INTRODUCTION TO
COMPUTER SCIENCE

Rutgers University

Searching and Sorting



http://introcs.cs.rutgers.edu



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

Searching and Sorting

- A typical client
- Linear/sequential search
- Binary search
- Selection sort
- Insertion sort

Adopted and modified from Slides by Sedgewick and Wayne

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
{
    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;
    }

    public static void main(String[] args)
    {
        In in = new In(args[0]);
        String[] words = in.readAllStrings();

        while (!StdIn.isEmpty())
        {
            String key = StdIn.readString();
            if (search(key, words) != -1)
                StdOut.println(key);
        }
    }
}
```

```
% 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
```



Searching and Sorting

- A typical client
- Linear/Sequential search
- Binary search
- Selection sort
- Insertion sort

Strawman implementation: Linear/Sequential search (first try)

Linear/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] == key)
            return i;
    return -1;
}
```



Strawman implementation: Linear/Sequential 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.

Linear/Sequential search

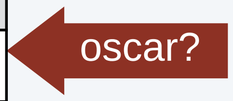
- Check each array entry 0, 1, 2, 3, ...
  for match with search string.
- If match found, return index of matching string.
- If not, return −1.

```
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;
}
```

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



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**)

```
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;
}
```

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

Mathematical analysis of whitelist filter using linear search

Model

- *N* strings on the whitelist.
- *cN* transactions (searches) for constant *c*.
- String length not long.

Analysis

- A random search *hit* checks *about half* of the *N* strings on the whitelist, on average.
- A random search *miss* checks *all* of the *N* strings on the whitelist, on average.
- Expected order of growth of running time: N^2.

whitelist

dobqi
xwnzb
dqwak
lnuqv
czpwx
bshla
idhld
utfyw
hafah
tsirv

transactions

xwnzb
lnuqv
lnuqv
czpwx
czpwx
dqwak
idhld
dobqi
dobqi
tsirv
dqwak
dobqi
idhld
dqwak
dobqi
lnuqv
xwnzb
idhld
bshla
xwnzb

Random representative inputs for searching and sorting

Generate N random strings of length L from a given alphabet

```
public class Generator
{
    public static String randomString(int L, String alpha)
    {
        char[] a = new char[L];
        for (int i = 0; i < L; i++) {
            int t = StdRandom.uniform(alpha.length());
            a[i] = alpha.charAt(t);
        }
        return new String(a);
    }

    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]); // Num strings
        int L = Integer.parseInt(args[1]); // String length
        String alpha = args[2];            // Alphabet
        for (int i = 0; i < N; i++)
            StdOut.println(randomString(L, alpha));
    }
}
```

```
% java Generator 10 3 abc
bab
bab
bbb
cac
aba
abb
bab
ccb
cbc
bab
```

good chance of duplicates

```
% java Generator 15 8 0123456789
62855405
83179069
79061047
27258805
54441080
76592141
95956542
19442316
75032539
10528640
42496398
34226197
10320073
80072566
87979201
```

not much chance of duplicates

```
% java Generator 1 60 actg
tctatagggtcgtttgcgaagcctacacaaaagtagttgttggacaacgattgacaaaca
```

Test client for linear search

Print time required for 10N searches in a whitelist of length N

```
public class TestSS
{
   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;
   }

   public static void main(String[] args)
   {
      String[] words = StdIn.readAllStrings();
      int N = words.length;
      double start = System.currentTimeMillis() / 1000.0;

      for (int i = 0; i < 10 * N; i++)
      {
         String key = words[StdRandom.uniform(N)];
         if (search(key, words) == -1)
            StdOut.println(key);
      }
      double finish = System.currentTimeMillis() / 1000.0;
      StdOut.println(Math.round(finish - start) + " seconds");
   }
}
```

a-z = abcdefghijklmnopqrstuvwxyz

```
% java Generator 10000 10 a-z | java TestSS
3 seconds
```

generate 10,000 ten-letter words (lowercase)

print time for 100,000 searches

random successful search (no output)

Empirical tests of linear search

Print time required for 10N searches in a whitelist of length N

N	T_N (seconds)	$T_N/T_{N/2}$	*transactions per second*
10,000	1		10,000
20,000	4	4	5,000
40,000	17	4.25	2,353
80,000	59	3.47	1,356
...			
1.28 million	16,384	4	78

Whitelist filter scenario

- Whitelist of size *N*.
- 10*N* transactions.

```
% java Generator 10000 ...
1 seconds
% java Generator 20000 ...
4 seconds
% java Generator 40000 ...
17 seconds
% java Generator 80000 ...
59 seconds
```

... = 10 a-z | java TestSS

more than 4.5 hours

1.28 million transactions at a rate of 78 per second and dropping

Doubling method

Hypothesis. The running time of my program is $T_N \sim a\,N^b$.

Consequence. As *N* increases, $T_N/T_{N/2}$ approaches 2^b.

no need to calculate *a* (!)

Proof: $\frac{a(2N)^b}{aN^b} = 2^b$



Validates hypothesis that order of growth is N^2.

Does NOT scale.



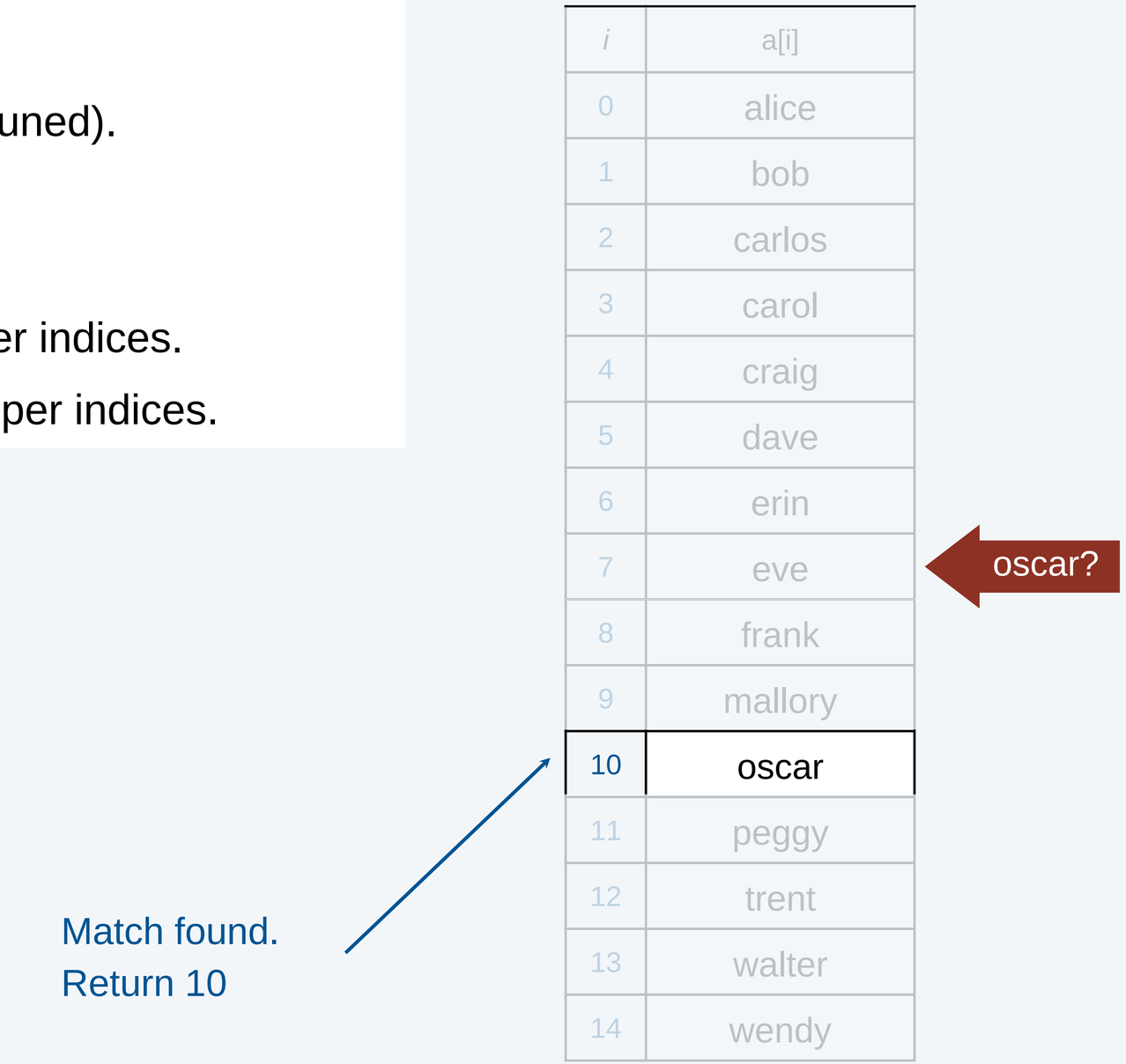
Searching and Sorting

- A typical client
- Linear/Sequential search
- Binary search
- Selection sort
- Insertion sort

Binary Search

Binary search

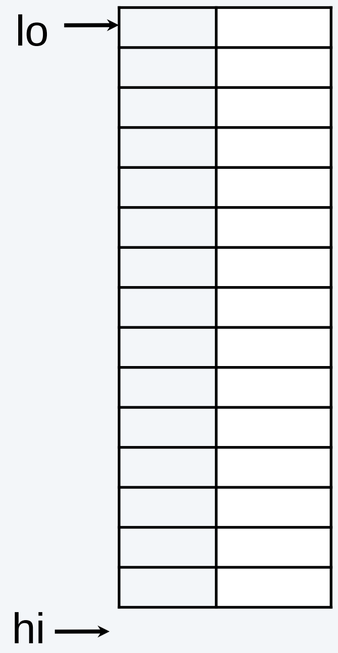
- Keep the array in sorted order (stay tuned).
- Examine the middle key.
- If it matches, return its index.
- If it is larger, search the half with lower indices.
- If it is smaller, search the half with upper indices.



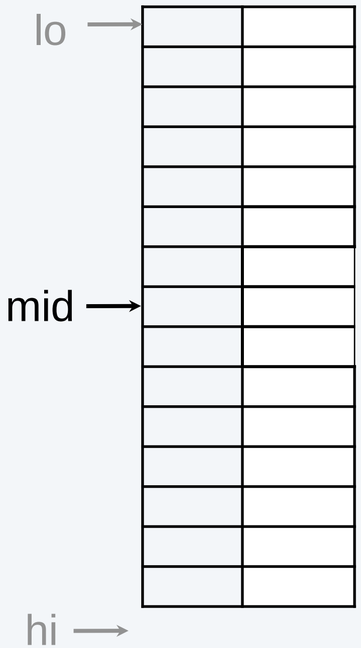
Binary search arithmetic

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

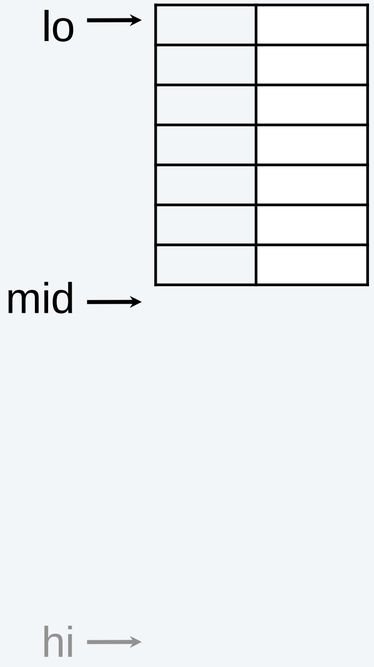
Search in a[lo,hi)



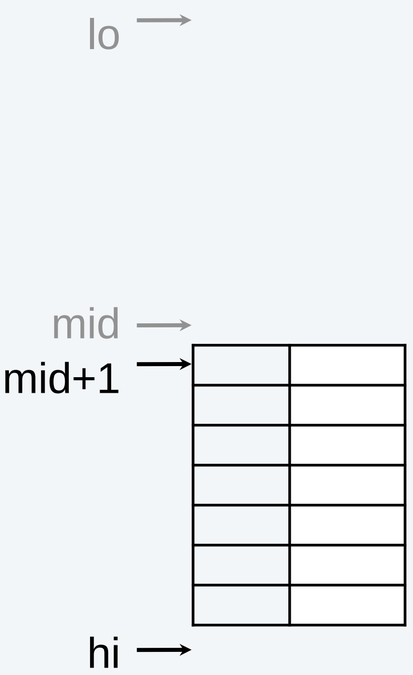
mid = (lo + hi)/2



Lower half: a[lo,mid)



Upper half: a[mid+1,hi)



Binary search: Java implementation

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

```
// Return the index of the key in the sorted array a[]
// -1 if not found
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) {
    // possible key indices in [lo, hi)
    if (hi <= lo) return -1;
    int mid = (hi + lo) / 2;

    //System.out.println(mid + " " + a[mid] + " " + key);
    int cmp = a[mid].compareTo(key);
    if (cmp > 0)
        return search(key, a, lo, mid);
    else if (cmp < 0)
        return search(key, a, mid+1, hi);
    else
        return mid;
}
```

Overload method to make calling easier

Recursive!

Binary search: Test client

```
public static void main(String[] args)
{
    In in = new In(args[0]);
    String s = in.readAll();
    String[] words = s.split("\\s+");
    System.err.println("Done reading words");

    // sort the words (if needed)
    Arrays.sort(words);
    System.err.println("Done sorting words");

    // prompt user to enter a word and check if it's there
    while (!StdIn.isEmpty()) {
        String key = StdIn.readString();
        int idx = search(key, words);
        if (idx < 0)
            StdOut.println(key + " not found");
        else
            StdOut.println("found " + key + " at " + idx);
    }
}
```

Read Whitelist

Ensure Whitelist is sorted

Search Whitelist

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

```
$ java BinarySearch names15.txt
Done reading words
Done sorting words
shea
shea not found
olivia
olivia not found
oscar
found oscar at 10
```

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

```
// return the index of the key in the sorted array a[];
// -1 if not found
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) {
    // possible key indices in [lo, hi)
    if (hi <= lo) return -1;
    int mid = (hi + lo) / 2;

    //System.out.println(mid + " " + a[mid] + " " + key);
    int cmp = a[mid].compareTo(key);
    if (cmp > 0)
        return search(key, a, lo, mid);
    else if (cmp < 0)
        return search(key, a, mid+1, hi);
    else
        return 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, 11);

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

Binary search: **Iterative** Java implementation

```
public static int searchIterative(String key, String[] a, int lo, int hi)
{
    // possible key indices in [lo, hi)
    while (hi > lo)          // While bounds have not crossed
    {
        int mid = (lo + hi) / 2;

        int cmp = a[mid].compareTo(key);
        if (cmp > 0)
            hi = mid;        // Lower hi index
        else if (cmp < 0)
            lo = mid + 1;    // Raise lo index
        else
            return mid;      // Found it
    }
    return -1;  // Not found
}
```

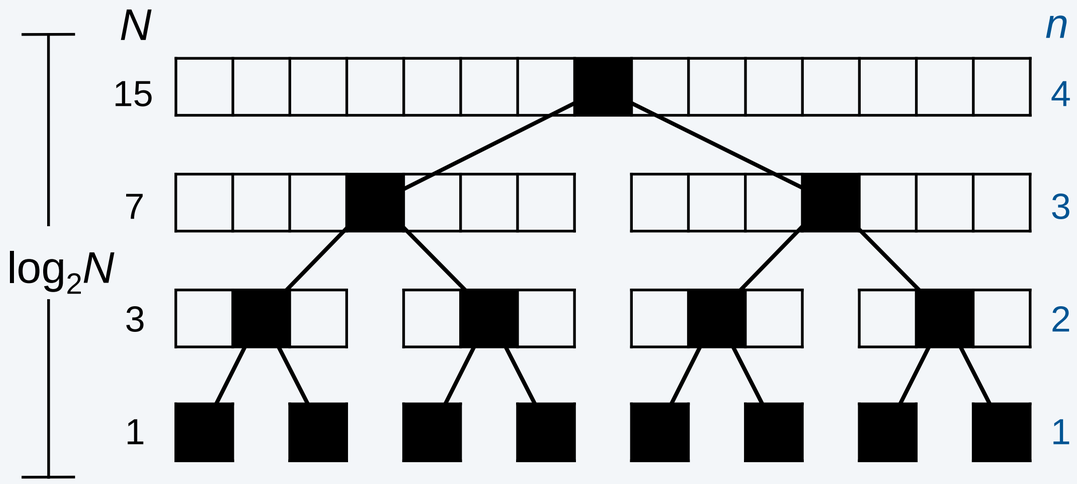
Mathematical analysis of binary search (optional)

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 $n = \log(N+1) \sim \log N$.
- Subarray size for 1st call is $2^n - 1$.
- Subarray size for 2nd call is $2^{n-1} - 1$.
- Subarray size for 3rd call is $2^{n-2} - 1$.
- ...
- Subarray size for n^{th} call is 1.
- Total # compares (one per call): $n \sim \log N$.



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

Proposition. Binary search uses ~log *N* compares for a search miss.

Proof. An (easy) exercise in discrete math.

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

Proof. A slightly more difficult exercise in discrete math.



Interested in details? Take a course in algorithms.

Worst-Case Time Complexity of Binary Search

(An informal derivation)

We start with **N items** to search

At each step we **divide the array in half**

- after one step we have ½ × N items remaining to search
- after two steps we have ½ × ½ × N = $(½)^2$ × N items remaining to search
- **after K steps we have $(½)^K$ × N items** remaining to search

We continue to search (divide the set) until we reduce the number of items to search to one

$(½)^K \times N = 1$

We want to know how steps (K) changes with N.

Rearrange to solve for K, the number of operations

$N = 2^K$

$\log_2(N) = K$

*<u>Binary Search</u> is proportional to $\log_2(N)$ operations; it has **O(logN)** worst-case time complexity*

An array of size 1000 requires 10 operations ($\log_2(1000) \approx 10$)

This is 100 times faster than sequential search, which is **O(N)**

Best, worst and average case of binary 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 12.1e

Best case.	Oscar is the “middle” record (1 comparison)	**O(1)**
Worst case.	Oscar is the last record (log N comparisons)	**O(logN)**
Average case.	Oscar is at other record (log N comparisons)	**O(logN)**

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



Empirical tests of binary search

Whitelist filter scenario

- Whitelist of size *N*.
- 10*N* transactions.

N	T_N (seconds)	$T_N/T_{N/2}$	*transactions per second*
200,000	1		200,000
400,000	2	2	200,000
800,000	6	3	133,333
1,600,000	15	2.5	106,666
3,200,000	31	2.07	103,225
…	…	…	…
10.24 million	122		83,934

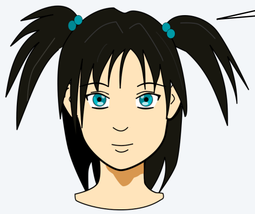
```
% java Generator 200000 ...
1 seconds
% java Generator 400000 ...
2 seconds
% java Generator 800000 ...
6 seconds
% java Generator 1600000 ...
15 seconds
% java Generator 3200000 ...
31 seconds
% java Generator 10240000 ...
122 seconds
```

```
... = 10 a-z | java TestBS
a-z = abcdefghijklmnopqrstuvwxyz
```

About 100,000 transactions per second, and holding

Validates hypothesis that order of growth is *N* log *N*.

Will scale.





Searching and Sorting

- A typical client
- Linear/Sequential search
- Binary search
- Selection sort
- Insertion sort

Selection sort

Given an array of values, give a step-by-step illustration of executing the ***selection sort*** on the array. State the array contents after each pass of the sort.

LO 12.2a

Main idea

- In iteration i, find the index min of the smallest remaining entry.
- Swap a[i] and a[min]
- Repeat for the remainder of the sequential data structure



		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Trace of selection sort (array contents just after each exchange)

Selection sort implementation (comparable objects)

Write a program that sorts an array of objects using the selection/insertion sort.

LO 12.2a

```
/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (less(a[j], a[min])) min = j;
        }
        exch(a, i, min);
    }
}
```

Sorting an array of comparable using selection sort

```
// is v < w ?
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}
```

Class of object has implemented a compareTo method

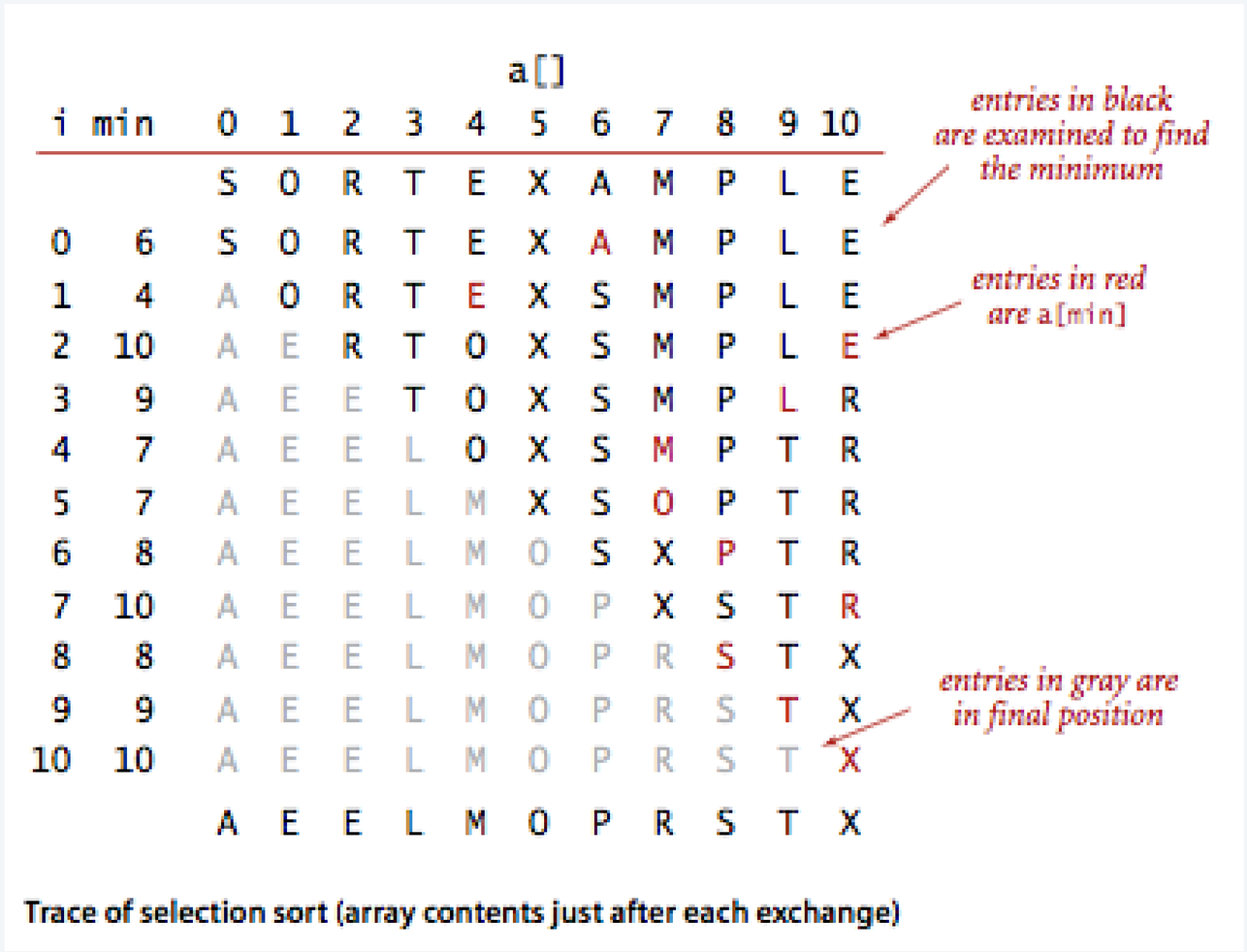
```
// exchange a[i] and a[j]
private static void exch(Object[] a, int i, int j) {
    Object swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

https://algs4.cs.princeton.edu/21elementary/Selection.java.html

Selection sort trace

Given an array of values, give a step-by-step illustration of executing the ***selection sort*** on the array. State the array contents after each pass of the sort.

LO 12.2a



Trace of selection sort (array contents just after each exchange)

Best, average and worst case of selection sort

```
/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
    {
        int min = i;
        for (int j = i+1; j < N; j++)
        {
            if (less(a[j], a[min]))
                min = j;
        }
        exch(a, i, min);
    }
}
```

Analysis

- Selection sort selects the minimum element of the array segment a[i+1…a.length-1] and swaps with element in the i^{th} location
- The array a[0..i] remains sorted at the i^{th} iteration
- The amount of work that needs to be done remains the same regardless of the values in the array ~ $O(n^2)$
- Hence best, average and worst cases are all $O(n^2)$ for selection sort

Sorting an array of N integers using selection sort

Write a program that sorts an array of numerical using the selection/insertion sort

LO 12.2e

```
public static void sort(int[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++) {

        int min = i;
        for (int j = i + 1; j < N ; j++)
        {
            if (a[j] < a[min] )
                  min = j;
        }

        int tmp = a[i];
        a[i]    = a[min];
        a[min]  = tmp;
    }
}
```

Numerical types can be compared using < > or ==

Sorting an array of objects using selection sort

Write a program that sorts an array of objects using the selection/insertion sort.

LO 12.2f

```
/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (less(a[j], a[min])) min = j;
        }
        exch(a, i, min);
    }
}
```

Sorting an array of comparable using selection sort

```
// is v < w ?
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}
```

Class of object has implemented a compareTo method

```
// exchange a[i] and a[j]
private static void exch(Object[] a, int i, int j) {
    Object swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```



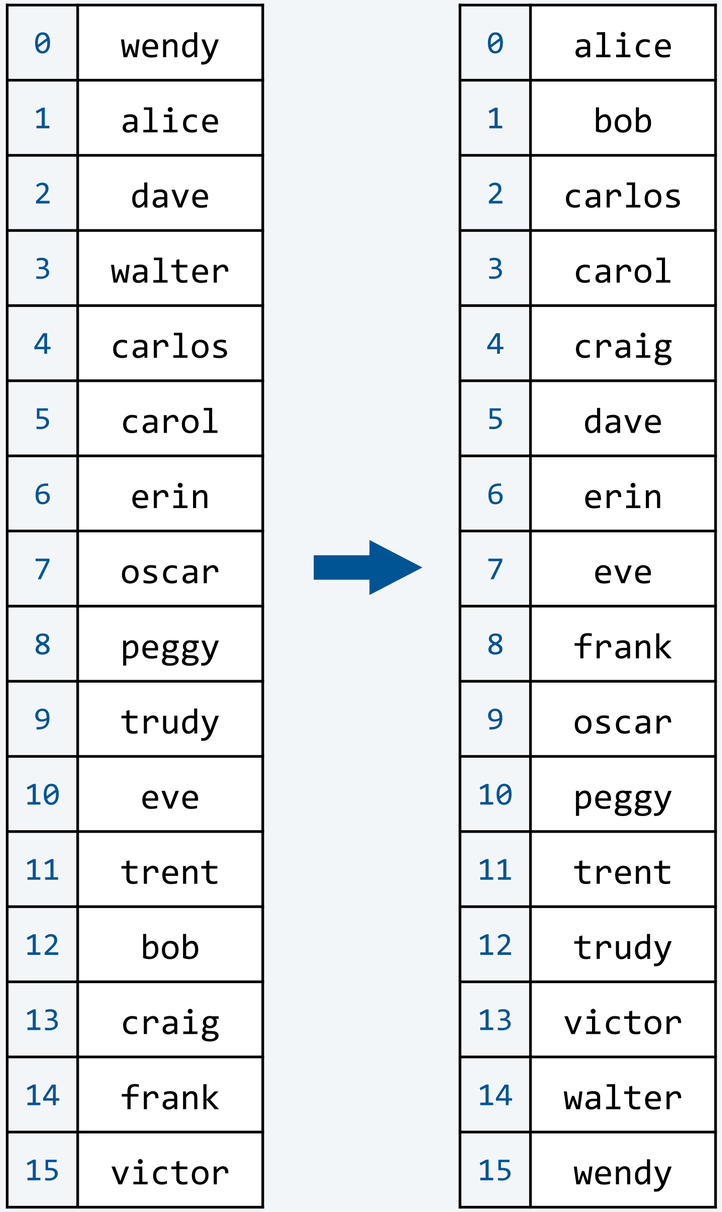
Searching and Sorting

- A typical client
- Linear/Sequential search
- Binary search
- Selection sort
- Insertion sort

Sorting: Rearrange N items to put them in ascending order

Applications

- Binary search
- Statistics
- Databases
- Data compression
- Bioinformatics
- Computer graphics
- Scientific computing
- ...
- [Too numerous to list]



0	wendy
1	alice
2	dave
3	walter
4	carlos
5	carol
6	erin
7	oscar
8	peggy
9	trudy
10	eve
11	trent
12	bob
13	craig
14	frank
15	victor

0	alice
1	bob
2	carlos
3	carol
4	craig
5	dave
6	erin
7	eve
8	frank
9	oscar
10	peggy
11	trent
12	trudy
13	victor
14	walter
15	wendy

Insertion sort algorithm

Insertion sort

- Move down through the array.
- Each item *bubbles up* above the larger ones above it.
- Everything above the current item is in order.
- Everything below the current item is untouched.

Like bubble sort, but not bubble sort.
We don't teach bubble sort any more because this is simpler and faster.

0	**wendy**
1	**alice**
2	**dave**
3	**walter**
4	**carlos**
5	**carol**
6	erin
7	oscar
8	peggy
9	trudy
10	eve
11	trent
12	bob
13	craig
14	frank
15	victor

Insertion sort trace

Given an array of values, give a step-by-step illustration of executing the ***insertion sort*** on the array. State the array contents after each pass of the sort.

LO 12.2b

0	wendy	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice	alice
1	alice	wendy	dave	dave	carlos	carlos	carlos	carlos	carlos	carlos	carlos	carlos	bob	bob	bob	bob
2	dave	dave	wendy	walter	dave	carol	carol	carol	carol	carol	carol	carol	carlos	carlos	carlos	carlos
3	walter	walter	walter	wendy	walter	dave	dave	dave	dave	dave	dave	dave	carol	carol	carol	carol
4	carlos	carlos	carlos	carlos	wendy	walter	erin	erin	erin	erin	erin	erin	dave	craig	craig	craig
5	carol	carol	carol	carol	carol	wendy	walter	oscar	oscar	oscar	eve	eve	erin	dave	dave	dave
6	erin	erin	erin	erin	erin	erin	wendy	walter	peggy	peggy	oscar	oscar	eve	erin	erin	erin
7	oscar	oscar	oscar	oscar	oscar	oscar	oscar	wendy	walter	trudy	peggy	peggy	oscar	eve	eve	eve
8	peggy	peggy	peggy	peggy	peggy	peggy	peggy	peggy	wendy	walter	trudy	trent	peggy	oscar	frank	frank
9	trudy	trudy	trudy	trudy	trudy	trudy	trudy	trudy	trudy	wendy	walter	trudy	trent	peggy	oscar	oscar
10	eve	eve	eve	eve	eve	eve	eve	eve	eve	eve	wendy	walter	trudy	trent	peggy	peggy
11	trent	trent	trent	trent	trent	trent	trent	trent	trent	trent	trent	wendy	walter	trudy	trent	trent
12	bob	bob	bob	bob	bob	bob	bob	bob	bob	bob	bob	bob	wendy	walter	trudy	trudy
13	craig	craig	craig	craig	craig	craig	craig	craig	craig	craig	craig	craig	craig	wendy	walter	victor
14	frank	frank	frank	frank	frank	frank	frank	frank	frank	frank	frank	frank	frank	frank	wendy	walter
15	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	victor	wendy

Insertion sort: Java implementation (comparable)

```
public class Insertion
{
    public static void sort(String[] a)
    {
        int N = a.length;
        for (int i = 1; i < N; i++)
            for (int j = i; j > 0; j--)
                if (a[j - 1].compareTo(a[j]) > 0)
                    exch(a, j - 1, j);
                else
                    break;
    }

    private static void exch(String[] a, int i, int j)
    {
        String t = a[i];
        a[i] = a[j];
        a[j] = t;
    }

    public static void main(String[] args)
    {
        String[] a = StdIn.readAllStrings();
        sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

```
% more names16.txt
wendy
alice
dave
walter
carlos
carol
erin
oscar
peggy
trudy
eve
trent
bob
craig
frank
victor
```

```
% java Insertion < names16.txt
alice
bob
carlos
carol
craig
dave
erin
eve
frank
oscar
peggy
trent
trudy
victor
walter
wendy
```

Empirical tests of insertion sort

Sort random strings

- Array of length *N*.
- 10-character strings.

N	T_N (seconds)	$T_N/T_{N/2}$
20,000	1	
40,000	4	
80,000	35	9
160,000	225	6.4
320,000	1019	4.5
...		
1.28 million	14400	4

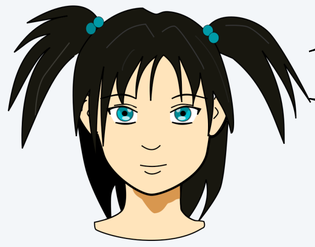
← 4 hours

```
% java Generator 20000 ...
1 seconds
% java Generator 40000 ...
4 seconds
% java Generator 80000 ...
35 seconds
% java Generator 160000 ...
225 seconds
% java Generator 320000 ...
1019 seconds
```

... = 10 a-z | java Insertion
a-z = abcdefghijklmnopqrstuvwxyz

Confirms hypothesis that order of growth is N^2.

↑ will NOT scale



And 4x64/24 = 10+ *days* to sort 10 million? Sounds bad.

Do you have anything better?

A rule of thumb

Moore's law. The number of transistors in an integrated circuit doubles about every 2 years.

Implications

- Memory size doubles every two years.
- Processor speed doubles every two years.



Gordon Moore
Founder of Intel
1929 –

computer	*instructions per second*	*words of memory*
PDP-9	tens of thousands	tens of thousands
VAX 11-780	millions	millions
CRAY 1	tens of millions	tens of millions
MacBook Air	billions	billions

Scalability

An algorithm *scales* if its running time doubles when the problem size doubles.

2x faster computer with 2x memory using an alg that scales?

- Can solve problems we're solving now in half the time.
- Can solve a 2x-sized problem in the *same* time it took to solve an x-sized problem.
- Progress.

2x faster computer with 2x memory using quadratic alg?

- Can solve problems we're solving now in half the time.
- Takes *twice* as long solve a 2x-sized problem as it took to solve an x-sized problem.
- Frustration.

order of growth	*scales?*
N	✓
$N \log N$	✓
N^2	✗
N^3	✗

Bottom line. Need algorithms that scale to keep pace with Moore's law.

Finding best, average and worst cases

Determine the best case, average case, and worst case Big-O analysis of the selection sort/insertion sort.

LO 12.2cd

Best case. Describe the performance of an algorithm under optimal conditions.

Worst case. A safe analysis of the performance of an algorithm under worst conditions.

Average case. A safe analysis of the performance of an algorithm under average conditions.

ALGORITHM	BEST	AVERAGE	WORST	REMARKS
selection sort	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges; quadratic in best case
insertion sort	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small or partially-sorted arrays
bubble sort	n	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	rarely useful; use insertion sort instead
mergesort	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable

INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

Searching and Sorting

- A typical client
- Linear/sequential search
- Binary search
- Selection sort
- Insertion sort



INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

Searching and Sorting



http://introcs.cs.rutgers.edu