

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

12. Searching and Sorting



http://introcs.cs.rutgers.edu



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

12. 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)
  // Search method (stay tuned).
  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
```

Alice and Bob



Bob

Hey, Alice. I think I'm going to start an Internet company

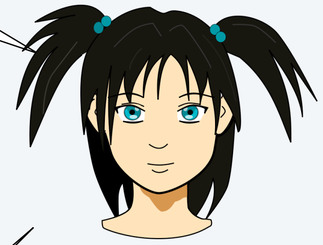
We're hoping to grow even faster than that.

Yes, I know. I'm going to a hackathon to knock it out.

Me too. I'm thinking about having 1 thousand customers next month and 1 million next year.

Good luck!
BTW, you're going to need a whitelist filter.

I'm going to take a few CS courses first.



Alice



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

12. Searching and Sorting

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

Strawman implementation: Linear search (first try)

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

loop comparing location

target not found



BUG, WHERE?

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?

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.

Works on sorted and unsorted arrays

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.

Best: O(1)
Worst: O(n)

```
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;
}
```

counts towards runningtime

also works a[i].equals(key)



Still, this was even easier than I thought!

Match found.
Return 10

SUCCESS

Best: 1 comp at index 0

Worst: last array index n comp

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

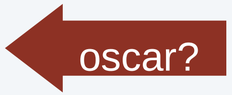
Best case. Oscar is the first record (1 comparison)

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

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

```
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
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



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

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?

Match found.
Return 10

Mathematical analysis of whitelist filter using sequential search

Model

- *N* strings on the whitelist.
- *cN* transactions 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]);
      int L = Integer.parseInt(args[1]);
      String alpha = args[2];
      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 now = System.currentTimeMillis()/1000.0;
      StdOut.println(Math.round(now-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 sequential search

Whitelist filter scenario

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

N	T_N (seconds)	$T_N/T_{N/2}$	*transactions per second*
10,000	3		3,333
20,000	9		2,222
40,000	35	3.9	1,143
80,000	149	4.3	536
...			
1.28 million	38,500	4	34

```
% java Generator 10000 ...
3 seconds
% java Generator 20000 ...
9 seconds
% java Generator 40000 ...
35 seconds
% java Generator 80000 ...
149 seconds
```

... = 10 a-z | java TestSS

more than 10.5 hours

1.28 million transactions at a rate of 34 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.



12. Searching and Sorting

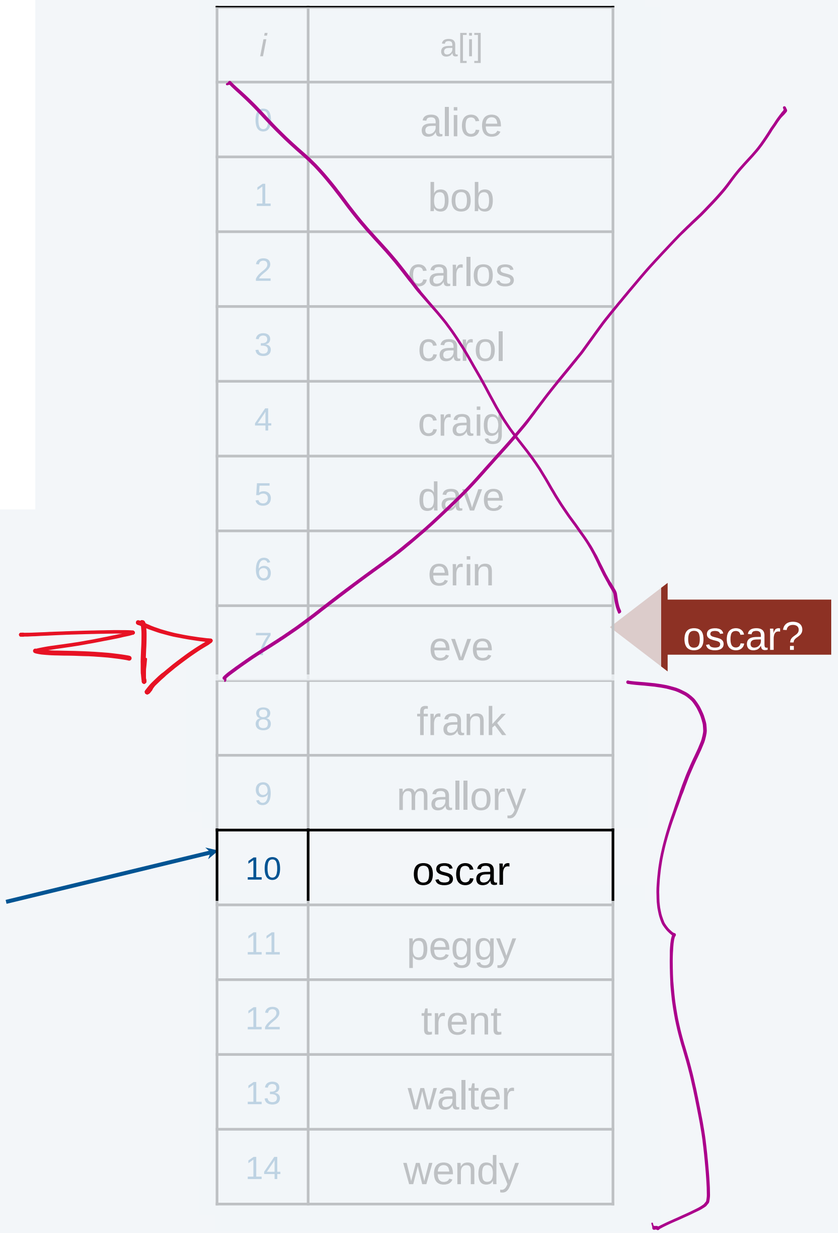
- A typical client
- Linear/Sequential search
- Binary search
- Selection sort
- Insertion sort

Binary Search

array was to be sorted

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.



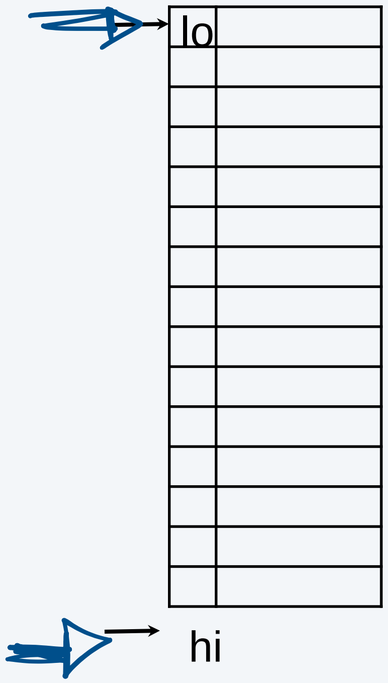
Match found.
Return 10

Binary search arithmetic

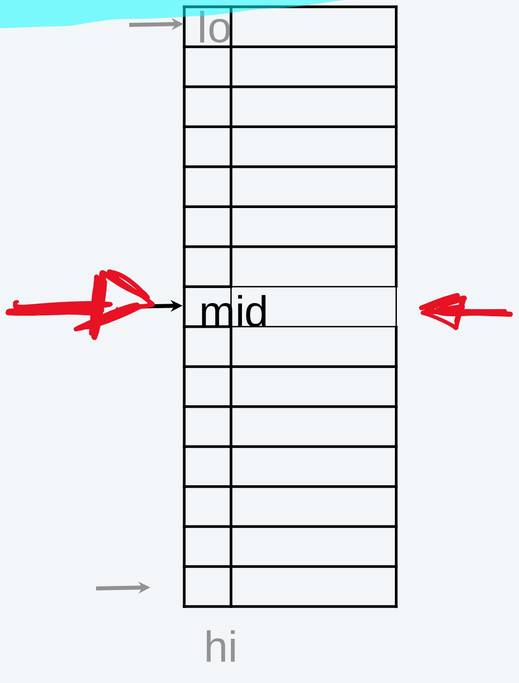
a[lo, hi) => a[2, 7) : 2, 3, 4, 5, 6

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

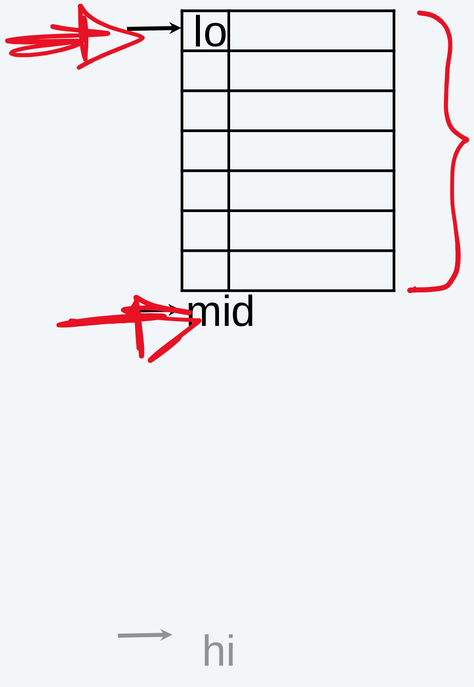
Search in a[lo,hi)



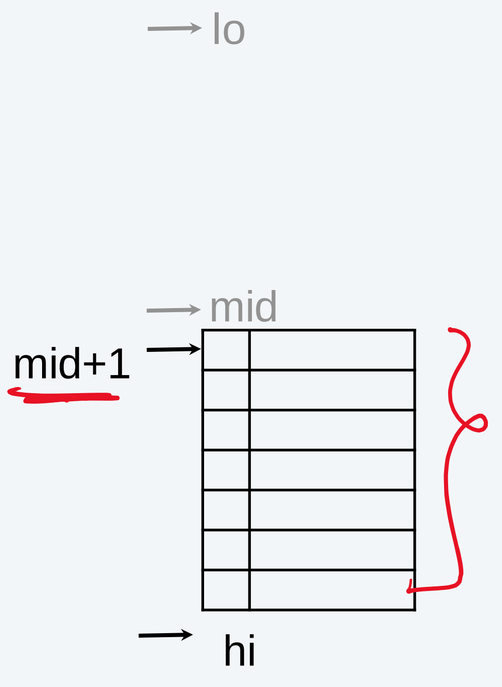
mid = lo + (hi-lo)/2

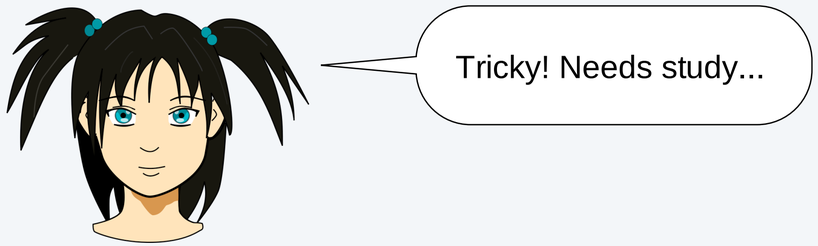


Lower half: a[lo,mid)



Upper half: a[mid+1,hi)





Binary search: Java implementation

a [lo, hi)

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 target, String[] a) {

  int hi = a.length; // hi is not included in the search

  int lo = 0;

  while (hi > lo) {

     int mid = lo + (hi - lo) / 2;

     int cmp = a[mid].compareTo(target);

     if  (cmp > 0) {

       hi = mid; // a[mid] is greater than target

     } else if (cmp < 0) {

       lo = mid + 1; // a[mid] is smaller than target

     }  else   return mid;

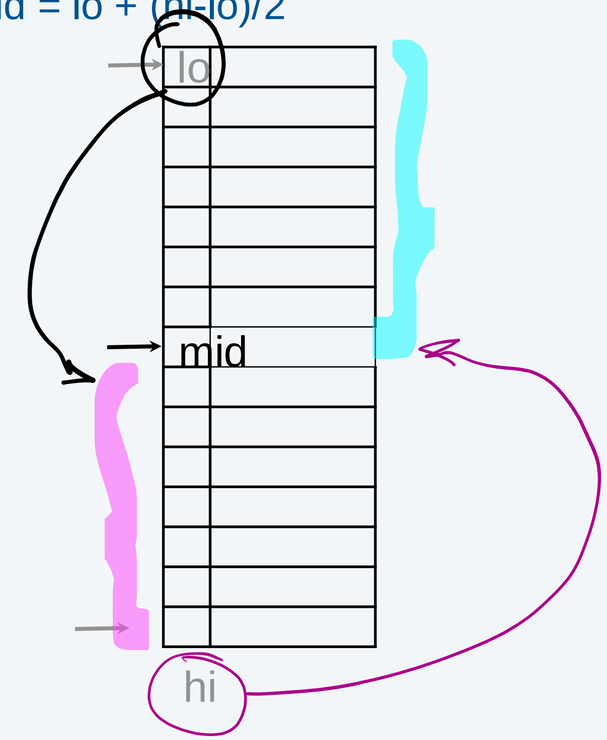
  }

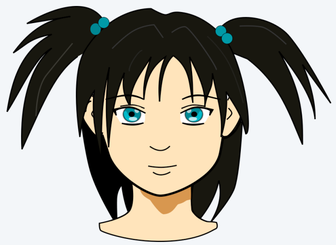
}
```

compute middle

compare

mid = lo + (hi-lo)/2





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

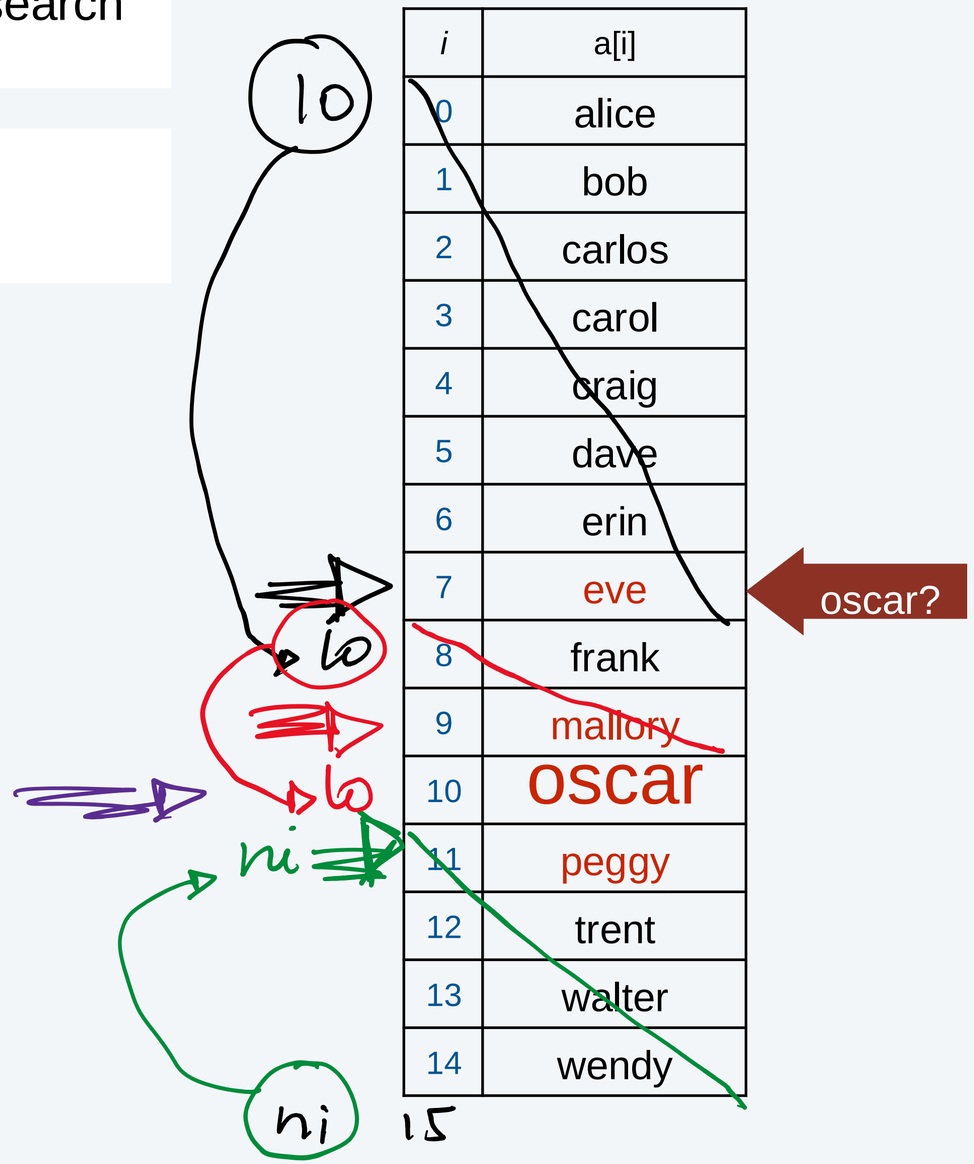
mid = lo + (hi-lo) / 2

$= 0 + (15-0)/2 \Rightarrow 7$

$= 8 + (15-8)/2 \Rightarrow 11$

$= 8 + (11-8)/2 \Rightarrow 9$

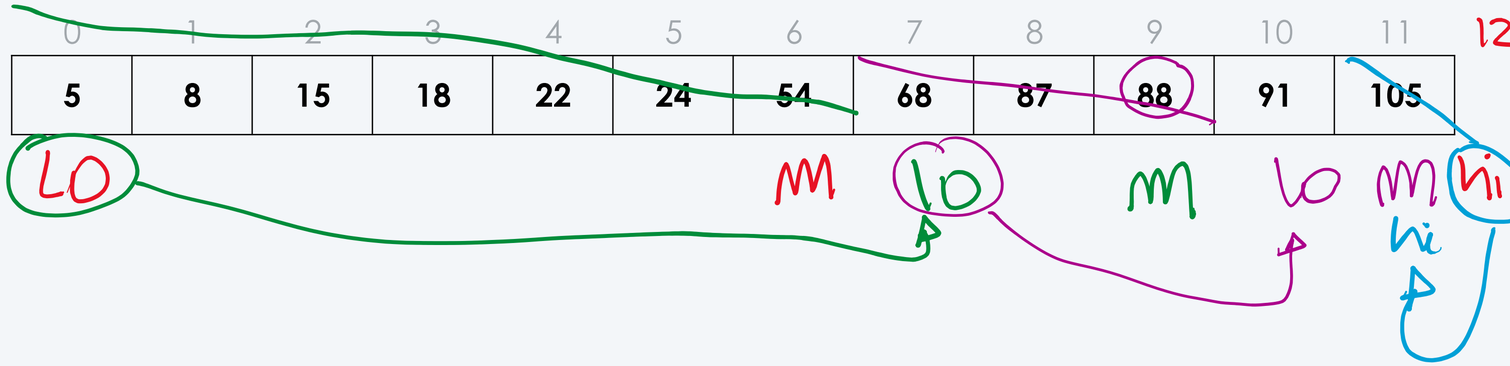
$= 10 + (11-10)/2 \Rightarrow 10$



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

Another example – target is 91

```
public static int search (String target, String[] a) {
 int hi = a.length; // hi is not included in the search
 int lo = 0;
 while (hi > lo) {
    int mid = lo + (hi - lo) / 2;
    int cmp = a[mid].compareTo(target);
    if  (cmp > 0) {
       hi = mid; // a[mid] is greater than target
    } else if (cmp < 0) {
        lo = mid + 1; // a[mid] is smaller than target
    }  else return mid;
 }
return -1; // target not found
}
```



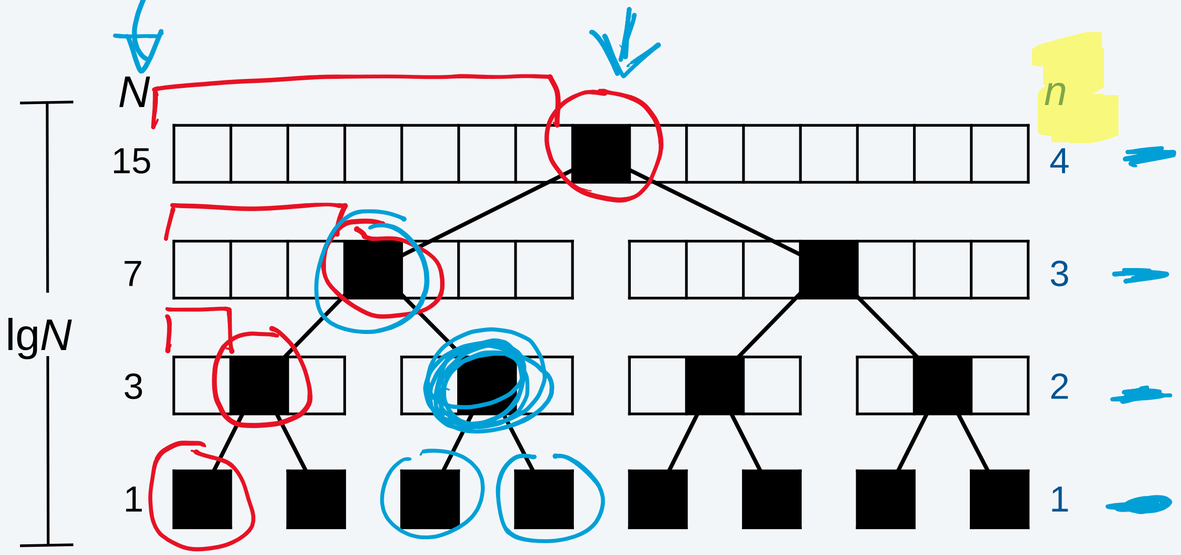
Mathematical analysis of binary search (optional)

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 nth 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.

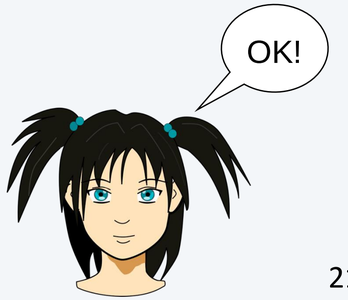
Proof. An (easy) exercise in discrete math.

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

Proof. A slightly more difficult exercise in discrete math.



Interested in details? Take a course in algorithms.



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 first record (log N comparison)

Worst case. Oscar is the last record (log N comparisons)

Average case. Oscar is the “middle” record (log N comparisons)



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*
100,000	1		
200,000	3		
400,000	6	2	67,000
800,000	14	2.35	57,000
1,600,000	33	2.33	48,000
10.28 million	264	2	48,000

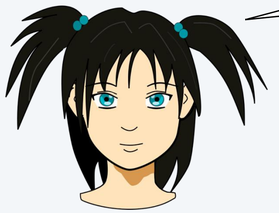
```
% java Generator 100000 ...
1 seconds
% java Generator 200000 ...
3 seconds
% java Generator 400000 ...
6 seconds
% java Generator 800000 ...
14 seconds
% java Generator 1600000 ...
33 seconds
```

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

nearly 50,000 transactions per second, and holding

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

Will scale.





INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

12. Searching and Sorting

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

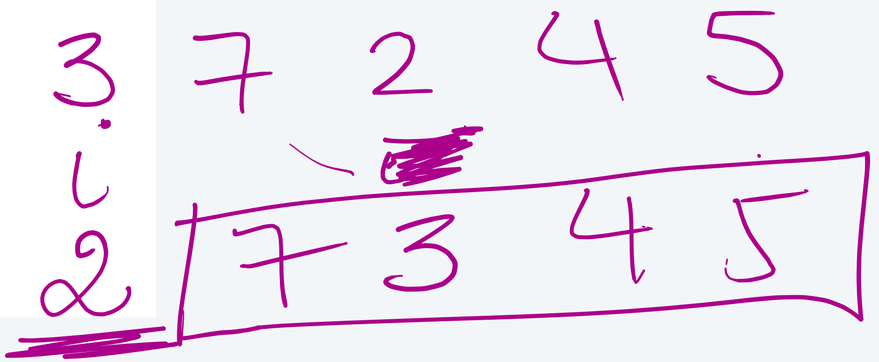
Selection sort

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]





		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

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

LO 12.2a

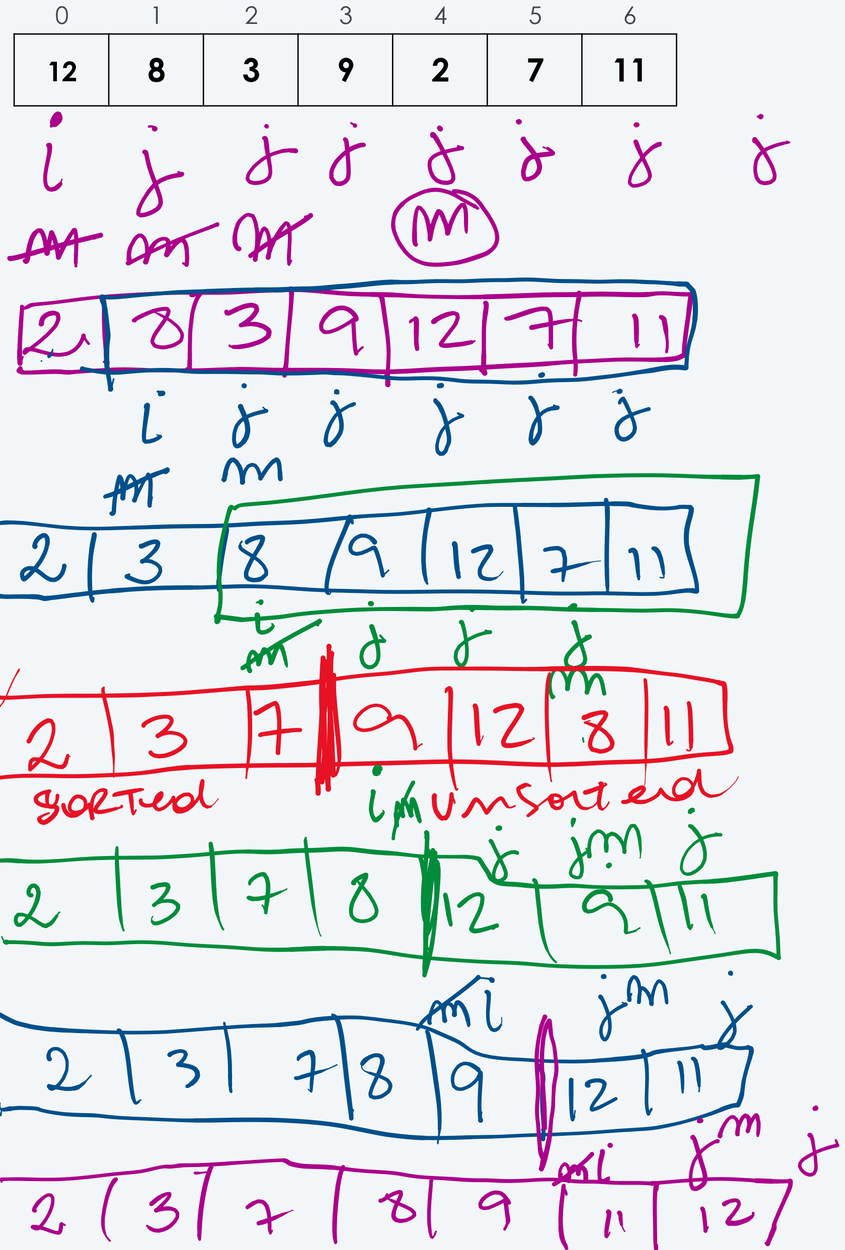
```
public static void selectionSort(int[] a) {

    int n = a.length;

    // i is the start of the unsorted region
    // all items before i are on their final position
    for ( int i = 0; i < n; i++ ) {

        // find the smallest value between i and n-1
        int min = i;
        for (int j =  i + 1; j < n;  j++) {
            if (a[j] < a[min] )
                min = j;
        }
        // swap a[i] with a[min]
        swap(a, i, min);
        print(a);
    }
}
```

```
/*
 * Swaps the value of a[i] with a[j]
 */
public static void swap (int[] a, int i, int j) {
    int tmp = a[i];
    a[i]  = a[j];
    a[j] = tmp;
}
```



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

		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)

Best, average and worst case of selection sort (number of array accesses)

```java
public static void selectionSort(int[] a) {

    int n = a.length;

    // i is the start of the unsorted region
    // all items before i are on their final position
    for ( int i = 0; i < n; i++ ) {

        // find the smallest value between i and n-1
        int min = i;
        for (int j =  i + 1; j < n;  j++) {
            if (a[j] < a[min] )
                min = j;
        }
        // swap a[i] with a[min]
        swap(a, i, min);
        print(a);
    }
}
```

i	j	j
0	3	n-1
1	2	n-2
2	1	n-3
3		n-4
…	…	…
n-1	0	0

SUM this column for the number of array accesses

Arithmetic Series

N=6

f(n) = 0 + 1 + 2 + 3 + 4 + 5 = (0 + 5) * 3 = 15

Number of pairs

Sum of one pair

f(n) = 0 + 1 + 2 + 3 + … + n-1 $\Rightarrow$ $(n-1+0) * \frac{n}{2} \Rightarrow \frac{n^2-n}{2}$

$\frac{n^2-n}{2} * 2 = n^2 - n \Rightarrow O(n^2)$

two array accesses

Analysis

- Selection sort selects the **<u>minimum</u>** 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



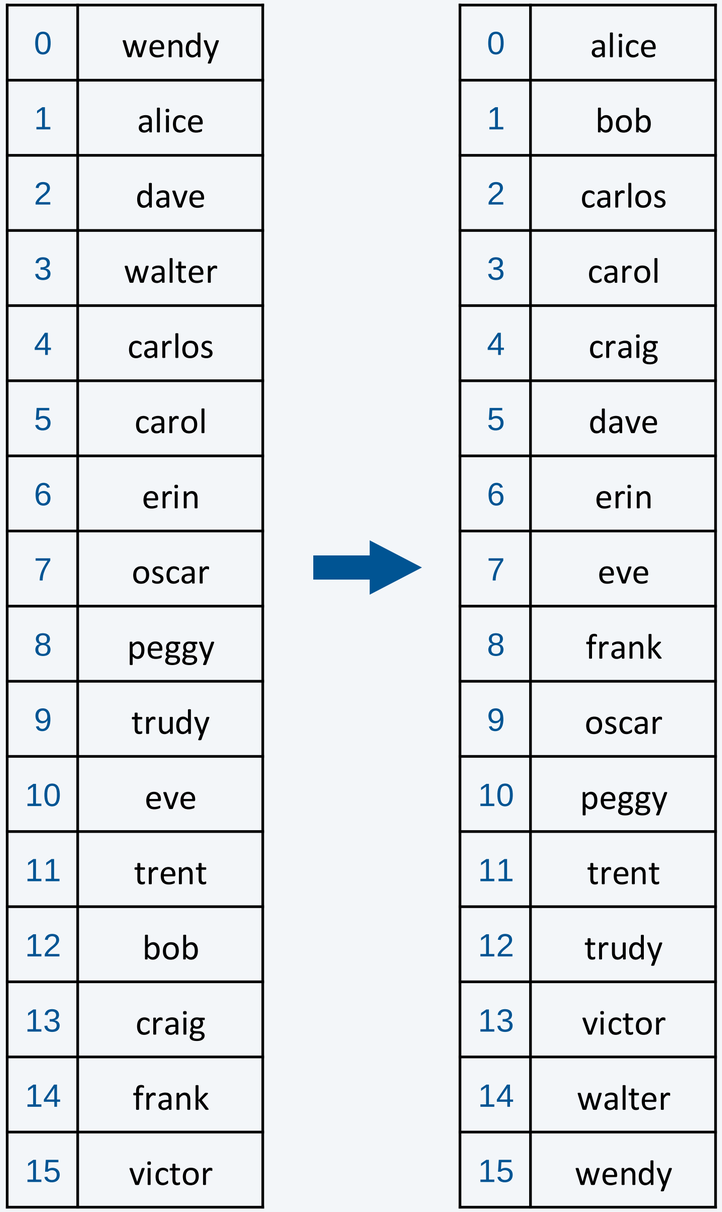
12. Searching and Sorting

- 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 anymore 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

```java
public static void insertionSort (int[] a) {

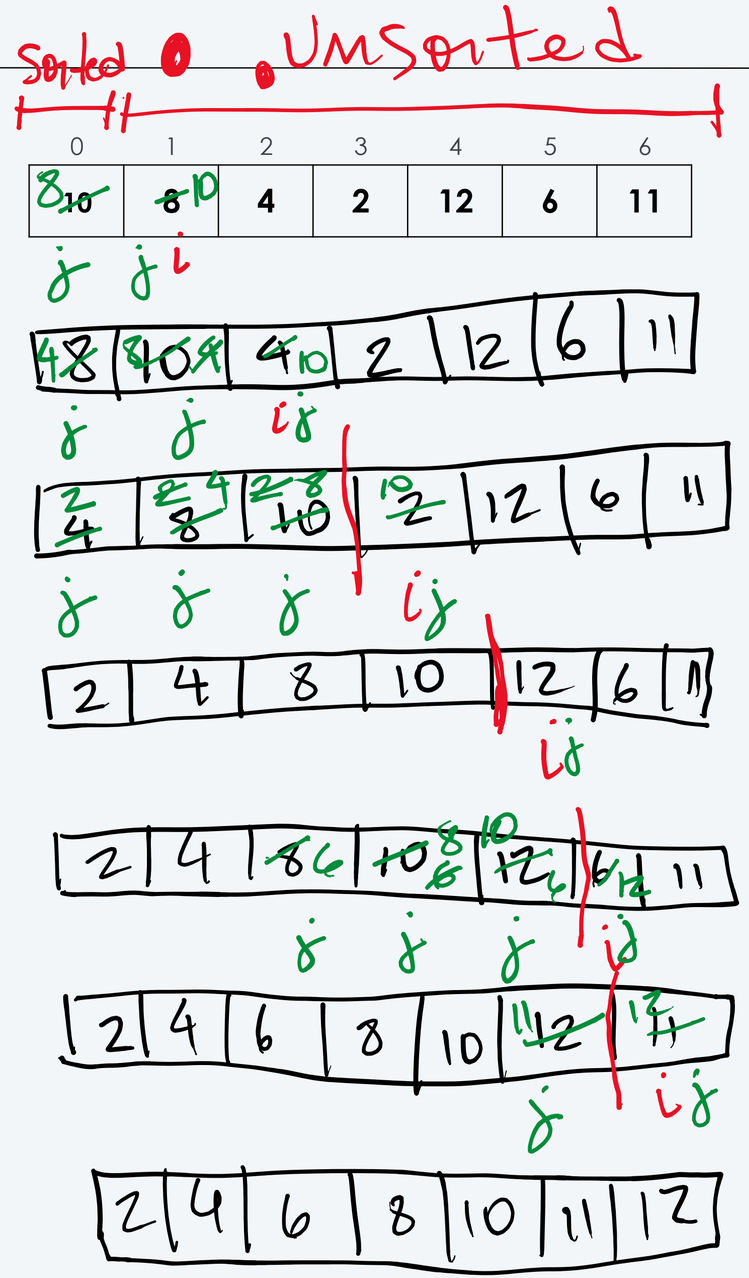
    int n = a.length;

    // i marks the beginning of the unsorted region
    for (int i = 1; i < n; i++) {

        for (int j = i; j > 0; j--) {

            if ( a[j-1] > a[j] ) {
                // swap a[j-1] with a[j]
                swap(a, j, j-1);
            } else {
                break;
            }
        }
        print(a);
    }
}
```

```java
/*
 * Swaps the value of a[i] with a[j]
 */
public static void swap (int[] a, int i, int j) {
    int tmp = a[i];
    a[i]  = a[j];
    a[j] = tmp;
}
```



Insertion sort – Best and Worst case analysis (number of array accesses)

```java
public static void insertionSort (int[] a) {

    int n = a.length;

    // i marks the beginning of the unsorted region
    for (int i = 1; i < n; i++) {

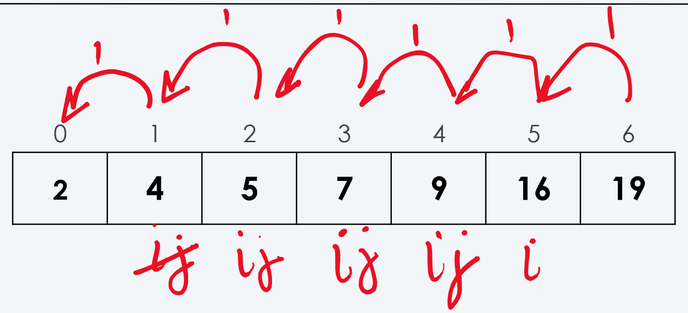
        for (int j = i; j > 0; j--) {

            if ( a[j-1] > a[j] ) {
                // swap a[j-1] with a[j]
                swap(a, j, j-1);
            } else {
                break;
            }
        }
        print(a);
    }
}
```

```java
/*
 * Swaps the value of a[i] with a[j]
 */
public static void swap (int[] a, int i, int j) {
    int tmp = a[i];
    a[i]  = a[j];
    a[j] = tmp;
}
```

array is already sorted

array is in reverse order



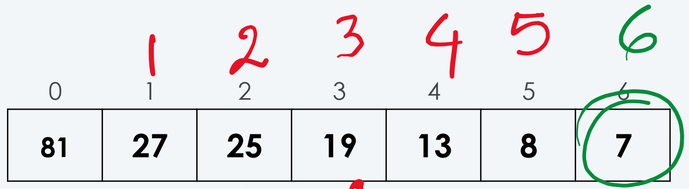
i	j
0	
1	
2	
3	
...	...
n-1	

$n-1 * 2 = 2n-2$ cmps

$\Rightarrow O(n)$

Best-case

$f(n) = 0+1+2+3+\dots+n-1 \Rightarrow \frac{n^2-n}{2} * 2 = n^2-n = O(n^2)$



i	j
0	
1	
2	
3	
...	...
n-1	

27 81
25 25
27

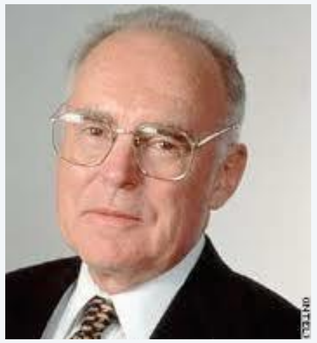
81

19 25 27 81
13 19 25 27 81
8 13 19 25 27 81

Worst-case

A rule of thumb

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



Gordon Moore
Founder of Intel
1929 –

Implications

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

Sedgewick's rule of thumb. It takes *a few seconds* to access every word in a computer.

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

Sorting an array of N integers using insertion 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;
        assert (isSorted(a, 0, i));
    }
```

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);
        assert isSorted(a, 0, i);
    }
    assert isSorted(a);
}
```

Objects must be compared using a specific function written to compare the objects of that type

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

Class of object has implemented a compare to 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;
}
```

Image sources

Image sources

https://openclipart.org/detail/25617/astrid-graeber-adult-by-anonymous-25617

https://openclipart.org/detail/169320/girl-head-by-jza

Image sources

https://www.youtube.com/watch?v=k4RRi_ntQc8

INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

12. Searching and Sorting

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



INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

12. Searching and Sorting



http://introcs.cs.rutgers.edu