CSN 102:
DATA STRUCTURES

Searching Algorithms: Linear and Binary Search

Why we need to search?

- Searching through records for a specific record or set of records
- Locating the index through searching and later retrieve data stored
- There are multiple algorithms to perform search, lets take a look to common ones:
  - Linear (Sequential) Search
  - Binary Search

Linear Search(1)

- Compare every element in list with key element
- If list[**i**] is equal to key, return **i**
- If end of list is reached and key is not found, return -1

- LinearSearch(list, n, key)

```
for i: 1 to n
        if (element i in list = key)
                return(i)
return(-1)
```

Linear Search(2)

- For successful search:
  - Best case: 1 comparison (first element in list is key)
  - Worst case: n comparison (last element in list is key)
  - Average case: (n+1)/2

- For unsuccessful search:
  - n comparison required
  - What if list is sorted? Can we improve number of comparisons if we know list is sorted?

```
LinearSearchSorted(list, n, key)
        for i: 1 to n
                if (element i in list < key)
                        continue
                else
                        break
        if (element i in list = key)
                return(i)
        else
                return(-1)
```

Linear Search(2)

- For successful search:
  - Best case: 1 comparison (first element in list is key)
  - Worst case: n comparison (last element in list is key)
  - Average case: (n+1)/2

- For unsuccessful search:
  - n comparison required
  - What if list is sorted? Can we improve number of comparisons if we know list is sorted?
  - Comparison required if list is sorted using LinearSearchSorted
    - Best case: 1 comparison (first element is greater than key)
    - Worst case: n comparison (compare till last element)
    - Average case: (n+1)/2

Linear Search(3)

- Can we devise a better searching technique for unordered list?
  - Linear search has the best performance
- Can we devise a better searching technique for ordered list?
  - Yes, Binary Search.

Binary Search(1)

- Compare middle element with key,
  - If middle element < key, reduce search to right half of list
  - If middle element > key, reduce search to left half of list
- Keep repeating the above process till key is found or size of list becomes empty
- Any constraints?
  - What if list is unordered?
    - Can’t perform binary Search
  - What if number of elements are not known?
    - Can’t find the middle element

Binary Search(2)

```
BinarySearch(list, start, end, key)
        if (start > end)
                return(-1)
        middle ← (start + end)/2
        if (middle element in list = key)
                return middle
        else if (middle element in list < key)
                return(BinarySearch(list, middle+1, end, key))
        else
                return(BinarySearch(list, start, middle-1, key))
```

Binary Search(2)

a																
	5	7	10	13	13	16	18	19	23	28	28	32	32	37	41	46
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Search for 16 in **a**; sequence of function call
  - BinarySearch(a, 0, 15, 16)

    (0+15)/2 = 7; a[7]=19;

    (19 ≠ 16) and (19 > 16) → search left half
  - BinarySearch(a, 0, 6, 16)

    (0+6)/2 = 3; a[3] = 13;

    (13 ≠ 16) and (13 < 16) → search right half
  - BinarySearch(a, 4, 6,19)

    (4+6)/2 = 5; a[5]=16;

    (16==16) → return(5)

Binary Search(3)

a																
	5	7	10	13	13	16	18	19	23	28	28	32	32	37	41	46
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Search for 17 in **a**; sequence of function call
  - BinarySearch(a, 0, 15, 17)
    (0+15)/2 = 7; a[7]=19;
    (19 ≠ 17) and (19 > 17) → search left half
  - BinarySearch(a, 0, 6, 17)
    (0+6)/2 = 3; a[3] = 13;
    (13 ≠ 17) and (13 < 17) → search right half
  - BinarySearch(a, 4, 6,17)
    (4+6)/2 = 5; a[5] = 16;
    (16 ≠ 17) and (16 < 17) → search right half
  - BinarySearch(a, 6, 6, 17)
    (6+6)/2 = 6; a[6] = 18;
    (18 ≠ 17) and (18 > 17) → search left half
  - BinarySearch(a, 6, 5, 17);
    start > end → return(-1)

Binary Search Analysis

- For 16 elements, in worst case (unsuccessful) = 4 tries
- For 32 (2^5) elements = 5 tries = $\log_2 2^5$
- For 64 (2^6) elements = 6 tries = $\log_2 2^6$
- For 120 ($2^6 < 120 < 2^7$) elements = 7 tries

- For 50000 elements ($50000 < 2^{16}$) = 16 tries ≈ $\log_2 50000$

- Really Fast!!
- $\log_2 n$ comparison required for list with n elements

Better than Binary Search?

- No algorithm exist which runs faster than binary search
- What about searching in telephone diary?
  - Indexing is another practical approach for frequent searches in a very large amount of data