Searching(1)

College of Computer Science, CQU

outline

- Searching
- Searching on unsorted arrays
- Searching on sorted arrays
- Binary search

Searching

- Search can be viewed abstractly as a process to determine if an element with a particular value is a member of a particular set.
- The more common view of searching is an attempt to find the record within a collection of records that has a particular key value, or those records in a collection whose key values meet some criterion such as falling within a range of values.

Searching: formal definition

- **Suppose that we have a collection L of n records of the form**

  $(k_1, I_1), (k_2, I_2), \circ \circ \circ, (k_n, I_n)$

  where I_j is information associated with key k_j from record j for $1 \leq j \leq n$. Given a particular key value K, the search problem is to locate a record (k_j, I_j) in L such that $k_j = K$ (if one exists). Searching is a systematic method for locating the record (or records) with key value $k_j = K$.

Searching

- A **successful search** is one in which a record with key $k_j = K$ is found.
- An **unsuccessful search** is one in which no record with $k_j = K$ is found (and no such record exists).
- An **exact-match query** is a search for the record whose key value matches a specified key value.
- A **range query** is a search for all records whose key value falls within a specified range of key values.

Searching Algorithms

- **We can categorize search algorithms into three general approaches:**
  - 1. Sequential and list methods.
  - 2. Direct access by key value (hashing).
  - 3. Tree indexing methods.

Searching on Unsorted Arrays

- **Sequential search algorithm**
  - the simplest form of search
  - Sequential search on an unsorted list requires $\Theta(n)$ time in the worst case.
- **Basic algorithm:**

  Get the search criterion (**key**)

  Get the first record from the file

  While ((record != key) and (still more records))

  Get the next record

  End_while
- **When do we know that there wasn't a record in the file that matched the key?**

Sequential Search on unsorted arrays: implementation

```
Int SeqSearch(int A[],int n,int K)
{  int i=n;
   A[0]=K;
   while(A[i]!=K) i--;
   return i;
}
```

Searching on sorted Arrays

- **Basic algorithm:**

  Get the search criterion (key)
  Get the first record from the file
  While ((record < key) and (still more records))
    Get the next record
  End_while
  If (record = key)
    Then success
    Else there is no match in the file
  End_else

- **When do we know that there wasn't a record in the file that matched the key?**

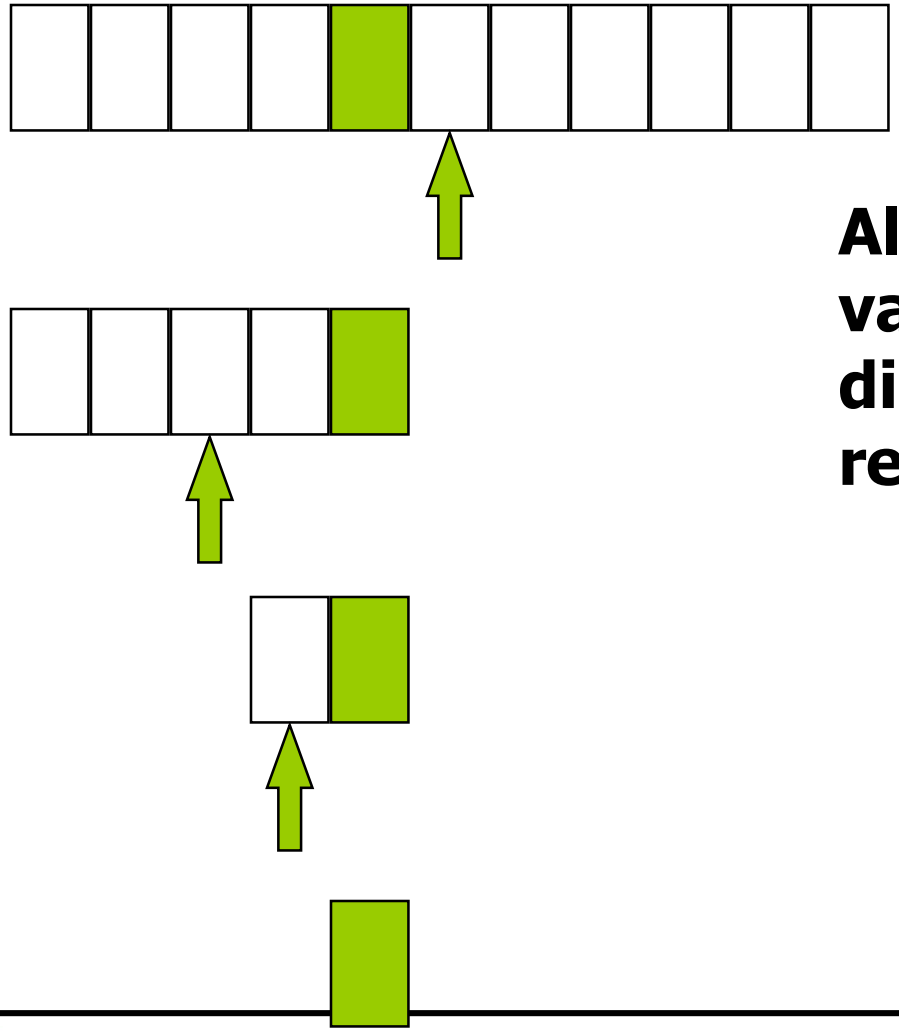
Unsorted vs. Sorted

- **Observation: the search is faster on an sorted list only when the item being searched for is not in the list.**
- **Also, keep in mind that the list has to first be placed in order for the ordered search.**
- **Conclusion: the efficiency of these algorithms is roughly the same.**
- **So, if we need a faster search, we need a completely different algorithm.**
- **How else could we search an ordered file?**

Binary search

- If we have an ordered list and we know how many things are in the list (i.e., number of records in a file), we can use a different strategy.
- The binary search gets its name because the algorithm continually divides the list into two parts.

How a Binary Search Works



Always look at the center value. Each time you get to discard half of the remaining list.

Is this fast ?

Binary search: Implementation

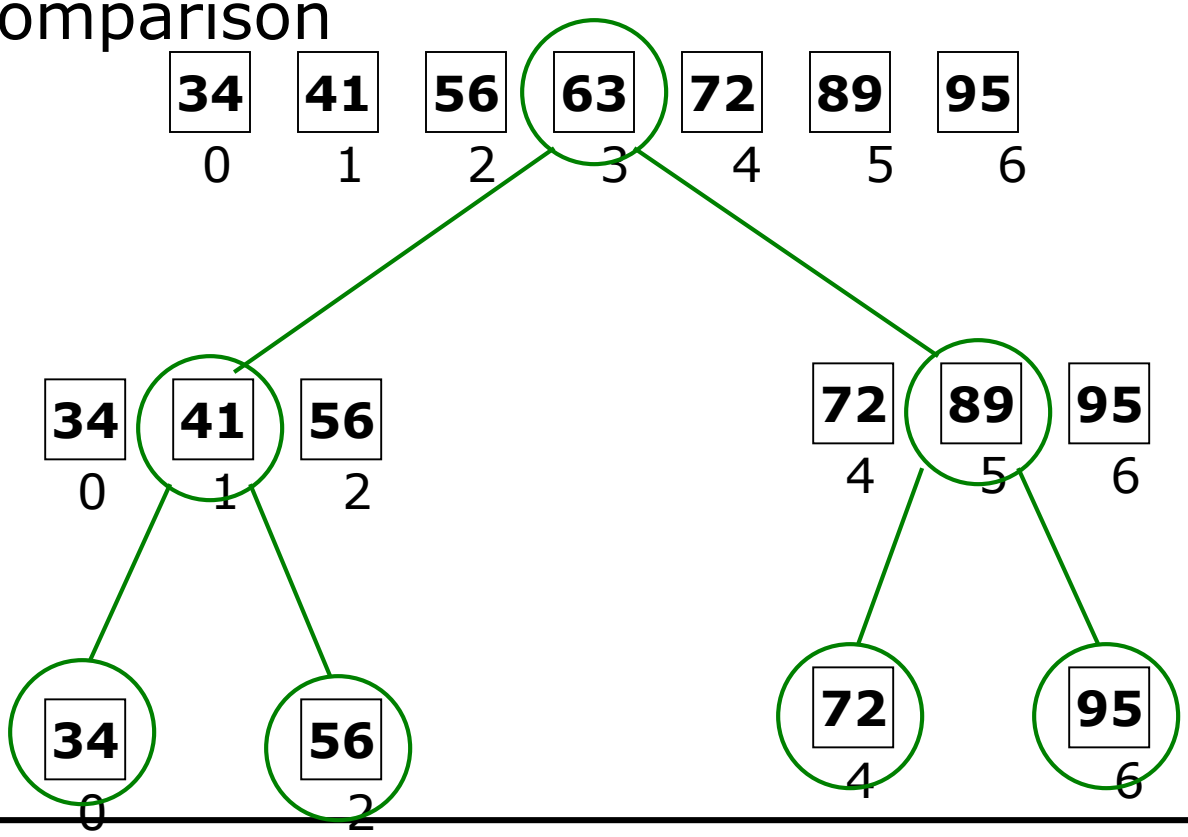
```
// Return the position of an element in sorted array "A" of
// size "n" with value "K".  If "K" is not in "A", return
// the value "n".
int binary(int A[], int n, int K) {
  int l = -1;
  int r = n;             // l and r are beyond array bounds
  while (l+1 != r) {     // Stop when l and r meet
    int i = (l+r)/2;     // Check middle of remaining subarray
    if (K < A[i]) r = i;      // In left half
    if (K == A[i]) return i; // Found it
    if (K > A[i]) l = i;      // In right half
  }
  return n; // Search value not in A
}
```

Binary search: non-recursive Implementation

```
int BinSearch1(int A[ ], int n, int k)
{
    low=1; high=n;
    while (low<=high)
    {
       mid=(low+high)/2;
       if (k<A[mid])  high=mid-1;
       else if (k>A[mid])  low=mid+1;
             else return mid;
    }
    return 0;
}
```

Binary Search Tree

Binary Search algorithm of an array of *sorted* items reduces the search space by one half after each comparison



How Fast is a Binary Search?

- **Worst case: 11 items in the list took 4 tries**
- **How about the worst case for a list with 32 items ?**
  - 1st try - list has 16 items
  - 2nd try - list has 8 items
  - 3rd try - list has 4 items
  - 4th try - list has 2 items
  - 5th try - list has 1 item

How Fast is a Binary Search? (con't)

List has 250 items

1st try - 125 items

2nd try - 63 items

3rd try - 32 items

4th try - 16 items

5th try - 8 items

6th try - 4 items

7th try - 2 items

8th try - 1 item

List has 512 items

1st try - 256 items

2nd try - 128 items

3rd try - 64 items

4th try - 32 items

5th try - 16 items

6th try - 8 items

7th try - 4 items

8th try - 2 items

9th try - 1 item

What's the Pattern?

- List of 11 took 4 tries
- List of 32 took 5 tries
- List of 250 took 8 tries
- List of 512 took 9 tries

- $32 = 2^5$ and $512 = 2^9$
- $8 < 11 < 16$ $\quad 2^3 < 11 < 2^4$
- $128 < 250 < 256$ $\quad 2^7 < 250 < 2^8$

A Very Fast Algorithm!

- **How long (worst case) will it take to find an item in a list 30,000 items long?**

$2^{10} = 1024$ $2^{13} = 8192$

$2^{11} = 2048$ $2^{14} = 16384$

$2^{12} = 4096$ $2^{15} = 32768$

- **So, it will take only 15 tries!**
- **Time complexity:** $\Theta(\log n)$

References

- **Data Structures and Algorithm Analysis Edition 3.2 (C++ Version)**
  - P.57 Example 3.1
  - P.74-75

End