A Guessing Game

Let's play a little game to give you an idea of how different algorithms for the same problem can have wildly different efficiencies. The computer is going to randomly select an integer from 1 to 16. You have to guess the number by making guesses until you find the number that the computer chose. Let's begin.

Reset Game

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Guess my number

Maybe you guessed 1, then 2, then 3, then 4, and so on, until you guessed the right number. We call this approach **linear search**, because you guess all the numbers as if they were lined up in a row. It would work. But what is the highest number of guesses you could need? If the computer selects 16, you would need 16 guesses. Then again, you could be really lucky, which would be when the computer selects 1 and you get the number on your first guess. How about on average? If the computer is equally likely to select any number from 1 to 16, then on average you'll need 8 guesses.

But you could do something more efficient than just guessing 1, 2, 3, 4, …, right? Since the computer tells you whether a guess is too low, too high, or correct, you can start off by guessing 15. If the number that the computer selected is less than 15, then because you know that 15 is too high, you can eliminate all the numbers from 15 to 30 from further consideration. If the number selected by the computer is greater than 15, then you can eliminate 1 through 15. Either way, you can eliminate about half the numbers. On your next guess, eliminate half of the remaining numbers. Keep going, always eliminating half of the remaining numbers. We call this halving approach eliminating half of the remaining numbers. We call this halving approach **binary search**, and no matter which number from 1 to 30 the computer has selected, you should be able to find the number in at most 5 guesses with this technique.

Here, try it for a number from 1 to 300. You should need no more than 9 guesses.

Reset Game

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300

Guess my number

How many guesses did it take you to find the number this time? Why should you never need more than 9 guesses? (Can you think of a mathematical explanation)?

We'll return to binary search, and we'll see how you can use it to efficiently search for an item in an array. But first, let's look at an algorithm for a trickier problem.