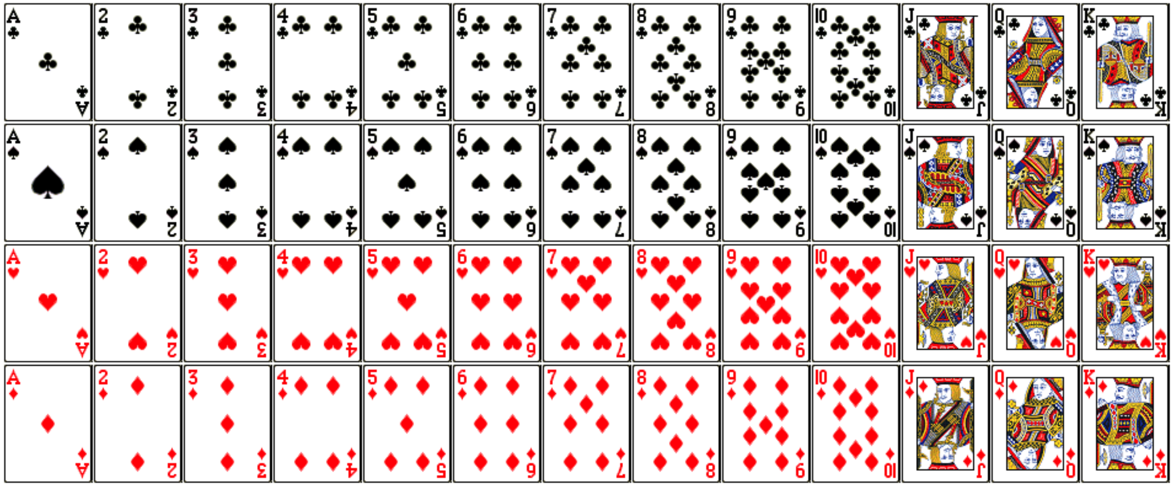
SCC.121: Fundamentals of Computer Science
Linear and Binary Searching

Amit Chopra

amit.chopra@lancaster.ac.uk

Find the six of diamonds!



Data storage and retrieval

- Fundamental operations:
  - Put data item in store
  - Retrieve specific item

- If storage is quick, retrieval will likely be slow
- If storage is careful, retrieval will be fast

Storage	Retrieval
Fast: just append latest data to the end of the store (e.g. at end of a list).	**Slow**: linear scan of store required to find item
Slow: maintain data in sorted order. Will need to find position in store where data should be placed. Depending on data structure, may need to make room (arrays vs linked list).	**Fast**: perform binary search

Theoretical Time Complexity

- The relationship between size of the input and algorithm running time
  - Let N be the size of the input
  - Characterize runtime of the algorithm as a *function* of N

- The relationship tells us, e.g., if the amount of data is doubled, does the algorithm take
  - twice as long to run? If yes, sounds OK
  - much longer than twice as long? If yes, sounds bad
  - much less than twice as long? If no, sounds good

Big-O classes

- Big O captures the *upper bound* on time required for the input
- Denoting the size of the input by *N*, algorithms are classified into big-O (or complexity) classes:

  O(1) – **constant time** (time not affected by N)

  O(log *N*) – **logarithmic time** (better than linear)

  O(*N*) – **linear time** (time proportional to N)

  O(*N* log *N*) – (worse than linear)

  O(N^2) – **quadratic time** (time proportional to N^2)

  O(N^3) – **cubic time** (time proportional to N^3)

  O(2^N) – **exponential time** (very very slow …)

Linear searching: Process

- For an unsorted linear array *A* containing *N* integers
  - find index of first occurrence of integer *x*
  - If *x* is not present in array, return -1

- Need to scan through array elements, checking for *X* and for the end of the array

```
ls(a[], x) {
   for i=0 to a.length
      if(a[i] = x)
         return i;
   return -1
}
```

Linear Search Algorithm

- Let N be the size of the input array
- Then, the foregoing linear search algorithm has complexity O(N)
  - Linear complexity
  - Runtime grows in direct proportion to input

- Can we do better?

Binary Search

- Can only be used on *sorted* arrays, but has *better theoretical time complexity* than linear searching
  - Linear search continues to have O(N) complexity even for sorted inputs

- To find *X* in array *A*, which has *N* sorted elements:

- Look at element *A*[*M*], where *M* is the mid-point of *A*;
  - if *X*=*A*[*M*]   X found
  - if size(A)=1 && X!=A[M] X not in A
  - if *X*<*A*[*M*]   X in left half (LH) of A; Repeat 1 for LH
  - if *X*>*A*[*M*]   X in right half (RH) of A; Repeat 1 for RH

Binary searching: Left and right halves

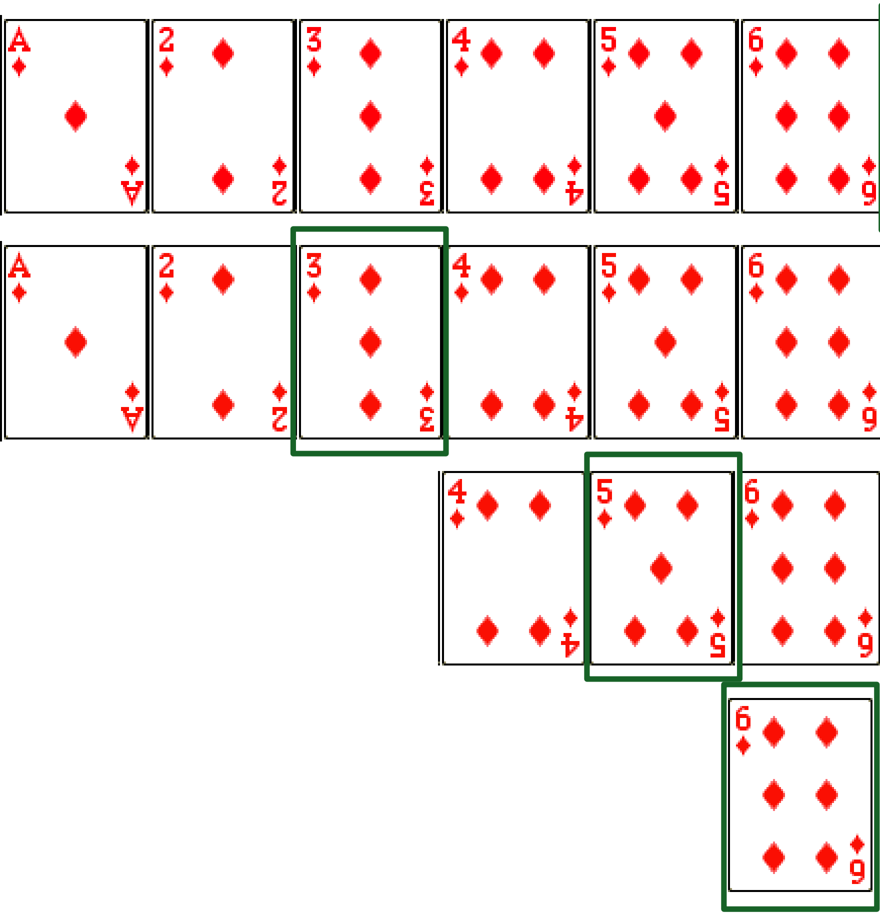
- if $X<A[M]$, look for X in *left half* of A;
- if $X>A[M]$, look for X in *right half* of A.

<---------left half--------->				M	<--------right half------→			
0	1	2	3	4	5	6	7	8

<---------left half--------->				M	<--------right half------→			
0	1	…	M-1	M	M+1	…	…	N-1

- the 'left half' consists of $A[0..M-1]$,
- the 'right half' consists of $A[M+1..N-1]$

Binary searching – Searching for 6♦



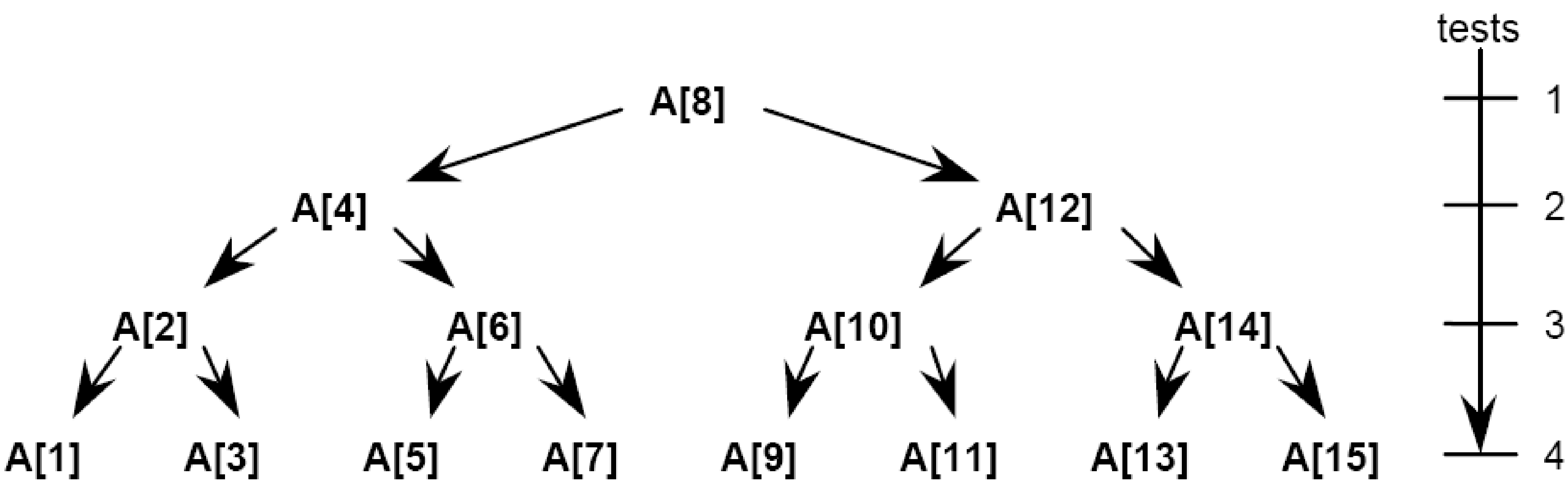
- We 'look' at 4 cards.
- How many would we look at doing a linear search?
- How many would you look at for:
  - 5♦ ?
  - 3♦ ?
  - 7♦ ?
  - 4♦ ?

Binary searching: algorithm

```
// search for X in sorted array A
int lo = 0;
int hi = N-1;
int mid;              // next element to try
while (lo<=hi)    //while subarray size >= 1
{
     mid = (lo + hi)/2;  //say, we take floor
     if (A[mid]==X) return mid;
     if (X<A[mid]) hi = mid-1;   //go left
     else lo = mid+1;            //go right
 }
return -1  //not found
```

Binary searching: O($log_2 N$)

- N=1,2,4,8,16,32,64,128,… at most 1,2,3,5,6,7,8,… comparisons, respectively; i.e., at most ($\log_2$N)+1 comparisons
- *N*=15 tree, at most 4 comparisons



- Here, A is indexed from 1
- Nodes: show array elements to be tested.
- Arrows: show next element to test after non-match.

Comparing search efficiencies

- Binary search is in complexity class O(log *N*)
- Linear search is in complexity class O(*N*)

logarithmic

linear