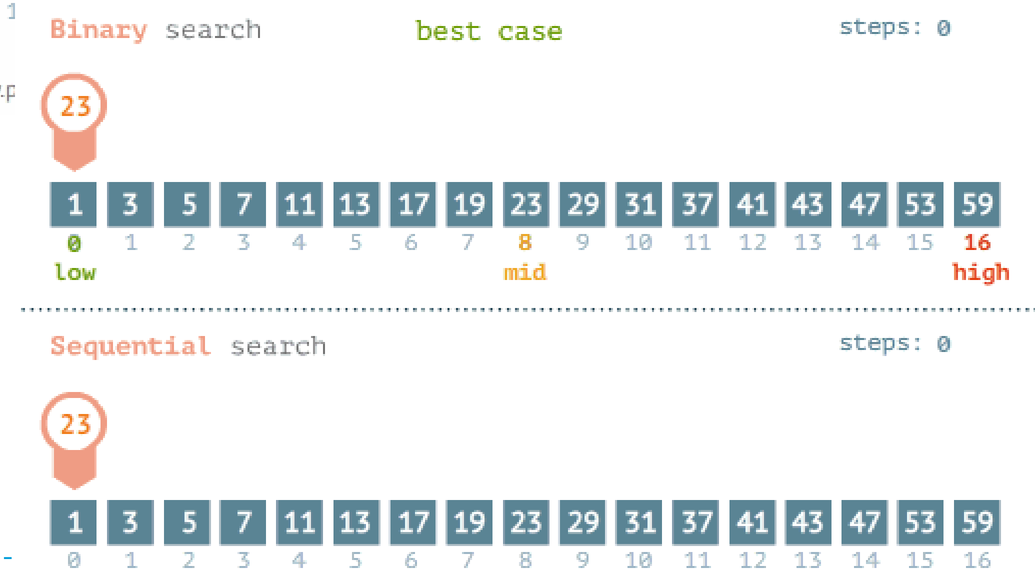
      $=1+1+...k+T(n/2^k)$

      $=O(\log(n))$

Worst Vs Best Case in Binary Search







www.penjee.com

Interpolation Search: Beating Binary Search

- Alternative to binary search that utilizes information about the underlying distribution of data to be searched.
- Binary search always splits the array in half and inspects the middle element.
- Interpolation search narrows the search space and tries to predict where the key would lie in the search space via a linear interpolation, reducing the search space to the part before or after the estimated position if the key is not found there.
- Resembles how humans search for any key in an ordered set like a word in dictionary.

Interpolation Search:

- Searching is guided by the values of the array
  - Lo: lower Index
  - Hi: Higher Index
  - search position

> ➢ h = Lo+((Hi-Lo) / (A[Hi]-A[Lo])) * x-A[lo]

- if x[h] = key element found; else search array on the left or on the right of h
  - e.g.
  - search(80): focuses on the 20% rightmost part of the array

0							100

> ➢ Binary search always goes to the middle position

Example

- Position Probing in Interpolation Search
  - Interpolation search finds a particular item by computing the probe position.
  - If a match occurs, then the index of the item is returned. To split the list into two parts, we use the following method −

```
mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])

where −
   A    = list
   Lo   = Lowest index of the list
   Hi   = Highest index of the list
   A[n] = Value stored at index n in the list
```

Algorithm-Interpolation Search

Step1: In a loop, calculate the value of “pos” using the probe position formula.

Step2: If it is a match, return the index of the item, and exit.

Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise calculate the same in the right sub-array.

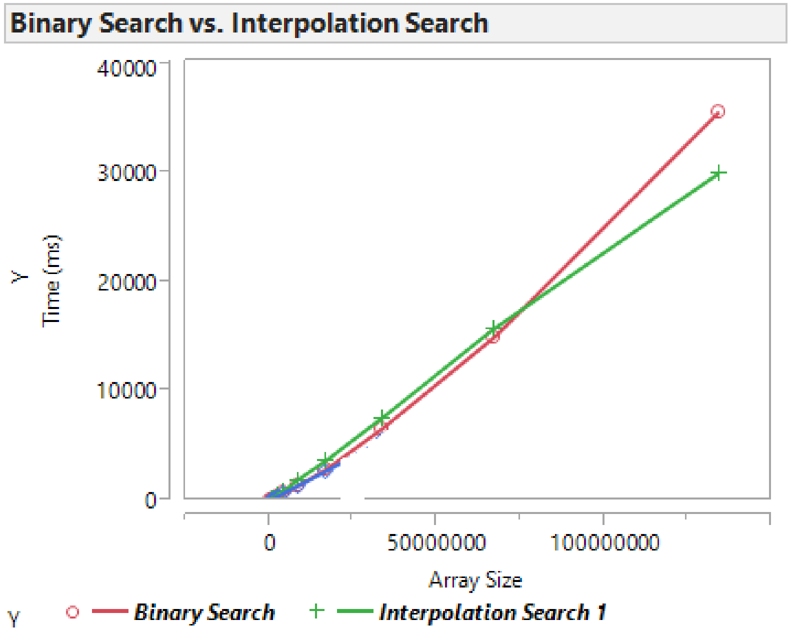
Step4: Repeat until a match is found or the sub-array reduces to zero.

Pseudo Code

```
A → Array list
N → Size of A
X → Target Value

Procedure Interpolation_Search()

   Set Lo  →  0
   Set Mid → -1
   Set Hi  →  N-1

   While X does not match

      if Lo equals to Hi OR A[Lo] equals to A[Hi]
         EXIT: Failure, Target not found
      end if

      Set Mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])

      if A[Mid] = X
         EXIT: Success, Target found at Mid
      else
         if A[Mid] < X
            Set Lo to Mid+1
         else if A[Mid] > X
            Set Hi to Mid-1
         end if
      end if

   End While

End Procedure
```

Complexity

- Average case: O(log(log n)) uniform distribution of keys in the array.
- Worst case: O(N) on non uniform distribution
- Binary search is O(log n) always!




Thank you!