A typical client: Whitelist filter

A blacklist is a list of entities to be *rejected* for service.

← Examples: Overdrawn account
Spammers

A whitelist is a list of entities to be *accepted* for service.

← Examples: Account in good standing
Friends and relatives

Whitelist filter

- Read a list of strings from a *whitelist* file.
- Read strings from StdIn and write to StdOut only those in the whitelist.

Example. Email spam filter
(message contents omitted)

whitelist

alice@home
bob@office
carl@beach
dave@boat

StdIn

bob@office	✓
carl@beach	✓
marvin@spam	
bob@office	✓
bob@office	✓
mallory@spam	
dave@boat	✓
eve@airport	
alice@home	✓
...	

StdOut

bob@office
carl@beach
bob@office
bob@office
dave@boat
alice@home
...

Search client: Whitelist filter

```
public class WhiteFilter {    need an efficient search!
   public static int search(String key, String[] a)
  // Search method (stay tuned).
  public static void main(String[] args)
  {
     In in = new In(args[0]);
     String[] words = in.readAllStrings();   the whitelist
     while (!StdIn.isEmpty())
     {  //for every input string, search the words array
        String key = StdIn.readString();
        if (search(key, words) != -1)  //-1 means not found
           StdOut.println(key);
     }
  }
}
```

O(n)

```
% more white4.txt
alice@home
bob@office
carl@beach
dave@boat

% more test.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home

% java WhiteFilter white4.txt <
test.txt
bob@office
carl@beach
bob@office
bob@office
dave@boat
alice@home
```

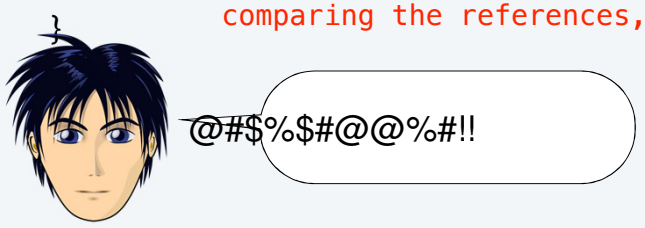
Strawman implementation: Linear search (first try)

Sequential search objects in any order in array

- Check each array entry 0, 1, 2, 3, ... for match with search string.
- If match found, return index of matching string.
- If not, return –1.

```
public static int search(String key, String[] a)
{
   for (int i = 0; i < a.length; i++)
      if (a[i] == key) return i;
   return -1;
}
```

comparing the references, NOT the strings themself!!




```
String [] words = new String[capacity];
** words is a reference to the first element of the array
** it is an array of references to string objects
```

words is the ref. to the first element

i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

oscar?

alice

words[o] contains the reference to alice

Strawman implementation: Linear search

Given an array of values, write a linear (sequential) search method to locate a specified value in the array. If the value is not present in the array, indicate that.

LO 12.1a

Sequential search

- Check each array entry 0, 1, 2, 3, ... for match with search string.
- If match found, return index of matching string.
- If not, return –1.

```
public static int search(String key, String[] a)
{
   for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(key) == 0) return i;
   return -1;
}
```

```
.compareTo() returns an integer
== 0 -> is equal
< 0 -> a[i] is smaller than key
> 0 -> a[i] is larger than key
```

```
can also use
a[i].equals(key), which
returns true or false
```

Still, this was even easier than I thought!

Match found.
Return 10

i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

oscar?

Best, worst and average case of linear search

Determine the best case, average case, and worst case Big-O analysis when a given search algorithm is executed and the search key and the array of values are provided.

LO 12.1c

Best case. Oscar is the first record (1 comparison)

Worst case. Oscar is the last record (n comparisons)

Average case. Oscar is the middle record (n/2 comparisons)

```
//objects in the array can be in any order
public static int search(String key, String[] a)
{
   for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(key) == 0) return i;    found
   return -1;    not found
}
```

O(n)

i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

oscar?

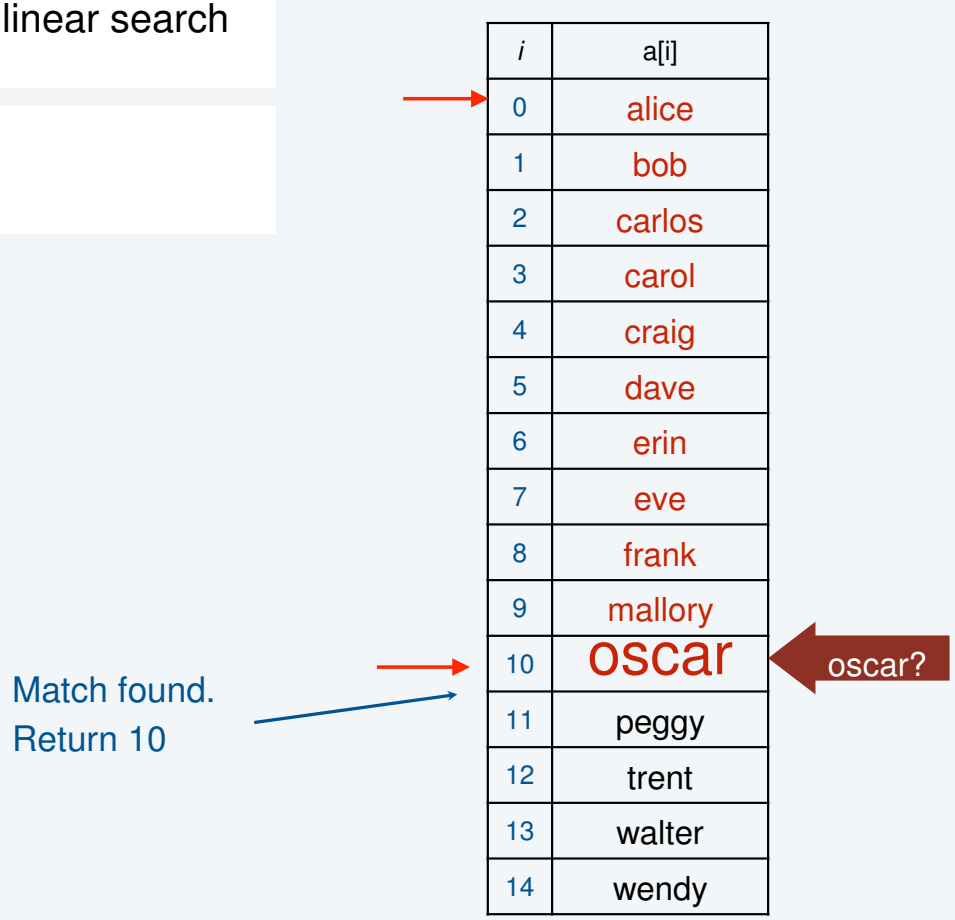
Understanding linear search

Identify all elements in a given array that are examined when a given search algorithm is executed and the search key and the array of values are provided.

LO 12.1d

Identify. All elements that are inspected during linear search

comparisons. 11 names inspected



Binary Search

Binary search

- Keep the array in sorted order (stay tuned).
- Examine the middle key.
- If it matches, return its index.
- If it is larger, search the half with lower indices.
- If it is smaller, search the half with upper indices.

```
public static int search(String key, String[] a)
{
   for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(key) == 0 ) return i;
   return -1;
}
```

11 array accesses OR
11 comparisons

Match found.
Return 10

Linear search

O(n) comparisons in worst case for searching a key
whitelist filtering -> o(n^2) doesn't scale!

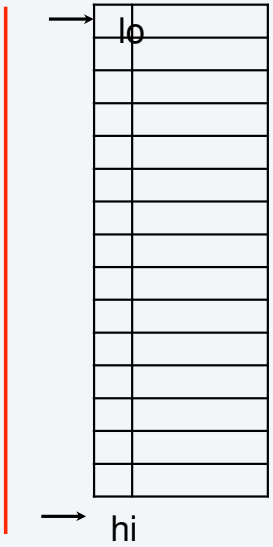
i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

oscar?

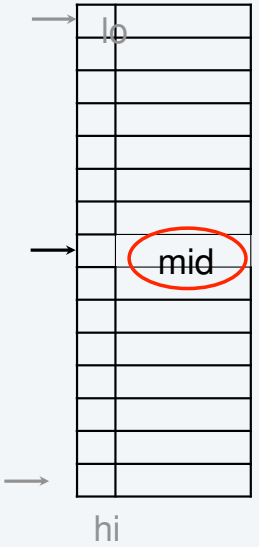
Binary search arithmetic

Notation. a[lo,hi) means a[lo], a[lo+1] ... a[hi-1] (does not include a[hi]).

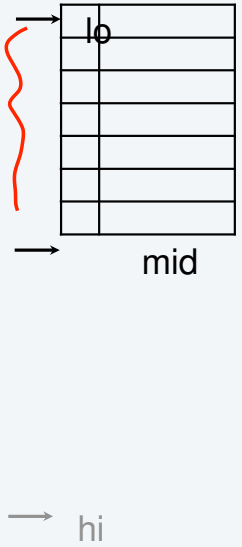
Search in a[lo,hi)



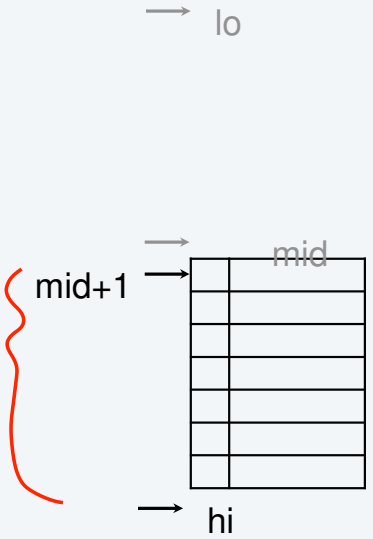
mid = lo + (hi-lo)/2 e.g., 11/2 = 5



Lower half: a[lo,mid) 0..5-1



Upper half: a[mid+1,hi) 6..11-1



~~Tricky~~! Needs study...

Binary search: Java implementation (recursive version)

Given an ordered array of values, write a non-recursive binary search method to locate a specified value in the array. If the value is not present in the array, indicate that.

LO 12.1b

```
public static int search(String key, String[] a)
{  return search(key, a, 0, a.length); }
                          for example, size of 11

public static int search(String key, String[] a, int lo, int hi)
{
   if (hi <= lo) return -1;
   int mid = lo + (hi - lo) / 2;   //mid = 5
   int cmp = a[mid].compareTo(key);
                                   0     5
   if      (cmp > 0) return search(key, a, lo, mid);   //a[mid] is greater than key; key will be in the first half of the array
   else if (cmp < 0) return search(key, a, mid+1, hi);   //a[mid] is smaller than key; key will be in the second half of the array
                                                6     11
   else              return mid;
              //found the one equals to key; method terminated
}
```

```
.compareTo() returns an integer
== 0 -> is equal
< 0 -> a[mid] is smaller than key
> 0 -> a[mid] is larger than key
```

```
//non-recursive implementation
public int binarySearch ( int[] a,
                int size, int key )
{
   int lo = 0, hi = size - 1;
   while ( lo <= hi )
   {
      int mid = ( lo + hi ) / 2;
      if ( a[mid] == key )
         return mid;
      if ( key < a[mid] )
         hi = mid - 1;
      else
         lo = mid + 1;
   }
   return -1; //not found
}
```

Still, this was easier than I thought!

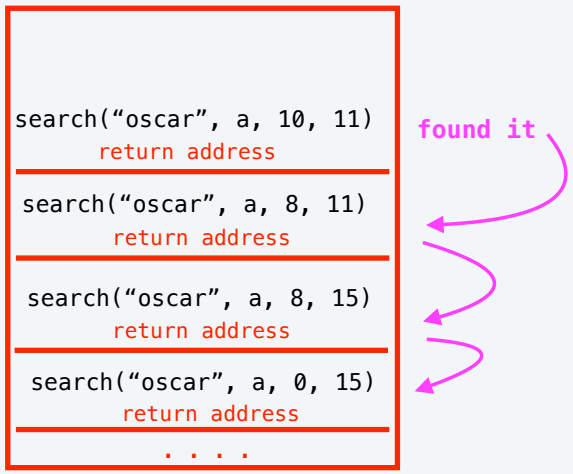
Understanding binary search

Count the number of comparisons made when a given search algorithm is executed and the search key and the array of values are provided.

LO 12.1cd

Identify. All elements that are inspected during binary search

comparisons. 4 names inspected



Match found.
Return 10

i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

oscar?

Recursion trace for binary search

```
public static int search(String key, String[] a)
{  return search(key, a, 0, a.length); }

public static int search(String key, String[] a,
                         int lo, int hi)
{
   if (hi <= lo) return -1;
   int mid = lo + (hi - lo) / 2;
   int cmp = a[mid].compareTo(key);
   if      (cmp > 0) return search(key, a, lo, mid);  //key is smaller than mid
   else if (cmp < 0) return search(key, a, mid+1, hi);
   else              return mid;                      //key is larger than mid
}
```

```
search("oscar")
   return search(... 0, 15);
```

```
search("oscar", a, 0, 15)
   mid = 7;
   > "eve"
   return search(... 8, 15);

search("oscar", a, 8, 15)
   mid = 11;
   < "peggy"
   return search(... 8, 10);

search("oscar", a, 8, 11)
   mid = 9;
   > "mallory"
   return search(... 10, 11);

search("oscar", a, 10, 11)
   mid = 10;
   == "oscar"
   return 10;
```

i	a[i]
0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	mallory
10	oscar
11	peggy
12	trent
13	walter
14	wendy

Mathematical analysis of binary search (optional)

Count the number of comparisons made when a given search algorithm is executed and the search key and the array of values are provided.

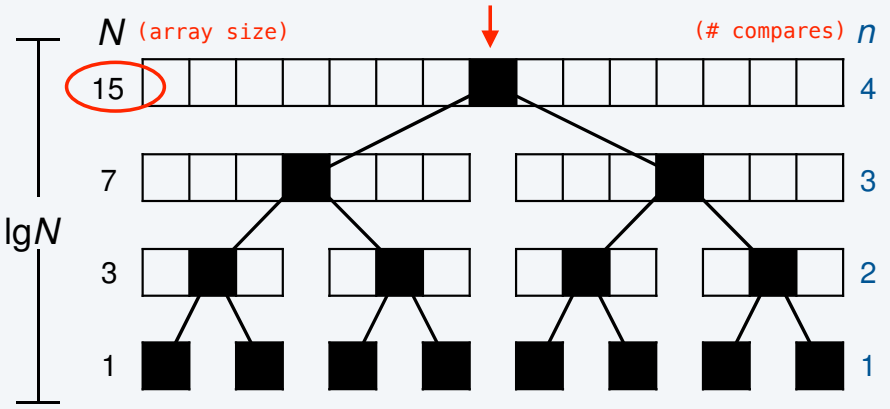
LO 12.1c

Exact analysis for search miss for $N = 2^n - 1$

- Note that $n = \lg(N+1) \sim \lg N$.
- Subarray size for 1st call is $2^n - 1$. 15
- Subarray size for 2nd call is $2^{n-1} - 1$. 7
- Subarray size for 3rd call is $2^{n-2} - 1$. 3

cut in half every call

- ...
- Subarray size for nth call is 1.
- Total # compares (one per call): $n \sim \lg N$. //log base 2



Every search miss is a top-to-bottom path in this tree.

Proposition. Binary search uses ~lg N compares for a search miss.

Not found

Proof. An (easy) exercise in discrete math.

Proposition. Binary search uses ~lg N compares for a random search hit.

found

Proof. A slightly more difficult exercise in discrete math.

Interested in details? Take a course in algorithms.