CS121:
Sorting and Searching Algorithms

John Magee
24 April 2012

Sorting: Overview/Questions

- What is sorting?
- Why does sorting matter?
- How is sorting accomplished?
- Why are there different sorting algorithms?
- On what basis should we compare algorithms?

Sorting

Sorting

Arranging items in a collection so that there is a natural ordering on one (or more) of the fields in the items.

Sort Key

The field (or fields) on which the ordering is based.

Sorting Algorithms

Algorithms that order the items in the collection based on the sort key.

3

Why Does Sorting Matter?

We as humans have come to expect data to be presented with a natural ordering (e.g. alphabetic, numeric, etc.).

Finding an item is much easier when the data is sorted. *Why?*

4

Why Does Sorting Matter?

Sorting is an operation which takes up a lot of computer cycles.

How many cycles?

It depends:

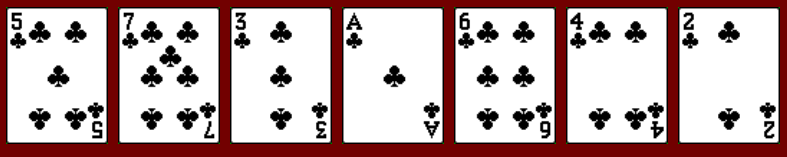
- *How many items to be sorted*
- *Choice of sorting algorithm*

What does this mean to the user?

5

How would you sort it?

Suppose you have these 7 cards, and you need to put them in ascending order:



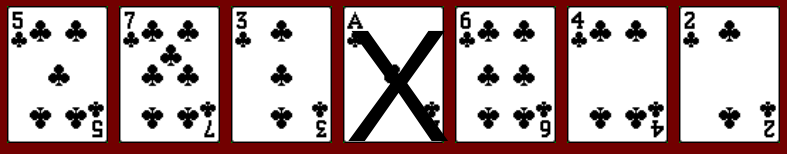
Describe your process in pseudo code.

Take a minute or two to write it down.

6

Sorting Example (1/7)

Find the card with the minimum value:

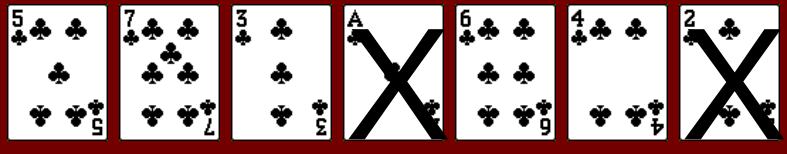


Put it in the "ordered" set:

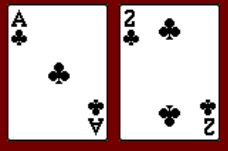
7

Sorting Example (2/7)

Find the card with the minimum value:



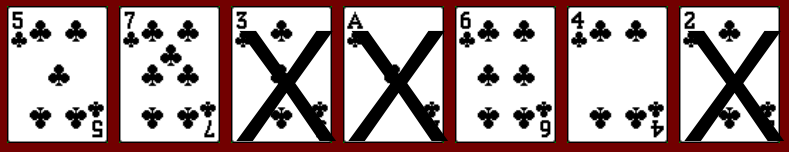
Put it in the "ordered" set:



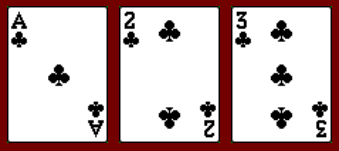
8

Sorting Example (3/7)

Find the card with the minimum value:



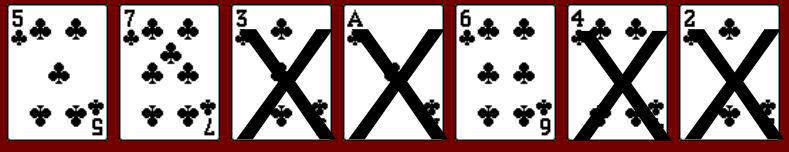
Put it in the "ordered" set:



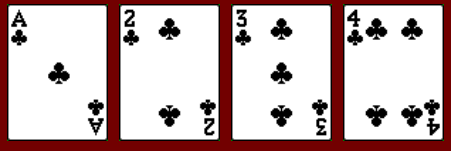
9

Sorting Example (4/7)

Find the card with the minimum value:



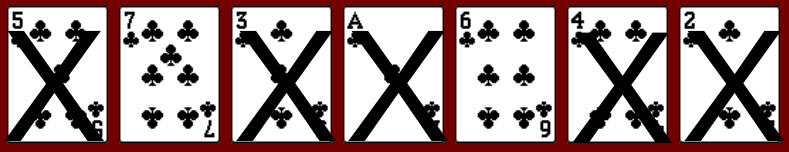
Put it in the "ordered" set:



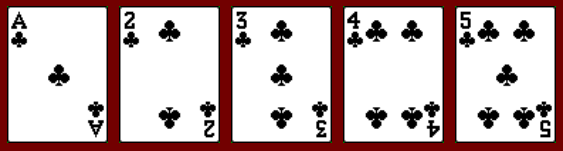
10

Sorting Example (5/7)

Find the card with the minimum value:



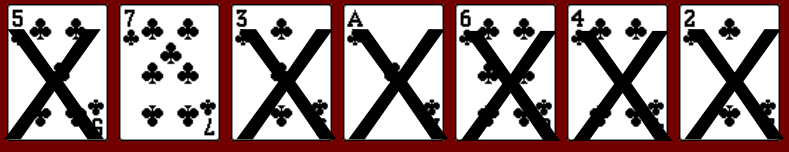
Put it in the "ordered" set:



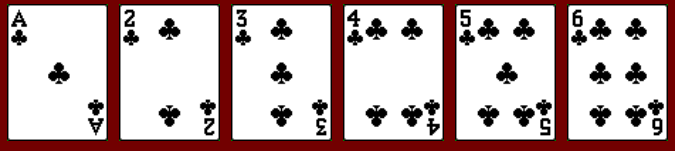
11

Sorting Example (6/7)

Find the card with the minimum value:



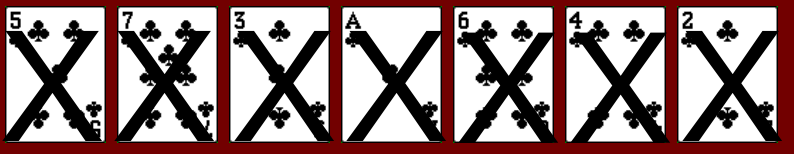
Put it in the "ordered" set:



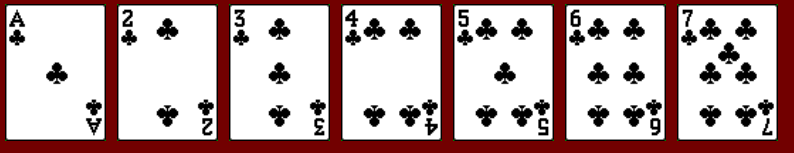
12

Sorting Example (7/7)

Find the card with the minimum value:



Put it in the "ordered" set:



13

Sorting: pseudo code

Given a set of values, put in ascending order:

```
Start a new pile for the "sorted" list
While length of "original" list > 0:
     Find minimum, copy to "sorted" list
     Remove value from the "original" list
```

This is called a selection sort.

14

Selection Sort

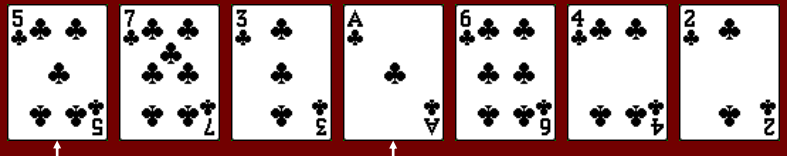
A slight adjustment to this manual approach does away with the need to duplicate space:

- As you cross a value off the original list, a free space opens up.
- Instead of writing the value found on a second list, exchange it with the value currently in the position where the crossed-off item should go.

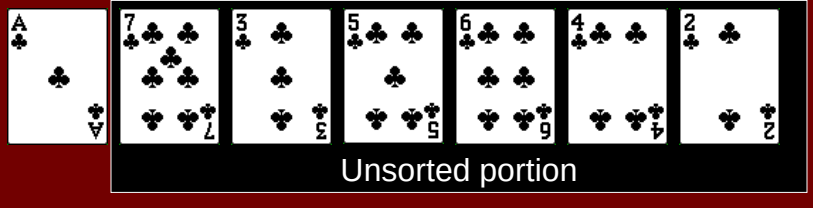
15

Selection Sort Example

Find the card with the minimum value:



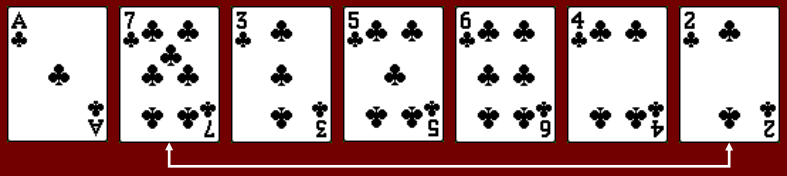
Exchange it with "first" unsorted item:



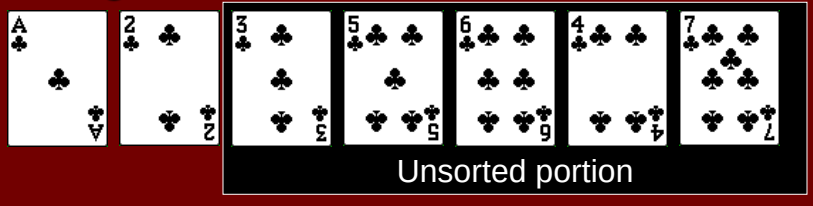
16

Selection Sort Example

Find the card with the minimum value:



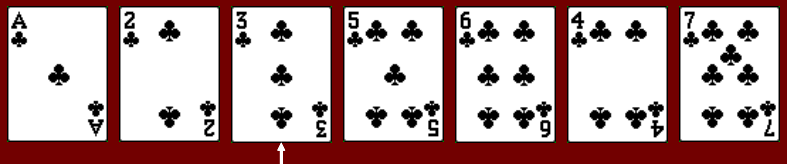
Exchange it with "first" unsorted item:



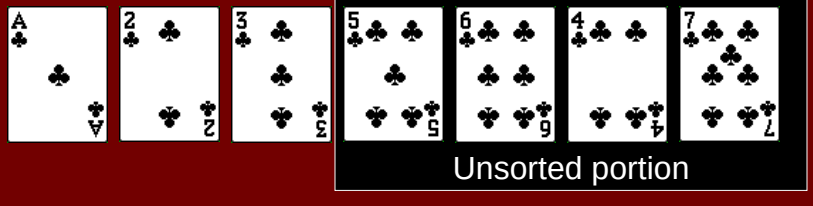
17

Selection Sort Example

Find the card with the minimum value:



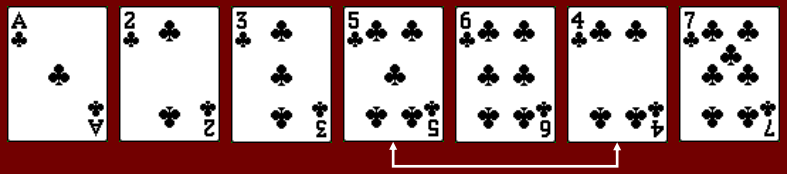
Exchange it with "first" unsorted item:



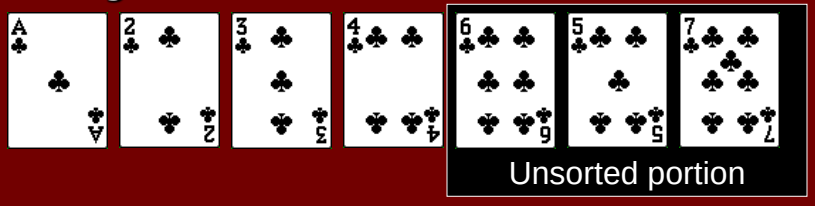
18

Selection Sort Example

Find the card with the minimum value:



Exchange it with "first" unsorted item:



19

Which item comes first?

Think about writing the code for this.
How do you find the first element in a list?

```
min = first item in list
Go through each item in the list:
    if item < min:
        min = item
```

How many comparisons does it take to find min?
Consider a list of size 10.

20

Calculating the Running Time

We measure the running time of an algorithm by the number of operations it requires.

Most of the work of sorting is making comparisons between pairs of items to see which comes first.

Thus our basic question:

How many comparisons must be done?

21

Calculating the Running Time

How can we determine the number of steps required to sort a list of n items?

- Selection Sort requires n comparisons to find the next unsorted item.*
- This process must be repeated n times, to sort all items on the list.*

Thus, we can say that it will require n passes through n items to complete the sort.

n times $n = n^2$ steps

We call Selection Sort an $O(n^2)$ algorithm.

* A mathematical simplification has been made. An explanation follows for those who care.

22

* A Mathematical Footnote

Of course, we don't really need to always compare every item in the list. Once part of the list is sorted, we can ignore that part and do comparisons against the unsorted part of the list. So for a list of size n, we really need to make:

$$n+(n-1)+(n-2)+\ldots+1$$ comparisons.

This series simplifies to:

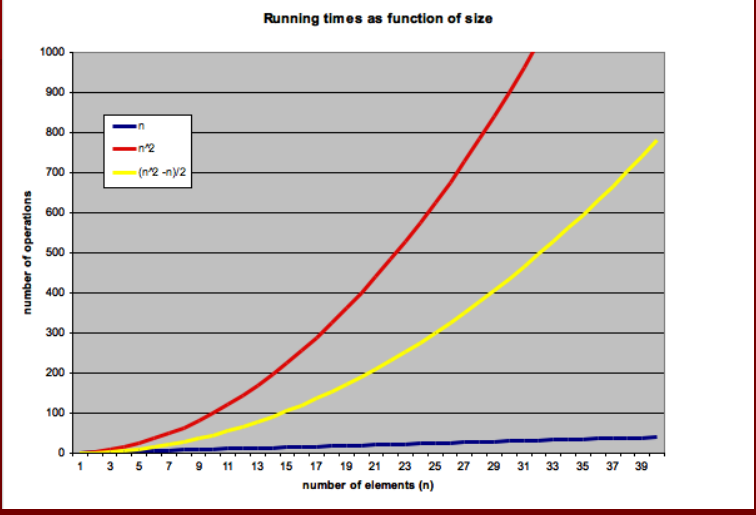
$$\frac{n(n-1)}{2}=\frac{n^2-n}{2}$$ comparisons.

This is indeed less then n^2. However, as n becomes sufficiently large, it is the n^2 part which dominates the equation's result. We make a simplification in notation and say that these algorithms are "on the order of magnitude of" n^2.

Hence the notation of $O(n^2)$ algorithm.

23

* A Mathematical Footnote



Actually, the running time is $(n^2-n)/2$, but as n becomes sufficiently large, the n^2 part of this equation dominates the outcome. Hence the notation of $O(n^2)$ algorithm.

24

Another Algorithm: Bubble Sort

Bubble Sort uses the same strategy:

- Find the next item.
- Put it into its proper place.

But uses a different scheme for finding the next item:

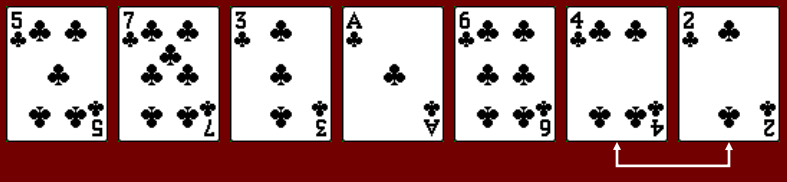
- Starting with the last list element, compare successive pairs of elements, swapping whenever the bottom element of the pair is smaller than the one above it.

The minimum "bubbles up" to the top (front) of the list.

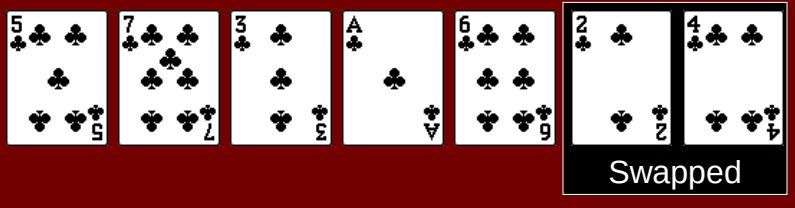
25

Bubblesort Example (1/6)

First pass: comparing last two items:



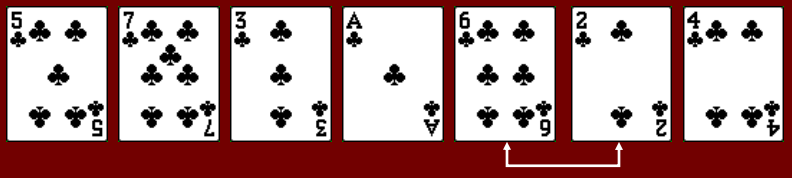
Swap if needed:



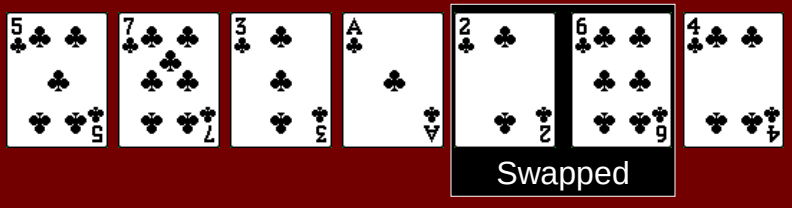
26

Bubblesort Example (2/6)

First pass: compare next pair of items:



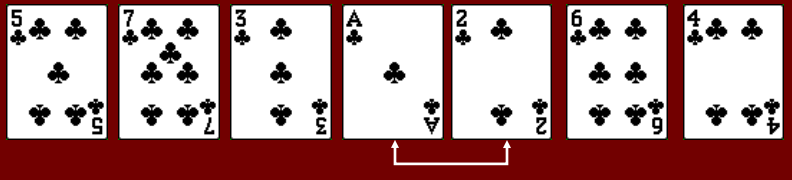
Swap if needed:



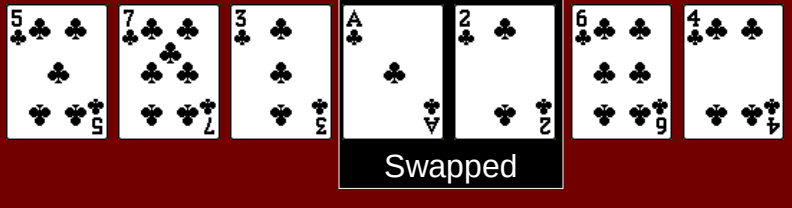
27

Bubblesort Example (3/6)

First pass: compare next pair of items:



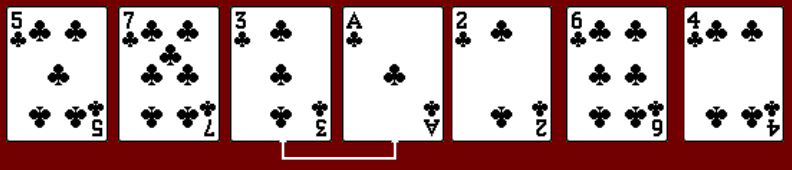
Swap if needed:



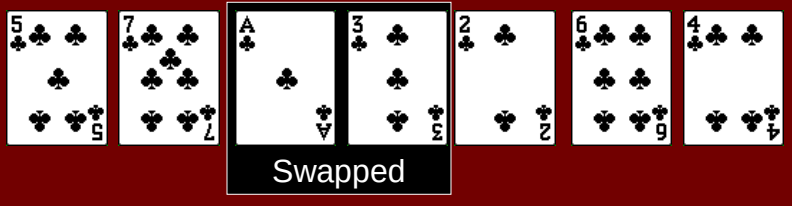
28

Bubblesort Example (4/6)

First pass: compare next pair of items:



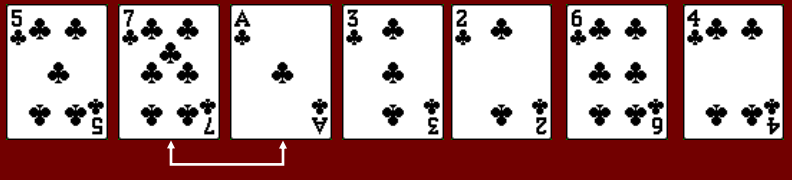
Swap if needed:



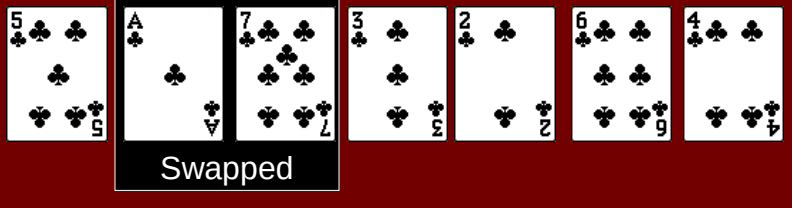
29

Bubblesort Example (5/6)

First pass: compare next pair of items:



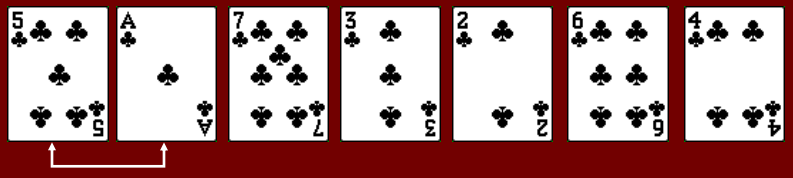
Swap if needed:



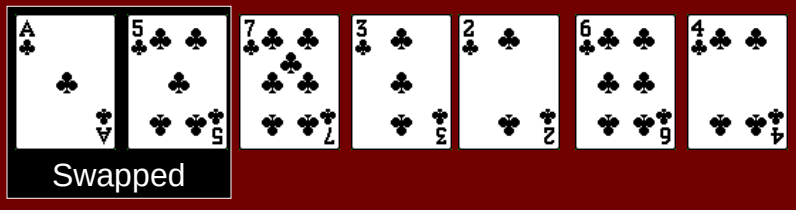
30

Bubblesort Example (6/6)

First pass: compare next pair of items:



Swap if needed:

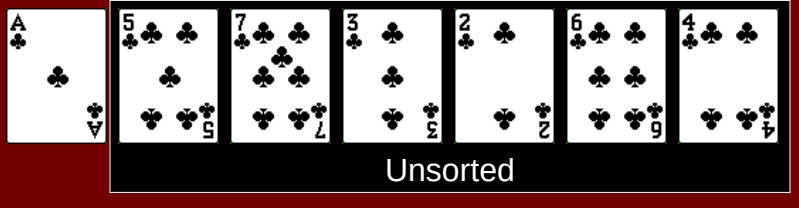


31

Bubblesort Example

After first pass:

We have 1 sorted item and 6 unsorted items:



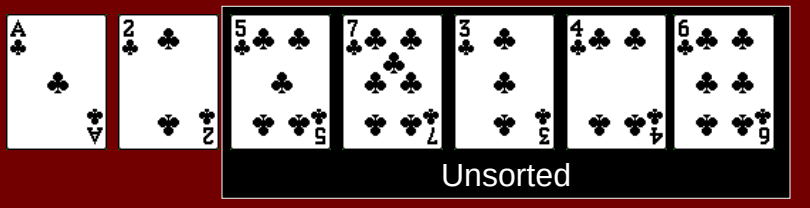
Notice: all other items are slightly "more sorted" then they were at the start.

32

Bubblesort Example

After second pass:

We have 2 sorted items and 5 unsorted items:



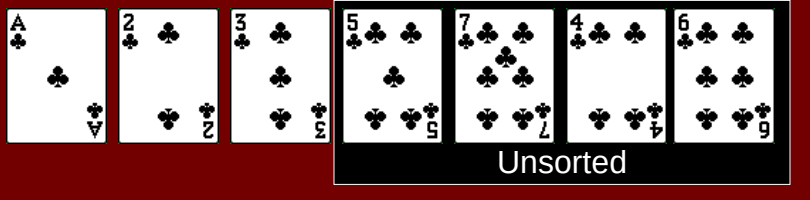
Notice: all other items are slightly "more sorted" then they were at the start.

33

Bubblesort Example

After third pass:

We have 3 sorted items and 4 unsorted items:



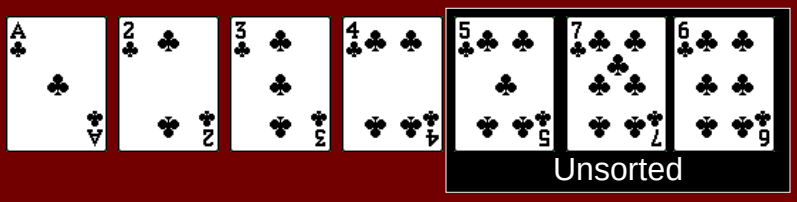
Notice: all other items are slightly "more sorted" then they were at the start.

34

Bubblesort Example

After fourth pass:

We have 4 sorted items and 3 unsorted items:



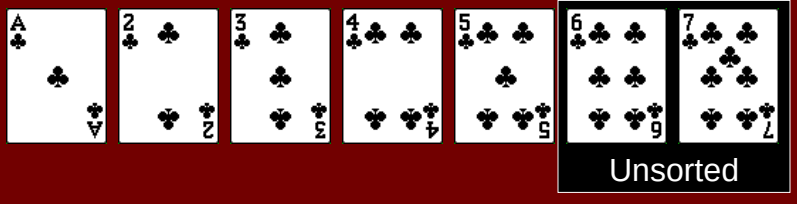
Notice: all other items are slightly "more sorted" then they were at the start.

35

Bubblesort Example

After fifth pass:

We have 5 sorted items and 2 unsorted items:



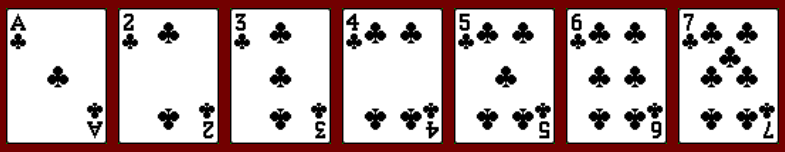
No, the last two items are not sorted yet!

Why not?

36

Bubblesort Example

After sixth pass, all items have been sorted:



37

Calculating the Running Time

How do we calculate the running time for Bubble Sort? Determine the number of comparisons.

For a list of size n:

- Bubble Sort will go through the list n times
- Each time compare n adjacent pairs of numbers.*

n times n = n^2 steps

Bubble Sort is also an $O(n^2)$ algorithm.

* A mathematical simplification has been made. See previous footnote.

38

A Different Approach to Sorting

Selection Sort and Bubble Sort have the same basic strategy:

- Find one item, and put it in place
- Repeat

How else might we approach this problem?

Hint: it takes fewer comparisons to sort a smaller list.

How many comparisons does it take to sort 2 items?

39

Quicksort: Divide and Conquer

Quicksort uses a divide-and-conquer strategy. It is simpler and shorter to solve a smaller problem.

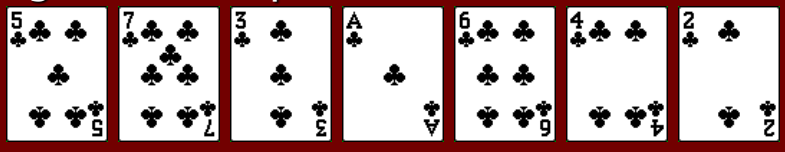
Basic strategy:

- Split the list based on a split value; put each item on a sub list (great then split or less than split).
- Sort each sub list using the same approach
- Continue splitting and sorting sub lists until we get
  - a list of length 1 (which is by definition sorted
- Combine all of the sorted sub lists together to create the complete ordered list.

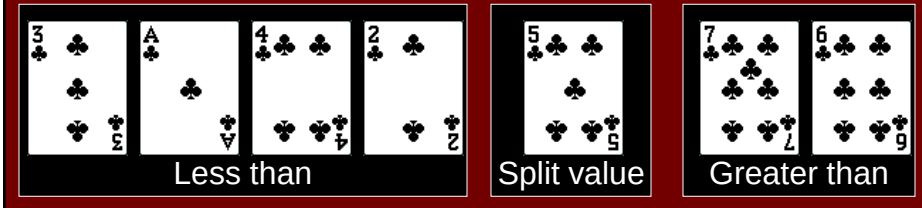
40

Quicksort Example

Begin with complete set of cards:



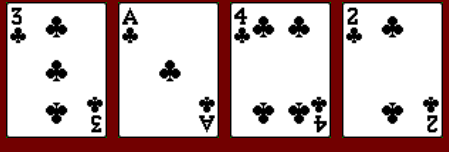
Split into two groups based on "less than 5" or "greater than 5":



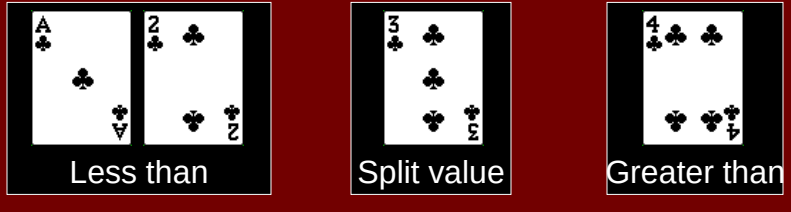
41

Quicksort Example

Repeat quicksort on each sublist:



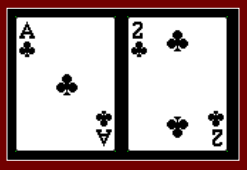
Split into two groups based on "less than 3" or "greater than 3":

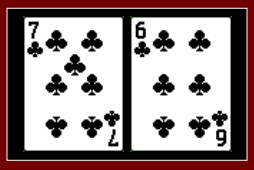


42

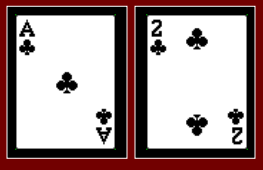
Quicksort Example

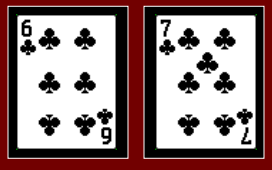
Repeat quicksort on each sub list:





Until we have sub lists of length 1:

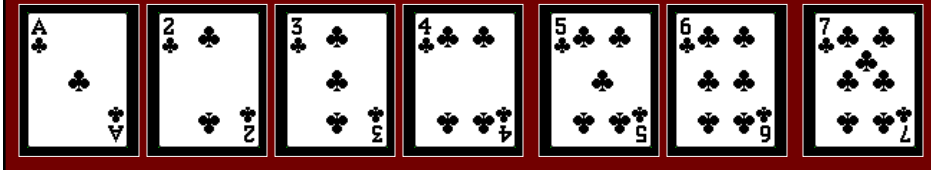




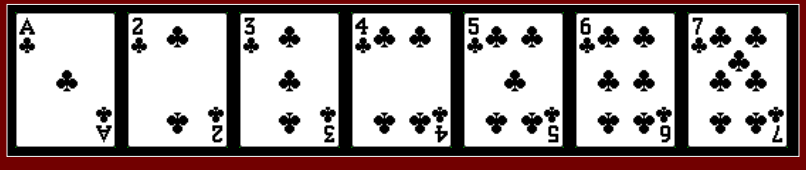
43

Quicksort Example

At the limit, we have this set of sublists:



Finally, we combine into the complete, sorted list.



44

Quicksort Algorithm

```
Quicksort(list):
  if length of list > 1 then
     select splitVal

     for each item in list:
        if item < splitVal:
           add item to lesser
        if item > splitVal:
           add item to greater

     Quicksort(lesser)
     Quicksort(greater)

     list = lesser + splitVal + greater
```

45

Calculating the Running Time

How do we calculate the running time for Quicksort?

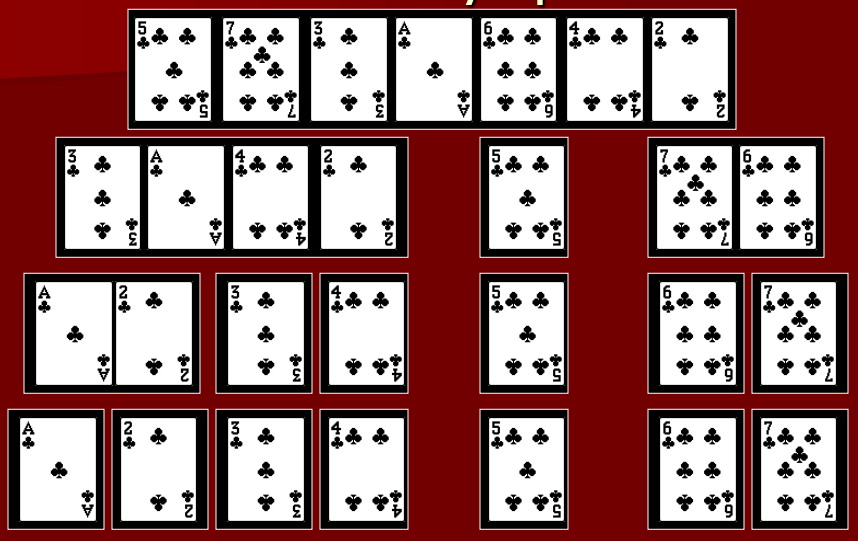
- Determine the number of comparisons.
- Determine the number of time we split the list.

Each time we want to split the list, we need to compare each item to the split value, and assign it to the correct sub-list.

- For a list of size n, we have n comparisons per split.

46

How many splits?



Running Time: Quicksort

How many times do we split a list of size n?

We keep splitting (in half) until we reach 1. How many splits is that?

For n = 2, splits = 1
For n = 4, splits = 2
For n = 8, splits = 3
For n = 16, splits = 4
For n = 32, splits = 5
For n = 64, splits = 6

What is the pattern here?

Running Time: Quicksort

Pattern: Each time we double the length of the list (n), we increase the number of splits by 1.

This is the opposite of the exponential relationship.

Recall that:

$2^2 = 2*2 = 4$

$2^3 = 2*2*2 = 8$

$2^4 = 2*2*2*2 = 16$

49

Recall: Logarithms

The base-2 logarithm describes how many times we need to divide a value in half to obtain 1:

$\log_2(2) = 1$
$\log_2(4) = 2$
$\log_2(8) = 3$
$\log_2(16) = 4$
$\log_2(32) = 5$

$\log_2(n) = x$

where x is the power to which we would raise 2 to obtain n:

$2^x = n$

50

Running Time: Quicksort

Recall that for a list of size n:

- We have n comparisons per split, and
- We have $\log_2(n)$ splits.

Combining these, we can write

n times $\log_2(n) = n^* \log_2(n)$ steps

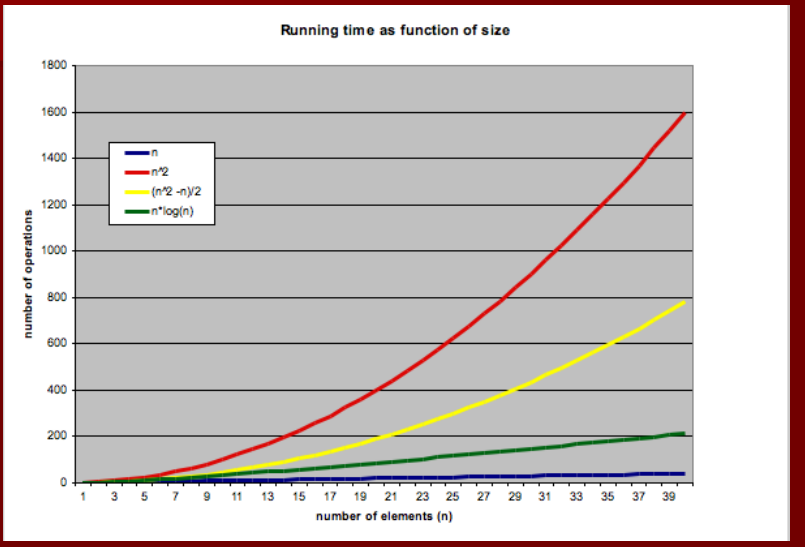
Quicksort is an $O(n^* \log_2(n))$ algorithm*.

$n^* \log_2(n)$ is always less than n^2.

*Average case running time based on optimal choice of split. What's the worst case?

51

Running Time Comparison



52

Sorting Algorithm Demo

A really cool graphical demo of different sorting algorithms running side-by-side:
http://www.cs.clarku.edu/~jmagee/cs121/examples/sorting/sorts.html
(with thanks to Penny Ellard for the original page)

Also, check this out:
http://www.sorting-algorithms.com/

53

Searching: Overview/Questions

- What is searching?
- Why does searching matter?
- How is searching accomplished?
- Why are there different searching algorithms?
- On what basis should we compare algorithms?

54

Searching

Searching

Attempting to find an item in a collection. Note the possibility that an item might not be found.

Search key

The field (or fields) on which the search is based.

55

Example: Search Key

Consider this list:

NO.	NAME	POS	HT	WT	BORN	YR	COLLEGE	HOMETOWN
88	Sam Aiken	WR	6-2	215	12/14/1980	6	North Carolina	Kenansville, N.C.
52	Eric Alexander	LB	6-2	240	2/8/1982	4	Louisiana State	Port Arthur, Texas
65	Wesley Britt	T	6-8	320	11/21/1981	3	Alabama	Cullman, Ala.
54	Tedy Bruschi	LB	6-1	247	6/9/1973	13	Arizona	Roseville, Calif.
16	Matt Cassel	QB	6-4	230	5/17/1982	4	Southern California	Northridge, Calif.
98	Shawn Crable	LB	6-5	243	12/26/1984	R	Michigan	Massillon, Ohio
44	Heath Evans	FB	6-0	250	12/30/1978	8	Auburn	West Palm Beach, Fla.
33	Kevin Faulk	RB	5-8	202	6/5/1976	10	Louisiana State	Carencro, La.
10	Jabar Gaffney	WR	6-1	200	12/1/1980	7	Florida	Jacksonville, Fla.
3	Stephen Gostkowski	K	6-1	210	1/28/1984	3	Memphis	Madison, Miss.
97	Jarvis Green	DL	6-3	285	1/12/1979	7	Louisiana State	Donaldsonville, La.
42	BenJarvus Green-Ellis	RB	5-11	215	7/2/1985	R	Mississippi	New Orleans, La.
7	Matt Gutierrez	QB	6-4	230	6/9/1984	2	Idaho State	Concord, Calif.
59	Gary Guyton	LB	6-3	242	11/14/1985	1	Georgia Tech	Hinesville, Ga.

http://www.patriots.com/team/

56

Example: Search Key

Now consider this list:

NO.	NAME	POS	HT	WT	BORN	YR	COLLEGE	HOMETOWN
3	Stephen Gostkowski	K	6-1	210	1/28/1984	3	Memphis	Madison, Miss.
5	Kevin O'Connell	QB	6-5	225	5/25/1985	R	San Diego State	Carlsbad, CA
6	Chris Hanson	P	6-2	202	10/25/1976	10	Marshall	Sharpsburg, Ga.
7	Matt Gutierrez	QB	6-4	230	6/9/1984	2	Idaho State	Concord, Calif.
10	Jabar Gaffney	WR	6-1	200	12/1/1980	7	Florida	Jacksonville, Fla.
15	Kelley Washington	WR	6-3	215	8/21/1979	6	Tennessee	Stephens City, Va.
16	Matt Cassel	QB	6-4	230	5/17/1982	4	Southern California	Northridge, Calif.
18	Matthew Slater	WR	6-0	198	9/9/1985	R	UCLA	Anaheim, Calif.
21	Deltha O'Neal	CB	5-11	194	1/30/1977	9	California	Milpitas, Calif.
22	Terrence Wheatley	CB	5-9	183	5/5/1985	R	Colorado	Plano, Texas
24	Jonathan Wilhite	CB	5-11	185	2/23/1984	R	Auburn	Monroe, La.
27	Ellis Hobbs III	CB	5-9	195	5/16/1983	4	Iowa State	DeSoto, Texas
28	Antwain Spann	CB	6-0	195	2/22/1983	3	Louisiana-Lafayette	Oceanside, Calif.
29	Lewis Sanders	CB	6-1	210	6/22/1978	9	Maryland	Staten Island, N.Y.
31	Brandon Meriweather	S	5-11	200	1/14/1984	2	Miami (Fla.)	Apopka, Fla.

57

Example: Search Key

NFL uniform numbers follow a specific scheme to make search by number easy for TV announcers…

- Numbers 1 to 19 are worn by quarterbacks, kickers, and punters.
- Numbers 20 to 49 are worn by running backs, tight ends, cornerbacks and safeties.
- Numbers 50 to 59 are worn by linebackers and offensive linemen.
- Numbers 60 to 79 are worn by members of both the offensive line and defensive line.
- Numbers 80 to 89 are worn by wide receivers and tight ends.
- Numbers 90 to 99 are worn by linebackers and defensive linemen.

50	Mike Vrabel	LB	6-4	261	8/14/1975	12	Ohio State	Akron, Ohio
51	Jerod Mayo	LB	6-1	242	2/23/1986	R	Tennessee	Hampton, Va.
52	Eric Alexander	LB	6-2	240	2/8/1982	4	Louisiana State	Port Arthur, Texas
53	Larry Izzo	LB	5-10	228	9/26/1974	13	Rice	Houston, Texas
54	Tedy Bruschi	LB	6-1	247	6/9/1973	13	Arizona	Roseville, Calif.
58	Pierre Woods	LB	6-5	250	1/6/1982	3	Michigan	Cleveland, Ohio
59	Gary Guyton	LB	6-3	242	11/14/1985	1	Georgia Tech	Hinesville, Ga.

58

Why Does Searching Matter?

Given a large enough set of data, it could take a long time to find something!

Example:

Consider a collection with 10,000,000 records (e.g. a phone book).

How would you find what you're looking for?

Searching Algorithms

Algorithms that traverse the collection in an attempt to find the desired item.

59

Why Does Searching Matter?

Searching is an operation which can take a lot of computing cycles.

How many cycles?

It depends:

- *How many items in the collection*
- *Choice of searching algorithm*

What does this mean to the user?

60

A Naïve Searching Algorithm

While there are more items in the list:
- Look at the next item.
- Is this what you were looking for?
- If yes, all done.
- If no, take the next item.

If you got here and didn't find the item, then it must not be in the list.

We call this Linear Search.

61

Linear Search

Characteristics of the Linear Search:
- Simple to implement.
- It doesn't matter if the collection is sorted.

How many items do you need to look at to find what you're looking for?
- *What if it is in the first position?*
- *What if it is in the last position?*
- *What if it is not found?*

62

Calculating the Running Time

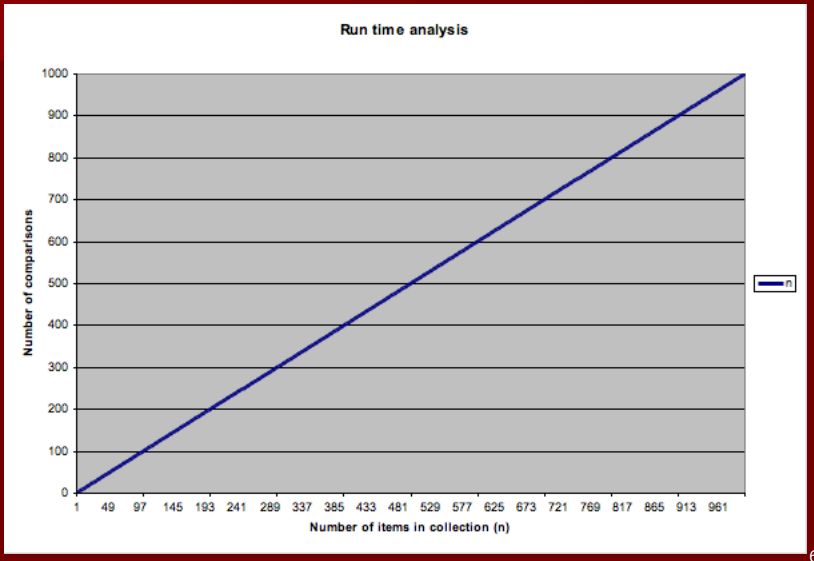
How many steps are required to do a linear search through a list of n items?

- If we check each one, it will take looking through n items to complete the search.
- On average, we might expect to look at n/2 items on a given search.
- If the item is not found, we will not discover that until the end of the list – after traversing all n items.

We call Linear Search an O(n) algorithm.

63

Running Time Analysis



64

A Different Approach to Searching

How else might we approach this problem?

Hint: can we divide and conquer?

65

Binary Search

Binary Search uses a divide-and-conquer strategy.

It requires that the collection be sorted.

Basic strategy:

- Divide the list in half based on some mid point, to get two shorter lists (before and after mid point).
- Compare the search key to the mid point. Does it come before or after the midpoint? Search the corresponding sub list.
- Continue splitting and searching sub lists until we get a list of length 1:
  - either this is our item, or the item is not in the collection.

66

Calculating the Running Time

How do we calculate the running time for Binary Search?

- Determine the number of comparisons.
- Determine the number of times we split the collection.

Each time we want to split the list, we need to make one comparison (search item to mid point), proceed to search a sub-list.

67

Running Time: Binary Search

How many times do we split a list of size n?

We keep splitting (in half) until we reach 1. How many splits is that?

For n = 2, splits = 1
For n = 4, splits = 2
For n = 8, splits = 3
For n = 16, splits = 4
For n = 32, splits = 5
For n = 64, splits = 6

Recall this is the $\log_2(n)$ pattern.

68

Running Time: Binary Search

Pattern: Each time we split the list in half, we have a list of length (n/2).

The base-2 logarithm describes how many times we need to divide a value in half to obtain 1:

$\log_2(2) = 1$
$\log_2(4) = 2$
$\log_2(8) = 3$
$\log_2(16) = 4$
$\log_2(32) = 5$

69

Running Time: Binary Search

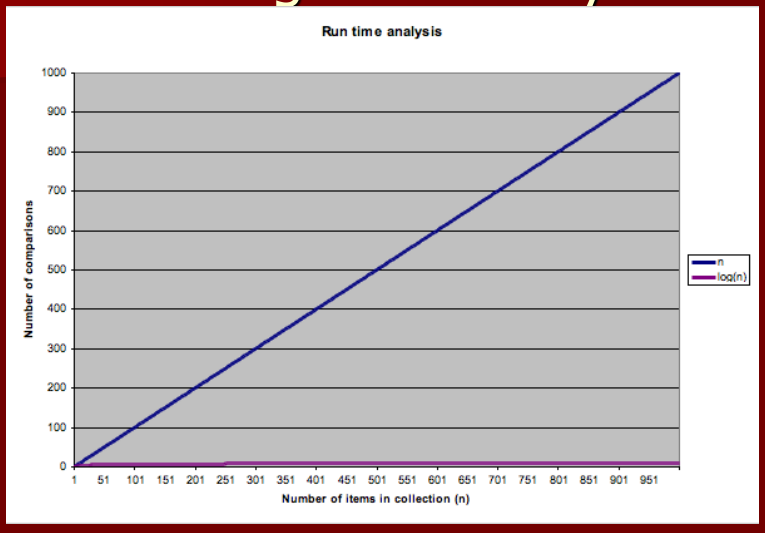
For a list of size n, we have $\log_2(n)$ splits.

Thus, Binary Search is an $O(\log_2(n))$ algorithm.

$\log_2(n)$ is always less than n.

70

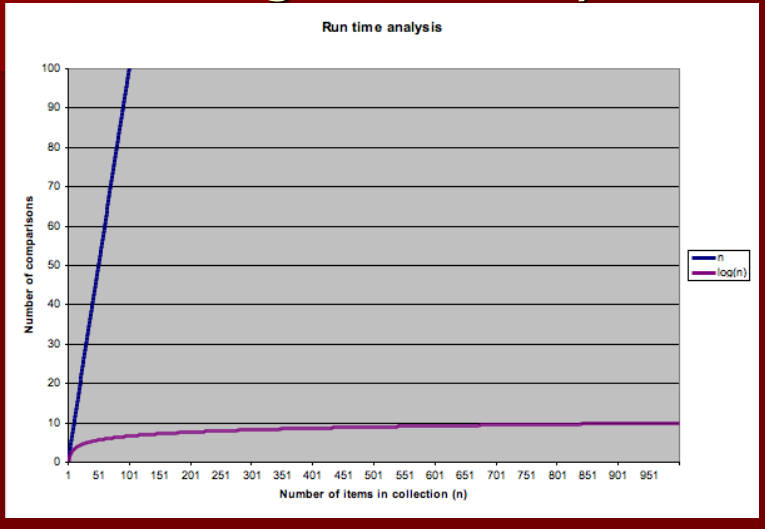
Running Time Analysis



We can barely see the line for $\log_2(n)$!

71

Running Time Analysis



Now we see $\log_2(n)$! Note: change of scale.

72