Professor Hamlin
Day 20

1. Admin

   Hw 8 due Nov 26^{th} } different due to holiday
   Hw 9 due Dec 3^{rd}
   Exam 3 Dec 3^{rd}
   - 30 min but get 50 min
   - covers classes 18, 19, 20
   - class 21 will be on hw 9, not on exam

2. Review
3. Search algorithm → linear, binary
4. Sort algorithms → insertion, merge
5. Algorithm runtime

Review

Big O
"$f(n) \le g(n)$"

Big-Omega
"$g(n) \le f(n)$"

Big Theta
"$f(n) = g(n)$"

Exercise: Sort functions below from fastest to slowest

constant: 1 | log: $\log N$ | linear: N | log-linear: $N \log N$ | polynomial: N^2 | N^3 | $\ldots$ | exponential: 2^N | 3^N | $\ldots$ | factorial: $N!$ | N^N

n	$\log n$	$2^x = n$
$\sqrt{n}$	$\log^2 n$	$n^{1/2}$
$\log n$	$\sqrt{n}$	
$\log^2 n = (\log n)^2$	n	
3^n	3^n	
$n!$	$n!$	

Quick review: Fun w/ logarithms

$$2^3 = 8 \iff \log_2 8 = 3$$

$\log_B x$ is the value raise B to to obtain x

Some laws:

1. $\log_B m \cdot n = \log_B m + \log_B n$
2. $\log_B m/n = \log_B m - \log_B n$
3. $\log_B n^p = p \cdot \log_B n$

Exercise: Solve for x

1. $\log_{10} 1000 = x$

   $10^x = 1000$

   $x = 3$

2. $\log_2 16 = x$

   $2^x = 16$

   $x = 4$

3. $\log_2 x = 10$

   $2^{10} = x$

   $x = 1024$

4. $\log_2 16 + \log_2 32 = x$

   $2^? = 16 \qquad 2^? = 32$

   $4 \quad + \quad 5$

   $x = 9$

5. $\log_2 (16 \cdot 32) = x$

   By our log rules!

   $x = 9$

Foundational Convention in CS

... indexing a list starts at 0

L =

14	coz	a	4	-5	6	q
L[0]	L[1]	L[2]	...			

L[0] = the 1st item in list

L[4] = -5

L[i] = the i-th item in list.

Now that everyone will never talk about indexing starting at 1 again, let's talk about

"Searching" and "Sorting"

What humans vs Computers see:

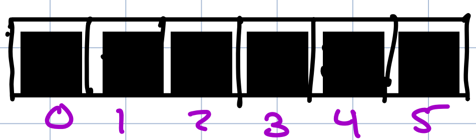
Human:

3	100	5	-7	19	42
0	1	2	3	4	5

"What is 19's index?"
4

Humans can look at list wholistically and spot 19 - we are good at seeing patterns!

Computer:

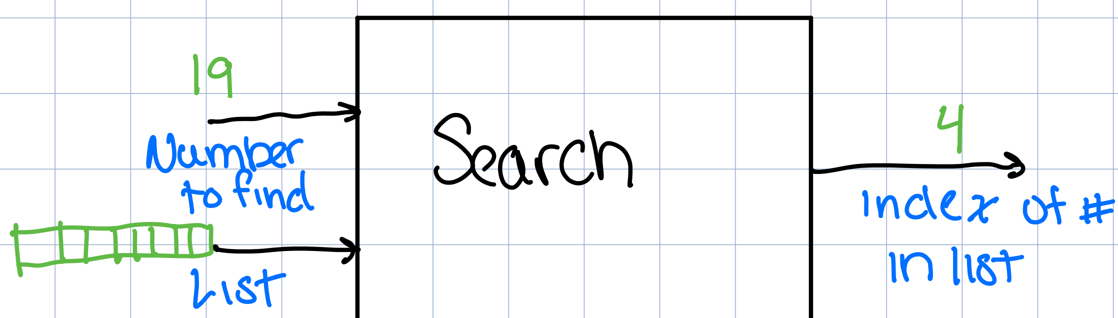


"What is 19's index?"

Computers can only "see" one thing at atime.

Searching

Informally: search is any algorithm that does the following.



How the box works may change but not it's inputs and outputs

Q: what if there is more than one copy of an item?

Search returns the first one's index

1st Searching Algorithm: Linear search

Intuition: Start at 0th index,
check if current item is equal
if yes stop
else move right and repeat

1st

14	102	-4	19	6

↑ current index

is $L[0] \stackrel{?}{=} 19$, no so move right one

2nd

14	102	-4	19	6

↑ current index

is $L[1] \stackrel{?}{=} 19$, no so move right one

3rd

14	102	-4	19	6

↑ current index

is $L[2] \stackrel{?}{=} 19$, no so move right one

4th

14	102	-4	19	6

↑ current index

$L[3] \stackrel{\checkmark}{=} 19$, so we return 3 as the result of search

Is this a good algorithm?

(what do we want from our algorithms?)

Fast, Correct, understandable

1. Correctness: will the algorithm always return the correct answer

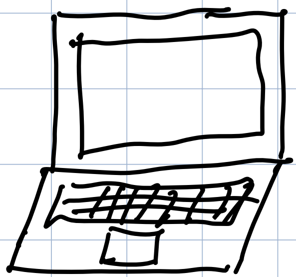
2. Runtime: completes tasks in as few "operations" as possible

3. Simplicity: Can we humans understand it and code it

4. Memory overhead: how much extra "stuff" the algorithm needs to remember beyond its input

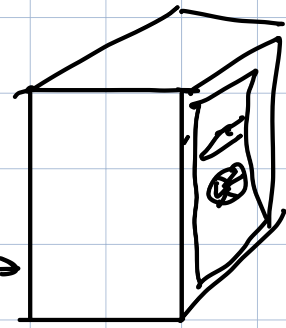
In this class we are mostly focus on runtime (the algorithm does need to be correct)

How do we count "runtime"



← old laptop

gaming pc →



If we just measure the time of an algorithm hard to compare because it depends on the computer.

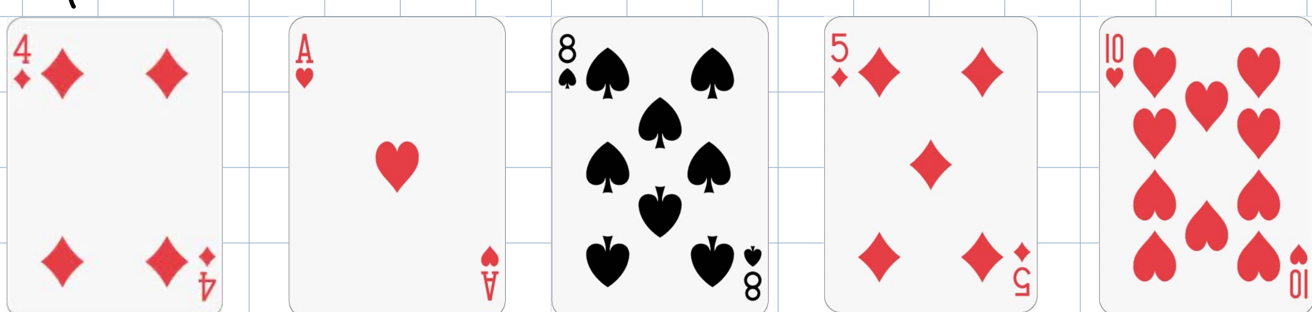
More useful to count the number of operations needed...

... in searching and sorting we count comparisons e.g

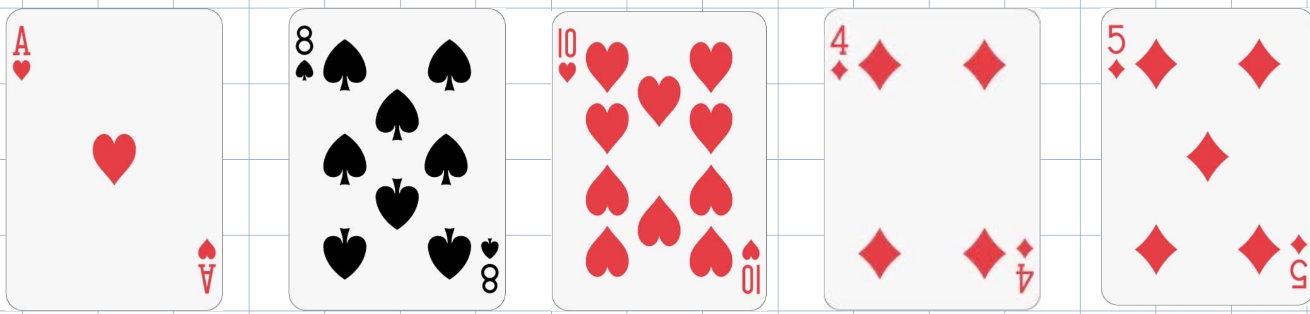
$x < y, \ x > y, \ x = y$

(... in others we may count the number of multiplications or AND gates)

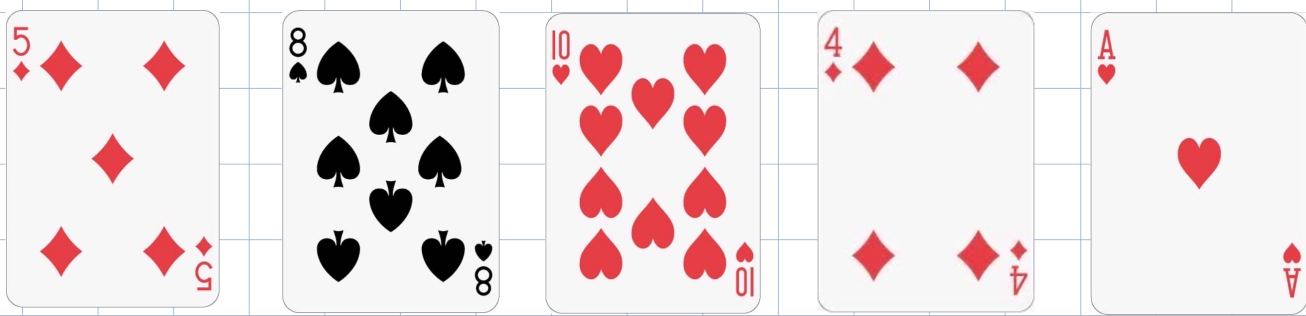
Example: find five using linear



4 comparisons



5 comparisons, worst case



1 comparisons, best case

Different inputs require different number of comparisons

Best case vs Worst case Runtime

Many algorithms have "best case" for certain input (e.g item is at start of list) or "worst case" (e.g item at end of list)

We will always think about worst case when analyzing algorithms

Fun Fact | Other algorithms are the same in best & worst case

$$\boxed{\text{Linear Search has a runtime}^{*}\ T(n) = n}$$

$O(n)$

* worst case, counting comparisons

Can we have a better runtime? Yes!

Binary Search:

Need some volunteers.

> Take away: if list is sorted, can use that information to make search faster

Intuition: *look in middle of remaining list
if equal stop
if middle is smaller than item
discard all elements to the left of middle and start from *
if middle is bigger than item
discard all elements to the right of middle and start from *

Example: find 19

0	1	2	3	4	5	6	7
1	4	7	11	14	19	21	23
			↑ current index				

19 > L[3] so discard left of L[4]

0	1	2	3	4	5	6	7
				14	19	21	23
						↑ current index	

19 < L[6] so discard right of L[4]

0	1	2	3	4	5	6	7
				14	19		
					↑ current index		

19 = L[5], return index 5

Exercise

1) Build example (list of length 7 & item) where binary search works quickest

0 1 2 $\boxed{3}$ 4 5 6

looking for 3
1 comparison

2) Build example where binary search works slowest

$\boxed{0}$ $\boxed{1}$ 2 $\boxed{3}$ 4 5 6
3 2 1

looking for $\boxed{2/4 \quad 0/6}$
3 comparison

3) For list of size 8, 16, 32, 64, etc...
what is the most number of comparisons required? 3, 4, 5, 6

Worst-Case Runtime: Binary Search

- each comparison cuts list in half
- stop when we reach one element

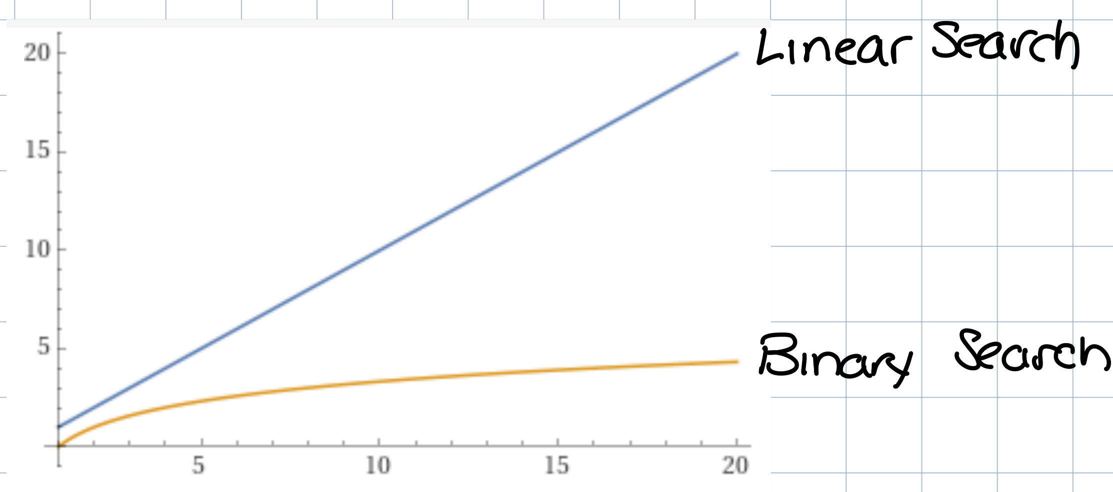
List size	128 →	64 →	32 →	16 →	8 →	4 →	2 →	1
powers of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
# of comparisons	8	7	6	5	4	3	2	1

Number of comparisons is $\log_2(\text{list size})$!

Runtimes so far...

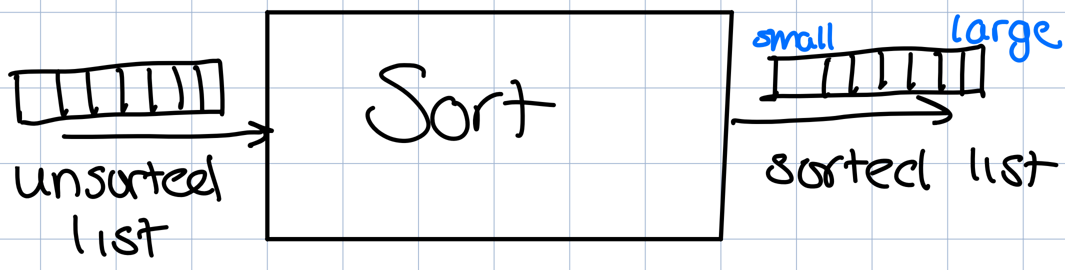
$\boxed{\text{Linear Search has a runtime } T(n) = n}$

$\boxed{\text{Binary Search has a runtime } T(n) = \log_2 n}$



Sorting

How can we sort our lists?



Cover two algorithms for sorting.

1. Insertion sort (today)
2. Merge sort (next class)

Need volunters

take away: an element by itself is sorted, if we add elements one by one, can maintain the sort

Intuition: add element to sorted section from unsorted, keep swapping until it is in its spot

Example (for reference)

start:

9	5	7	3	1	4	11

sorted

9	9	5	7	3	1	4	11	9 > 5 so swap

current item to be sorted

5	5	9	7	3	1	4	11	5 is as far left so done

7	5	9	7	3	1	4	11	9 > 7 so swap
	5	7	9	3	1	4	11	5 < 7 so stop

3	5	7	9	3	1	4	11	3 < 9 so swap
	5	7	3	9	1	4	11	3 < 7 so swap
	5	3	7	9	1	4	11	3 < 5 so swap
	3	5	7	9	1	4	11	3 at end, stop

1	3	5	7	9	1	4	11	1 < 9, swap
	3	5	7	1	9	4	11	1 < 7, swap
	3	5	1	7	9	4	11	1 < 5, swap
	3	1	5	7	9	4	11	1 < 3, swap
	1	3	5	7	9	4	11	stop

4

1	3	5	7	9	4	11	4 < 9, swap
1	3	5	7	4	9	11	4 < 7, swap
1	3	5	4	7	9	11	4 < 5, swap
1	3	4	5	7	9	11	3 < 4, stop

11

1	3	4	5	7	9	11	9 < 11, stop

How many comparisons? 18

Exercise Build two lists of length 4 that require the most / least comparisons.
(use # 2, 3, 4, 5)

2 3 4 5 — 3 comparisons

5 4 3 2 — 6 comparisons

Worst case:

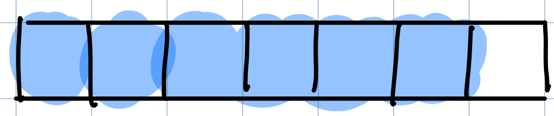
in the worst case, each element must be compared to all sorted elements

Step 1 — 1 comparison

Step 2 — 2 comparisons

⋮

Step $N-1$



$N-1$ comparisons

In total: $1 + 2 + 3 + \ldots\ldots + N-2 + N-1$

$$= \sum_{K=1}^{N-1} K$$

Arithmetic Sum

$$= (1 + N - 1)\left(\frac{N-1}{2}\right)$$

Partial Sum

$$= \frac{N^2}{2} - \frac{N}{2}$$

$$= O(N^2)$$

Big-O

Runtime Insertion Sort $O(n^2)$

$\boxed{\text{Next Time: Merge Sort ....}}$

Phase	Processed				◇		Unprocessed		
0	◇	34	16	12	11	54	10	65	37
1	34	◇	16	12	11	54	10	65	37
2	16	34	◇	12	11	54	10	65	37
3	12	16	34	◇	11	54	10	65	37
4	11	12	16	34	◇	54	10	65	37
5	11	12	16	34	54	◇	10	65	37
6	10	11	12	16	34	54	◇	65	37
7	10	11	12	16	34	54	65	◇	37
8	10	11	12	16	34	37	54	65	◇

EVERYTHING LEFT OF SYMBOL IS SORTED