Searching and Sorting

- Linear and Binary Search
- Order Statistics - MIN and MAX
- Comparison Based Sorting Algorithms
    - Selection Sort
    - Bubble Sort
    - Insertion Sort
- Lower Bound on Comparison based sorting
- Non-Comparison Based Sorting - Integers Sorting
    - Counting Sort
    - Radix Sort

IMDAD ULLAH KHAN

The Search Problem

Input: Array A of numbers, $|A| = n$ and a number x
Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

- A contains keys of the (potentially) large records
- A fundamental problem in almost all applications
- Hard to think of an application where searching is not a building block

Linear Search

Input: Array A of numbers, $|A| = n$ and a number x
Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

Linear search is the most natural solution

From left to right,
check if the current number is x $\begin{cases} \text{If yes, retrieve the element} \\ \text{else continue until the end of } A \end{cases}$

Algorithm Linear Search for x in array A

```
found ← 0
for i = 1 to n do
    if A[i] = x then
        found ← 1
        break
if found = 1 then
    return i                                ▷ to retrieve A[i].data
else
    return −1
```

Linear Search

Input: Array A of numbers, $|A| = n$ and a number x
Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

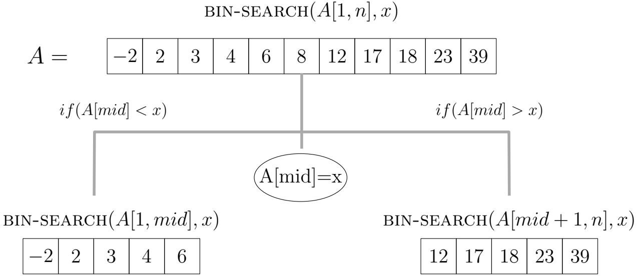
- Correctness follows from the above reasoning
- Best Case: $A[1] = x$
- Worst Case: $x \notin A$ $\triangleright$ it will compare with the whole array
- Runtime: Linear search will take $O(n)$ time in worst case
- Can we do better?
  - By the input scan argument we cannot
  - Any algorithm must at least look at every element in A, $\because$ if an element is missed from comparison, that may be x

Binary Search

Input: Sorted array A of n numbers and a number x

Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

- Compare $A[mid]$ with x
- If not equal, eliminate the half where x cannot lie
- Search x in the remaining half



BIN-SEARCH$(A[1,n],x)$

$A =$ | -2 | 2 | 3 | 4 | 6 | 8 | 12 | 17 | 18 | 23 | 39 |

$if(A[mid] < x)$

$if(A[mid] > x)$

A[mid]=x

BIN-SEARCH$(A[1,mid],x)$

| -2 | 2 | 3 | 4 | 6 |

BIN-SEARCH$(A[mid+1,n],x)$

| 12 | 17 | 18 | 23 | 39 |

Binary Search: Pseudo code

Input: Sorted array A of n numbers and a number x

Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

Algorithm Binary Search for x in sorted array $A[st, \ldots, end]$

```
function BIN-SEARCH($A$,$st$,$end$,$x$)
    if $end < st$ then                                  ▷ check if $A$ is empty
        return $-1$
    else
        $mid \leftarrow \frac{(end + st)}{2}$
        if $A[mid] = x$ then
            return $mid$                                ▷ If found return index
        else if $A[mid] > x$ then
            return BIN-SEARCH($A, st, mid - 1, x$)
        else
            return BIN-SEARCH($A, mid + 1, end, x$)
```

Binary Search

$T(n)$: time of BIN-SEARCH on $|A| = n$

Each call on $n \geq 1$ makes

- some comparisons
- plus a recursive call

$$T(n) = \begin{cases} 1 & \text{if } n < 1 \\ T(n/2) + 3 & \text{if } n \geq 1 \end{cases}$$

Recurrence Relation

$$\begin{aligned} T(n) &= T(n/2) + 3 \\ &= (T(n/4) + 3) + 3 \\ &= (T(n/8) + 3) + 3 + 3 \\ &\vdots \\ &= T(n/2^k) + \underbrace{3 + 3 \ldots + 3}_{k \text{ times}} \\ &= 1 + 3 \log n \\ &= O(\log n) \end{aligned}$$

Binary search takes $O(\log n)$ time in the worst case

Extended Binary Search

Input: Sorted array A of n numbers and a number x

Output: $$\begin{cases} \text{Index of } x \text{ in } A & \text{if } x \in A \\ \text{index of smallest element in } A \text{ larger than } x & \text{if } x \notin A \end{cases}$$

▷ if $x \notin A$, it returns an index where x would be (inserted)

- Just need to adjust the first if condition in the algorithm, where it returns -1
- Runtime of this algorithm is the same as usual BIN-SEARCH

Searching: Summary

- Searching is a fundamental problem
- Linear Search for arbitrarily ordered arrays takes $O(n)$ time
- Runtime of Linear Search matches the lower bound
- Binary Search takes $O(\log n)$ time for Sorted Array
- Binary Search can readily be extended to return the appropriate location to insert an element x in the array