

Unit 6: Where did I put it?

Searching and Sorting Algorithm

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 4: The student can use programming as a creative tool.
- 15: The student can develop an algorithm.
- 16: The student can express an algorithm in a language.
- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Readings/Lectures

- Blown to Bits: Chapter 4 (<http://www.bitsbook.com/wp-content/uploads/2008/12/chapter4.pdf>)
- Reading 6.01: All Algorithms are Not Equal (/bjc-course/curriculum/06-searching-sorting/readings/01-all-algorithms-are-not-equal)
- Reading 6.02: Algorithm Analysis (/bjc-course/curriculum/06-searching-sorting/readings/02-algorithm-analysis)
- Lecture Slides 6.03: Searching and Sorting with Alonzo (/bjc-course/curriculum/06-searching-sorting/readings/03-algorithms-slides.pdf)

External Resources

- How Algorithms Shape our World (http://www.ted.com/talks/kevin_slavin_how_algorithms_shape_our_world.html) - TED Talk

Labs/Exercises

- Activity 6.01: Searching Activity (/bjc-course/curriculum/06-searching-sorting/labs/01-searching-activity)
- Activity 6.02: Searching (non Video-) Game (/bjc-course/curriculum/06-searching-sorting/labs/02-searching-game)
- Lab 6.03: Update the Guessing Game (/bjc-course/curriculum/curriculum/06-searching-sorting/labs/03-update-guessing-game)

A FUTURIST PRECEDENT

In 1937, H. G. Wells anticipated Vannevar Bush's 1945 vision of a "memex." Wells wrote even more clearly about the possibility of indexing everything, and what that would mean for civilization:

There is no practical obstacle whatever now to the creation of an efficient index to all human knowledge, ideas and achievements, to the creation, that is, of a complete planetary memory for all mankind. And not simply an index; the direct reproduction of the thing itself can be summoned to any properly prepared spot. ... This in itself is a fact of tremendous significance. It foreshadows a real intellectual unification of our race. The whole human memory can be, and probably in a short time will be, made accessible to every individual. ... This is no remote dream, no fantasy.

Capabilities that were inconceivable then are commonplace now. Digital computers, vast storage, and high-speed networks make information search and retrieval necessary. They also make it possible. The Web is a realization of Bush's memex, and search is key to making it useful.

It Matters How It Works

How can Google or Yahoo! possibly take a question it may never have been asked before and, in a split second, deliver results from machines around the world? The search engine doesn't "search" the entire World Wide Web in response to your question. That couldn't possibly work quickly enough—it would take more than a tenth of a second just for bits to move around the earth at the speed of light. Instead, the search engine has *already* built up an index of web sites. The search engine does the best it can to find an answer to your query using its index, and then sends its answer right back to you.

To avoid suggesting that there is anything unique about Google or Yahoo!, let's name our generic search engine Jen. Jen integrates several different processes to create the illusion that you simply ask her a question and she gives back good answers. The first three steps have nothing to do with your particular query. They are going on repeatedly and all the time, whether anyone is posing any queries or not. In computer speak, these steps are happening in the *background*:

1. **Gather information.** Jen explores the Web, visiting many sites on a regular basis to learn what they contain. Jen revisits old pages because their contents may have changed, and they may contain links to new pages that have never been visited.
2. **Keep copies.** Jen retains copies of many of the web pages she visits. Jen actually has a duplicate copy of a large part of the Web stored on her computers.
3. **Build an index.** Jen constructs a huge index that shows, at a minimum, which words appear on which web pages.

When you make a query, Jen goes through four more steps, in the *foreground*:

4. **Understand the query.** English has lots of ambiguities. A query like “red sox pitchers” is fairly challenging if you haven’t grown up with baseball!
5. **Determine the relevance of each possible result to the query.** Does the web page contain information the query asks about?
6. **Determine the ranking of the relevant results.** Of all the relevant answers, which are the “best”?
7. **Present the results.** The results need not only to be “good”; they have to be shown to you in a form you find useful, and perhaps also in a form that serves some of Jen’s other purposes—selling more advertising, for example.

Each of these seven steps involves technical challenges that computer scientists love to solve. Jen’s financial backers hope that her engineers solve them better than the engineers of competing search engines.

We’ll go through each step in more detail, as it is important to understand what is going on—at every step, more than technology is involved. Each step also presents opportunities for Jen to use her information-gathering and editorial powers in ways you may not have expected—ways that shape your view of the world through the lens of Jen’s search results.

The background processing is like the set-building and rehearsals for a theatrical production. You couldn’t have a show without it, but none of it happens while the audience is watching, and it doesn’t even need to happen on any particular schedule.

Step 1: Gather Information

Search engines don't index everything. The ones we think of as general utilities, such as Google, Yahoo!, and Ask, find information rather indiscriminately throughout the Web. Other search engines are domain-specific. For example, Medline searches only through medical literature. ArtCyclopedia indexes 2,600 art sites. The FindLaw LawCrawler searches only legal web sites. Right from the start, with any search engine, some things are in the index and some are out, because some sites are visited during the gathering step and others are not. Someone decides what is worth remembering and what isn't. If something is left out in Step 1, there is no possibility that you will see it in Step 7.

Speaking to the Association of National Advertisers in October 2005, Eric Schmidt, Google's CEO, observed that of the 5,000 terabytes of information in the world, only 170 terabytes had been indexed. (A *terabyte* is about a trillion bytes.) That's just a bit more than 3%, so 97% was not included. Another estimate puts the amount of indexed information at only .02% of the size of the databases and documents reachable via the Web. Even in the limited context of the World Wide Web, Jen needs to decide what to look at, and how frequently. These decisions implicitly define what is important and what is not, and will limit what Jen's users can find.

How *often* Jen visits web pages to index them is one of her precious trade secrets. She probably pays daily visits to news sites such as CNN.com, so that if you ask tonight about something that happened this morning, Jen may point you to CNN's story. In fact, there is most likely a master list of sites to be visited frequently, such as whitehouse.gov—sites that change regularly and are the object of much public interest. On the other hand, Jen probably has learned from her repeated visits that some sites don't change at all. For example, the Web version of a paper published ten years ago doesn't change. After a few visits, Jen may decide to revisit it once a year, just in case. Other pages may not be posted long enough to get indexed at all. If you post a futon for sale on Craigslist.com, the ad will become accessible to potential buyers in just a few minutes. If it sells quickly, however, Jen may never see it. Even if the ad stays up for a while, you probably won't be able to find it with most search engines for several days.

Jen is clever about how often she revisits pages—but her cleverness also codifies some judgments, some priorities—some *control*. The more important Jen judges your page to be, the less time it will take for your new content to show up as responses to queries to Jen's search engine.

Jen roams the Web to gather information by following links from the pages she visits. Software that crawls around the Web is (in typical geek

HOW A SPIDER EXPLORES THE WEB

Search engines gather information by wandering through the World Wide Web. For example, when a spider visits the main URL of the publisher of this book, www.pearson.com, it retrieves a page of text, of which this is a fragment:

```
<div id="subsidiary">
<h2 class="hide">Subsidiary sites links</h2>
<label for="subsidiarySites" class="hide">Available
sites</label>
<select name="subsidiarySites" id="subsidiarySites" size="1">
<option value="">Browse sites</option>
<optgroup label="FT Group">
<option value="http://www.ftchinese.com/sc/index.jsp">
Chinese.FT.com</option>
<option value="http://ftd.de/">FT Deutschland</option>
```

This text is actually a computer program written in a special programming language called HTML ("HyperText Markup Language"). Your web browser renders the web page by executing this little program. But the spider is retrieving this text not to render it, but to index the information it contains. "FT Deutschland" is text that appears on the screen when the page is rendered; such terms should go into the index. The spider recognizes other links, such as www.ftchinese.com or ftd.de, as URLs of pages it needs to visit in turn. In the process of visiting those pages, it indexes them and identifies yet more links to visit, and so on!

A spider, or web crawler, is a particular kind of *bot*. A bot (as in "robot") is a program that endlessly performs some intrinsically repetitive task, often an information-gathering task.

irony) called a "spider." Because the spidering process takes days or even weeks, Jen will not know immediately if a web page is taken down—she will find out only when her spider next visits the place where it used to be. At that point, she will remove it from her index, but in the meantime, she may respond to queries with links to pages that no longer exist. Click on such a link, and you will get a message such as "Page not found" or "Can't find the server."

Because the Web is unstructured, there is no inherently "correct" order in which to visit the pages, and no obvious way to know when to stop. Page A may contain references to page B, and also page B to page A, so the spider has to be careful not to go around in circles. Jen must organize her crawl of

the Web to visit as much as she chooses without wasting time revisiting sections she has already seen.

A web site may stipulate that it does not want spiders to visit it too frequently or to index certain kinds of information. The site's designer simply puts that information in a file named `robots.txt`, and virtually all web-crawling software will respect what it says. Of course, pages that are inaccessible without a login cannot be crawled at all. So, the results from Step 7 may be influenced by what the sites want Jen to know about them, as well as by what Jen thinks is worth knowing. For example, Sasha Berkovich was fortunate that the Polotsky family tree had been posted to part of the `genealogy.com` web site that was open to the public—otherwise, Google's spider could not have indexed it.

Finally, spidering is not cost free. Jen's "visits" are really requests to web sites that they send their pages back to her. Spidering creates Internet traffic and also imposes a load on the web server. This part of search engines' background processing, in other words, has unintended effects on the experience of the entire Internet. Spiders consume network bandwidth, and they may tie up servers, which are busy responding to spider requests while their ordinary users are trying to view their pages. Commercial search engines attempt to schedule their web crawling in ways that won't overload the servers they visit.

Step 2: Keep Copies

Jen downloads a copy of every web page her spider visits—this is what it means to "visit" a page. Instead of rendering the page on the screen as a web browser would, Jen indexes it. If she wishes, she can retain the copy after she has finished indexing it, storing it on her own disks. Such a copy is said to be "cached," after the French word for "hidden." Ordinarily Jen would not do anything with her cached copy; it may quickly become out of date. But caching web pages makes it possible for Jen to have a page that no longer exists at its original source, or a version of a page older than the current one. This is the flip side of Jen never knowing about certain pages because their owners took them down before she had a chance to index them. With a cached page, Jen knows what used to be on the page even after the owner intended it to disappear.

Caching is another blow to the Web-as-library metaphor, because removing information from the bookshelf doesn't necessarily get rid of it. Efforts to scrub even dangerous information are beyond the capability of those who posted it. For example, after 9/11, a lot of information that was once available on the Web was pulled. Among the pages that disappeared overnight

were reports on government vulnerabilities, sensitive security information, and even a Center for Disease Control chemical terrorism report that revealed industry shortcomings. Because the pages had been cached, however, the bits lived on at Google and other search engine companies.

Not only did those pages of dangerous information survive, but anyone could find them. Anytime you do a search with one of the major search engines, you are offered access to the cached copy, as well as the link to where the page came from, whether or not it still exists. Click on the link for the “Cached” page, and you see something that looks very much like what you might see if you clicked on the main link instead. The cached copy is identified as such (see Figure 4.3).

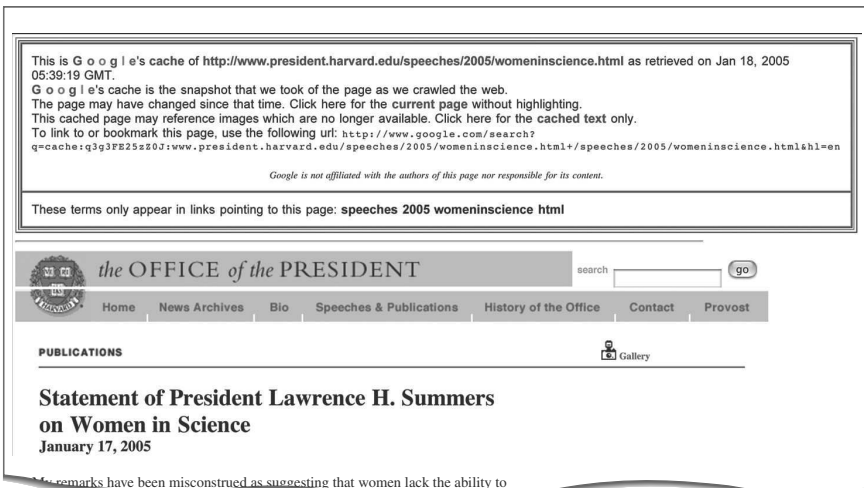


FIGURE 4.3 Part of a cached web page, Google's copy of an official statement made by Harvard's president and replaced two days later after negative public reaction. This copy was retrieved from Google after the statement disappeared from the university's web site. Harvard, which holds the copyright on this once-public statement, refused to allow it to be printed in this book (see Conclusion).

This is an actual example; it was the statement Lawrence Summers released on January 17, 2005, after word of his remarks about women in science became public. As reported in *Harvard Magazine* in March–April 2005, the statement began, “My remarks have been misconstrued as suggesting that women lack the ability to succeed at the highest levels of math and science. I did not say that, nor do I believe it.” This unapologetic denial stayed on the

The digital explosion grants the power of both instant communication and instant retraction—but almost every digital action leaves digital fingerprints.

carefully.” Those searching for the President’s statement were then led to the contrite new statement—but for a time, the original, defiant version remained visible to those who clicked on the link to Google’s cached copy.

FINDING DELETED PAGES

An easy experiment on finding deleted pages is to search using Google for an item that was sold on craigslist. You can use the “site” modifier in the Google search box to limit your search to the craigslist web site, by including a “modifier”:

`futon site:craigslist.com`

The results will likely return pages for items that are no longer available, but for which the cached pages will still exist.

Harvard web site for only a few days. In the face of a national firestorm of protest, Summers issued a new statement on January 19, 2005, reading, in part, “I deeply regret the impact of my comments and apologize for not having weighed them more

The digital explosion grants the power of both instant communication and instant retraction—but almost every digital action leaves digital fingerprints. Bits do not die easily, and digital words, once said, are hard to retract.

If Jen caches web pages, it may be possible for you to get information that was retracted after it was discovered to be in error or embarrassing. Something about this doesn’t feel quite right, though—is the information on those pages really Jen’s to do with as she wishes? If the material is copyrighted—a published paper from ten years ago, for example—

what right does Jen have to show you her cached copy? For that matter, what right did she have to keep a copy in the first place? If you have copyrighted something, don’t you have some authority over who can make copies of it?

This enigma is an early introduction to the confused state of copyright law in the digital era, to which we return in Chapter 6. Jen cannot index my web page without receiving a copy of it. In the most literal sense, any time you “view” or “visit” a web page, you are actually copying it, and then your web browser renders the copy on the screen. A metaphorical failure once again: The Web is *not a library*. Viewing is an exchange of bits, not a passive activity, as far as the web site is concerned. If “copying” copyrighted materials was totally prohibited, neither search engines nor the Web itself could work, so some sort of copying must be permissible. On the other hand, when Jen caches the material she indexes—perhaps an entire book, in the case of the

Google Books project—the legal controversies become more highly contested. Indeed, as we discuss in Chapter 6, the Association of American Publishers and Google are locked in a lawsuit over what Google is and is not allowed to do with the digital images of books that Google has scanned.

Step 3: Build an Index

When we searched the Web for “Zyprexa,” Jen consulted her index, which has the same basic structure as the index of a book: a list of terms followed by the places they occur. Just as a book’s index lists page numbers, Jen’s index lists URLs of web pages. To help the search engine give the most useful responses to queries, the index may record other information as well: the size of the font in which the term appears, for example, and where on the page it appears.

Indexes are critical because having the index in order—like the index

of a book, which is in alphabetical order—makes it possible to find things much faster than with sequential searching. This is where Jen’s computer scientists really earn their salaries, by devising clever ways of storing indexed information so it can be retrieved quickly. Moore’s Law also played a big role in the creation of web indexes—until computer memories got fast enough, cheap enough, and big enough, even the cleverest computer scientists could not program machines to respond instantly to arbitrary English queries.

When Jen wants to find a term in her index, she does not start at the beginning and go through it one entry at a time until she finds what she is looking for. That is not the way you would look up something in the index of a book; you would use the fact that the index is in order alphabetically. A very simple strategy to look up something in a big ordered index, such as a phone book, is just to open the book in the middle and see if the item you are looking for belongs in the first half or the second. Then you can ignore half the phone book and use the same strategy to subdivide the remaining half. The number of steps it takes to get down to a single page in a phone book with n pages using this method is the number of times you have to divide n by 2 to get down to 1. So if n is 1000, it takes only 10 of these probing steps to find any item using *binary search*, as this method is known.

INDEXES AND CONCORDANCES

The information structure used by search engines is technically known as an *inverted index*—that is, an index of the words in a document or a set of documents, and the places where those words appear. Inverted indexes are not a new idea; the biblical concordances laboriously constructed by medieval monks were inverted indexes. Constructing concordances was one of the earliest applications of computer technology to a nonmathematical problem.

In general, the number of steps needed to search an index of n things using binary search is proportional, not to n , but to the number of digits in n . That means that binary search is exponentially faster than linear search—searching through a million items would take only 20 steps, and through a billion items would take 30 steps. And binary search is fairly dumb by comparison with what people actually do—if you were looking for “Ledeen” in the phone book, you might open it in the middle, but if you were looking for “Abelson,” you’d open it near the front. That strategy can be reduced to an even better computer algorithm, exponentially faster than binary search.

How big is Jen’s index, in fact? To begin with, how many terms does Jen index? That is another of her trade secrets. Jen’s index could be useful with a few tens of millions of entries. There are fewer than half a million words in the English language, but Jen probably wants to index some numbers too (try searching for a number such as 327 using your search engine). Proper names and at least some words in foreign languages are also important. The list of web pages associated with a term is probably on disk in most cases, with only the information about *where* on the disk kept with the term itself in main memory. Even if storing the term and the location on disk of the list of associated URLs takes 100 bytes per entry, with 25 million entries, the table of index entries would occupy 2.5 gigabytes (about 2.5 billion bytes) of main memory. A few years ago, that amount of memory was unimaginable; today, you get that on a laptop from Wal-Mart. The index can be searched quickly—using binary search, for example—although retrieving the list of URLs might require going to disk. If Jen has Google’s resources, she can speed up her query response by keeping URLs in main memory too, and she can split the search process across multiple computers to make it even faster.

Now that the preparations have been made, we can watch the performance itself—what happens when you give Jen a query.

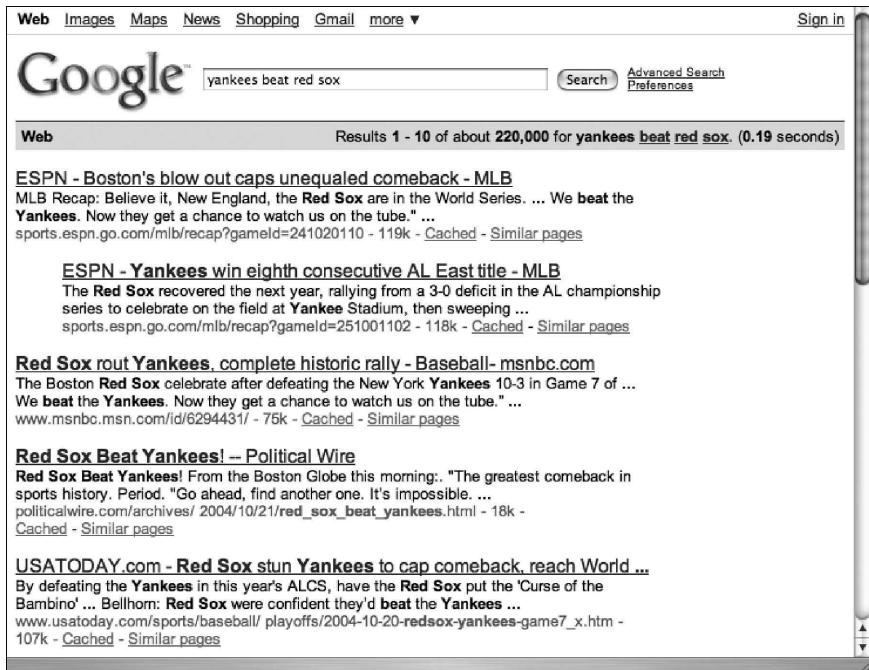
Step 4: Understand the Query

When we asked Google the query *Yankees beat Red Sox*, only one of the top five results was about the Yankees beating the Red Sox (see Figure 4.4). The others reported instead on the Red Sox beating the Yankees. Because English is hard for computers to understand and is often ambiguous, the simplest form of query analysis ignores syntax, and treats the query as simply a list of keywords. Just looking up a series of words in an index is computationally easy, even if it often misses the intended meaning of the query.

To help users reduce the ambiguity of their keyword queries, search engines support “advanced queries” with more powerful features. Even the simplest, putting a phrase in quotes, is used by fewer than 10% of search

engine users. Typing the quotation marks in the query “Red Sox beat Yankees” produces more appropriate results. You can use “~” to tell Google to find synonyms, “-” to exclude certain terms, or cryptic commands such as “allinurl:” or “inanchor:” to limit the part of the Web to search. Arguably we didn’t ask our question the right way, but most of us don’t bother; in general, people just type in the words they want and take the answers they get.

Often they get back quite a lot. Ask Yahoo! for the words “allergy” and “treatment,” and you find more than 20,000,000 references. If you ask for “allergy treatment”—that is, if you just put quotes around the two words—you get 628,000 entries, and quite different top choices. If you ask for “treating allergies,” the list shrinks to 95,000. The difference between these queries may have been unintentional, but the search engine thought they were drastically different. It’s remarkable that human-computer communication through the lens of the search engine is so useful, given its obvious imperfections!



Google™ is a registered trademark of Google, Inc. Reprinted by permission.

FIGURE 4.4 Keyword search misses the meaning of English-language query. Most of the results for the query “Yankees beat Red Sox” are about the Red Sox beating the Yankees.

NATURAL LANGUAGE QUERIES

Query-understanding technology is improving. The experimental site www.digger.com, for example, tells you when your query is ambiguous and helps you clarify what you are asking. If you ask Digger for information about "java," it realizes that you might mean the beverage, the island, or the programming language, and helps get the right interpretation if it guessed wrong the first time.

Powerset (www.powerset.com) uses natural language software to disambiguate queries based on their English syntax, and answers based on what web pages actually say. That would resolve the misunderstanding of "Yankees beat Red Sox."

Ongoing research promises to transfer the burden of disambiguating queries to the software, where it belongs, rather than forcing users to twist their brains around computerese. Natural language understanding seems to be on its way, but not in the immediate future. We may need a hundred-fold increase in computing power to make semantic analysis of web pages accurate enough so that search engines no longer give boneheaded answers to simple English queries.

Today, users tend to be tolerant when search engines misunderstand their meaning. They blame themselves and revise their queries to produce better results. This may be because we are still amazed that search engines work at all. In part, we may be tolerant of error because in web search, the cost to the user of an inappropriate answer is very low. As the technology improves, users will expect more, and will become less tolerant of wasting their time sorting through useless answers.

Step 5: Determine Relevance

A search engine's job is to provide results that match the intent of the query. In technical jargon, this criterion is called "relevance." Relevance has an objective component—a story about the Red Sox beating the Yankees is only marginally responsive to a query about the Yankees beating the Red Sox. But relevance is also inherently subjective. Only the person who posed the query can be the final judge of the relevance of the answers returned. In typing my query, I probably meant the New York Yankees beating the Boston Red Sox of Major League Baseball, but I didn't say that—maybe I meant the Flagstaff Yankees and the Continental Red Sox of Arizona Little League Baseball.

Finding all the relevant documents is referred to as “recall.” Because the World Wide Web is so vast, there is no reasonable way to determine if the search engine is finding everything that is relevant. Total recall is unachievable—but it is also unimportant. Jen could give us thousands or even millions more responses that she judges to be relevant, but we are unlikely to look beyond the first page or two. Degree of relevance always trumps level of recall. Users want to find a few good results, not all possible results.

The science of measuring relevance is much older than the Web; it goes back to work by Gerald Salton in the 1960s, first at Harvard and later at Cornell. The trick is to automate a task when what counts as success has such a large subjective component. We want the computer to scan the document, look at the query, do a few calculations, and come up with a number suggesting how relevant the document is to the query.

As a very simple example of how we might calculate the relevance of a document to a query, suppose there are 500,000 words in the English language. Construct two lists of 500,000 numbers: one for the document and one for the query. Each position in the lists corresponds to one of the 500,000 words—for example, position #3682 might be for the word “drugs.” For the document, each position contains a count of the number of times the corresponding word occurs in the document. Do the same thing for the query—unless it contains repeated words, each position will be 1 or 0. Multiply the lists for the document and the query, position by position, and add up the 500,000 results. If no word in the query appears in the document, you’ll get a result of 0; otherwise, you will get a result greater than 0. The more frequently words from the query appear in the document, the larger the results will be.

SEARCH ENGINES AND INFORMATION RETRIEVAL

Three articles offer interesting insights into how search engines and information retrieval work:

“The Anatomy of a Large-Scale Hypertextual Web Search Engine” by Sergey Brin and Larry Page was written in 2000 and gives a clear description of how the original Google worked, what the goal was, and how it was differentiated from earlier search engines.

“Modern Information Retrieval: A Brief Overview” by Amit Singhal was written in 2001 and surveys the IR scene. Singhal was a student of Gerry Salton and is now a Google Fellow.

“The Most Influential Paper Gerald Salton Never Wrote” by David Dubin presents an interesting look at some of the origins of the science.

Figure 4.5 shows how the relevance calculation might proceed for the query “Yankees beat Red Sox” and the visible part of the third document of Figure 4.4, which begins, “Red Sox rout Yankees” (The others probably contain more of the keywords later in the full document.) The positions in the two lists correspond to words in a dictionary in alphabetical order, from “ant” to “zebra.” The words “red” and “sox” appear two times each in the snippet of the story, and the word “Yankees” appears three times.

Lexicon:	ant, ...,	beat, ...,	defeating, ...,	new, ...,	patriots, ...,	red, ...,	sox, ...,	Yankees, ...,	zebra, ...
Doc:	0, ...,	1, ...,	2, ...,	1, ...,	0, ...,	2, ...,	2, ...,	3, ...,	0, ...
Query:	0, ...,	1, ...,	0, ...,	0, ...,	0, ...,	1, ...,	1, ...,	1, ...,	0, ...
<hr/>									
Doc									
×	0, ...,	1, ...,	0, ...,	0, ...,	0, ...,	2, ...,	2, ...,	3, ...,	0, ...
Query									
<hr/>									
Sum of elements of Doc × Query = 1+2+2+3 = 8 = “relevance” of document to query									

FIGURE 4.5 Document and query lists for relevance calculation.

That is a very crude relevance calculation—problems with it are easy to spot. Long documents tend to be measured as more relevant than short documents, because they have more word repetitions. Uninteresting words such as “from” add as much to the relevance score as more significant terms such as “Yankees.” Web search engines such as Google, Yahoo!, MSN, and Ask.com consider many other factors in addition to which words occur and how often. In the list for the document, perhaps the entries are not word counts, but another number, adjusted so words in the title of the page get greater weight. Words in a larger font might also count more heavily. In a query, users tend to type more important terms first, so maybe the weights should depend on where words appear in the query.

Step 6: Determine Ranking

Once Jen has selected the relevant documents—perhaps she’s chosen all the documents whose relevance score is above a certain threshold—she “ranks” the search results (that is, puts them in order). Ranking is critical in making the search useful. A search may return thousands of relevant results, and users want to see only a few of them. The simplest ranking is by relevance—putting the page with the highest relevance score first. That doesn’t work well, however. For one thing, with short queries, many of the results will have approximately the same relevance.

More fundamentally, the documents Jen returns should be considered “good results” not just because they have high relevance to the query, but also because the documents themselves have high quality. Alas, it is hard to say what “quality” means in the search context, when the ultimate test of success is providing what people want. In the example of the earlier sidebar, who is to judge whether the many links to material about Britney Spears are really “better” answers to the “spears” query than the link to Professor Spears? And whatever “quality” may be, the ranking process for the major web search engines takes place automatically, without human intervention. There is no way to include protocols for checking professional licenses and past convictions for criminal fraud—not in the current state of the Web, at least.

Even though quality can’t be measured automatically, something like “importance” or “reputation” can be extracted from the structure of linkages that holds the Web together. To take a crude analogy, if you think of web pages as scientific publications, the reputations of scientists tend to rise if their work is widely cited in the work of other scientists. That’s far from a

WHAT MAKES A PAGE SEARCHABLE

No search provider discloses the full details of its relevance and ranking algorithm. The formulas remain secret because they offer competitive advantages, and because knowing what gives a page high rank makes abuse easier. But here are some of the factors that might be taken into account:

- Whether a keyword is used in the title of the web page, a major heading, or a second-level heading
- Whether it appears only in the body text, and if so, how “prominently”
- Whether the web site is considered “trustworthy”
- Whether the pages linked to from within the page are themselves relevant
- Whether the pages that link to this page are relevant
- Whether the page is old or young
- Whether the pages it links to are old or young
- Whether it passes some objective quality metric—for example, not containing any misspellings

Once you go to the trouble of crawling the Web, there is plenty to analyze, if you have the computing power to do it!

perfect system for judging the importance of scientific work—junk science journals do exist, and sometimes small groups of marginal scientists form mutual admiration societies. But for the Web, looking at the linkage structure is a place to start to measure the significance of pages.

One of Google's innovations was to enhance the relevance metric with another numerical value called "PageRank." PageRank is a measure of the "importance" of each a page that takes into account the external references to it—a World Wide Web popularity contest. If more web pages link to a particular page, goes the logic, it must be more important. In fact, a page should be judged more important if a lot of *important* pages link to it than if the same number of unimportant pages link to it. That seems to create a circular definition of importance, but the circularity can be resolved—with a bit of mathematics and a lot of computing power.

This way of ranking the search results seems to reward reputation and to be devoid of judgment—it is a mechanized way of aggregating mutual opinions. For example, when we searched using Google for "schizophrenia drugs," the top result was part of the site of a Swedish university. Relevance was certainly part of the reason that page came up first; the page was specifically about drugs used to treat schizophrenia, and the words "schizophrenia" and "drugs" both appeared in the title of the page. Our choice of words affected the relevance of the page—had we gone to the trouble to type "medicines" instead of "drugs," this link wouldn't even have made it to the first page of search results. Word order matters, too—Google returns different results for "drugs schizophrenia" than for "schizophrenia drugs."

Sergey Brin and Larry Page, Google's founders, were graduate students at Stanford when they developed the company's early technologies. The "Page" in "PageRank" refers not to web pages, but to Larry Page.

This page may also have been ranked high because many other web pages contained references to it, particularly if many of those pages were themselves judged to be important. Other pages about schizophrenia drugs may have used better English prose style, may have been written by more respected scientific authorities, and may have contained more up-to-date

information and fewer factual errors. The ranking algorithm has no way to judge any of that, and no one at Google reads every page to make such judgments.

Google, and other search engines that rank pages automatically, use a secret recipe for ranking—a pinch of this and a dash of that. Like the formula

for Coca-Cola, only a few people know the details of commercial ranking algorithms. Google's algorithm is patented, so anyone can read a description. Figure 4.6 is an illustration from that patent, showing several pages with links to each other. This illustration suggests that both the documents themselves and the links between them might be assigned varying numbers as measures of their importance. But the description omits many details and, as actually implemented, has been adjusted countless times to improve its performance. A company's only real claim for the validity of its ranking formula is that people like the results it delivers—if they did not, they would shift to one of the competing search engines.

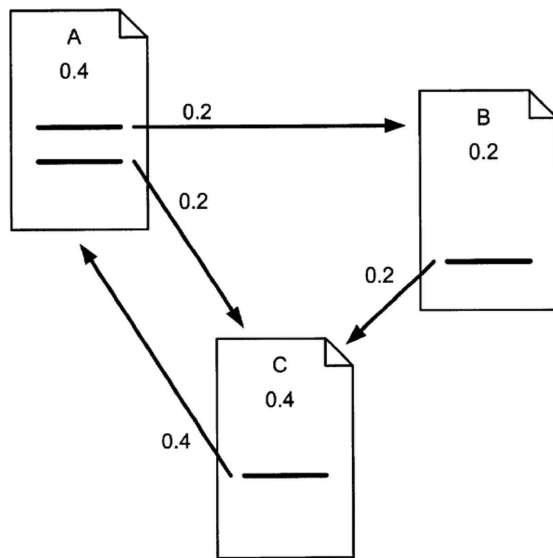


FIGURE 4.6 A figure from the PageRank patent (U.S. Patent #6285999), showing how links between documents might receive different weights.

It may be that one of the things people like about their favored search engine is consistently getting what they believe to be unbiased, useful, and even truthful information. But “telling the truth” in search results is ultimately only a means to an end—the end being greater profits for the search company.

Ranking is a matter of opinion. But a lot hangs on those opinions. For a user, it usually does not matter very much which answer comes up first or whether any result presented is even appropriate to the query. But for a

company offering a product, where it appears in the search engine results *can* be a matter of life and death.

KinderStart (www.kinderstart.com) runs a web site that includes a directory and search engine focused on products and services for young children. On March 19, 2005, visits to its site declined by 70% when Google lowered its PageRank to zero (on a scale of 0 to 10). Google may have deemed KinderStart's page to be low quality because its ranking algorithm found the page to consist mostly of links to other sites. Google's public description of its criteria warns about pages with "little or no original content." KinderStart saw matters differently and mounted a class action lawsuit against Google, claiming, among other things, that Google had violated its rights to free speech under the First Amendment by making its web site effectively invisible. Google countered that KinderStart's low PageRank was just Google's opinion, and opinions were not matters to be settled in court:

Google, like every other search engine operator, has made that determination for its users, exercising its judgment and expressing its opinion about the relative significance of web sites in a manner that has made it the search engine of choice for millions. Plaintiff KinderStart contends that the judiciary should have the final say over that editorial process.

SEEING A PAGE'S PAGERANK

Google has a toolbar you can add to certain browsers, so you can see PageRanks of web pages. It is downloadable from toolbar.google.com. You can also use the site www.iwebtool.com/pagerank_checker to enter a URL in a window and check its PageRank.

No fair, countered KinderStart to Google's claim to be just expressing an opinion. "PageRank," claimed KinderStart, "is not a mere statement of opinion of the innate value or human appeal of a given web site and its web pages," but instead is "a mathematically-generated product of measuring and assessing the quantity and depth of all the hyperlinks on the Web that tie into PageRanked web site, under programmatic determination by Defendant Google."

The judge rejected every one of KinderStart's contentions—and not just the claim that KinderStart had a free speech right to be more visible in Google searches. The judge also rejected claims that Google was a monopoly guilty of antitrust violations, and that KinderStart's PageRank of zero amounted to a defamatory statement about the company.

Whether it's a matter of opinion or manipulation, KinderStart is certainly much easier to find using Yahoo! than Google. Using Yahoo!, kinderstart.com is the top item returned when searching for "kinderstart." When we used Google, however, it did not appear until the twelfth page of results.

A similar fate befell bmw.de, the German web page of automaker BMW. The page Google indexed was straight text, containing the words "gebrauchtwagen" and "neuwagen" ("used car" and "new car") dozens of times. But a coding trick caused viewers instead to see a more conventional page with few words and many pictures. The effect was to raise BMW's position in searches for "new car" and "used car," but the means violated Google's clear instructions to web site designers: "Make pages for users, not for search engines. Don't deceive your users or present different content to search engines than you display to users, which is commonly referred to as 'cloaking.'" Google responded with a "death penalty"—removing bmw.de from its index. For a time, the page simply ceased to exist in Google's universe. The punitive measure showed that Google was prepared to act harshly against sites attempting to gain rank in ways it deemed consumers would not find helpful—and at the same time, it also made clear that Google was prepared to take *ad hoc* actions against individual sites.

Step 7: Presenting Results

After all the marvelous hard work of Steps 1–6, search engines typically provide the results in a format that is older than Aristotle—the simple, top-to-bottom list. There are less primitive ways of displaying the information.

If you search for something ambiguous like "washer" with a major web search engine, you will be presented with a million results, ranging from clothes washers to software packages that remove viruses. If you search Home Depot's web site for "washer," you will get a set of automatically generated choices to assist you in narrowing the search: a set of categories, price ranges, brand names, and more, complete with pictures (see Figure 4.7).

Alternatives to the simple rank-ordered list for presenting results better utilize the visual system. Introducing these new forms of navigation may shift the balance of power in the search equation. Being at the top of the list may no longer have the same economic value, but something else may replace the currently all-important rank of results—quality of the graphics, for example.

No matter how the results are presented, something else appears alongside them, and probably always will. It is time to talk about those words from the sponsors.

THE HOME DEPOT You can do it. We can help.™

SHOPPING CART ORDER STATUS MY LIST MY REGISTRY MY ACCOUNT SIGN IN

GET INSPIRED

Appliances Bath Building Supplies Décor Doors & Windows Electronics Flooring Kitchen Lighting & Fans Outdoors Paint Storage Tools & Hardware

Enter Keyword or SKU SEARCH Gift Cards Gifts Promotions Know-How Pro Credit Home Services Weekly Ad Store Finder Help

You are here: HOME > Text Search > washers

Shop Online & Browse Store Products

Online Products Online & Store Products

Category

- Appliances (175)
- Bath (1)
- Building Supplies (2)
- Outdoors (6)
- Tools & Hardware (21)

Price

- Less than \$50 (20)
- \$50 - 100 (6)
- \$100 - 200 (1)
- \$200 - 400 (37)
- \$400 - 600 (47)
- \$600 - 800 (35)
- \$800 - 1000 (43)
- \$1000 - 2000 (14)

Brand

- Admiral® (3)
- Amana® (3)
- DeWALT (6)
- GE (79)
- GE Profile (13)
- Haier America (4)
- Hotpoint (9)
- LG Electronics (23)
- Maytag® (40)
- Ramset (11)
- More...

Energy Star Compliant

- Energy Star (33)

MORE WAYS TO SHOP

- Shop By Brand
- What's New
- Most Popular
- For Contractors
- Registry

Search Results

You Searched for "washers"

210 Results: 203 Products, 7 Articles





Matching Categories include:

- Appliances > Washers & Dryers
- Appliances > Washers & Dryers > Washers
- Building Supplies > Plumbing > Maintenance & Repair > Faucet > Washers
- Outdoors > Outdoor Power Equipment > Pressure Washers
- Outdoors > Outdoor Power Equipment > Pressure Washers > Pressure Washer Accessories

203 Products Sort By: Best Match

View Products in a: Grid | List Results per page: 12 1 2 3 4 5 ▶

Select up to 4 items to compare. COMPARE

<input type="checkbox"/> Select to compare	<input type="checkbox"/> Select to compare	<input type="checkbox"/> Select to compare	<input type="checkbox"/> Select to compare
			
Hot Washer Screw	GE GE® 3.5 Cu. Ft. King-size Capacity Frontload Washer with Stainless Steel Basket	GE GE® 3.2 Cu. Ft. Super Capacity Washer	Maytag® Bravos High-Efficiency Top-Load Washer
Model TA-9	Model WSSH900GWW	Model WDGR2080GWW	Model MTW6600TQ
\$3.44 Free Shipping	\$549.00	\$319.00	\$899.00

Source: Home Depot.

FIGURE 4.7 Results page from a search for “washers” on the Home Depot web site.

Who Pays, and for What?

Web search is one of the most widely used functions of computers. More than 90% of online adults use search engines, and more than 40% use them on a typical day. The popularity of search engines is not hard to explain. Search engines are generally free for anyone to use. There are no logins, no fine print to agree to, no connection speed parameters to set up, and no personal information to be supplied that you’d rather not give away. If you have an Internet connection, then you almost certainly have a web browser, and it probably comes with a web search engine on its startup screen. There are no directions

dominant advertising engine. Among the items and services for which Google will not accept advertisements are fake designer goods, child pornography (some adult material is permitted in the U.S., but not if the models *might* be underage), term paper writing services, illegal drugs and some legal herbal substances, drug paraphernalia, fireworks, online gambling, miracle cures, political attack ads (although political advertising is allowed in general), prostitution, traffic radar jammers, guns, and brass knuckles. The list paints a striking portrait of what Joe and Mary Ordinary want to see, should see, or will tolerate seeing—and perhaps also how Google prudentially restrains the use of its powerfully liberating product for illegal activities.

Search Is Power

At every step of the search process, individuals and institutions are working hard to control what we see and what we find—not to do us ill, but to help us. Helpful as search engines are, they don't have panels of neutral experts deciding what is true or false, or what is important or irrelevant. Instead, there are powerful economic and social motivations to present information that is to our liking. And because the inner workings of the search engines are not visible, those controlling what we see are themselves subject to few controls.

Algorithmic Does Not Mean Unbiased

Because search engines compute relevance and ranking, because they are “algorithmic” in their choices, we often assume that they, unlike human researchers, are immune to bias. But bias can be coded into a computer program, introduced by small changes in the weights of the various factors that go into the ranking recipe or the spidering selection algorithm. And even what *counts* as bias is a matter of human judgment.

Having a lot of money will not buy you a high rank by paying that money to Google. Google's PageRank algorithm nonetheless incorporates something of a bias in favor of the already rich and powerful. If your business has become successful, a lot of other web pages are likely to point to yours, and that increases your PageRank. This makes sense and tends to produce the results that most people feel are correct. But the degree to which power should beget more power is a matter over which powerful and marginal businesses might have different views. Whether the results “seem right,” or the search algorithm's parameters need adjusting, is a matter only humans can judge.

For a time, Amazon customers searching for books about abortion would get back results including the question, “Did you mean adoption?” When a pro-choice group complained, Amazon responded that the suggestion was automatically generated, a consequence of the similarity of the words. The search engine had noticed, over time, that many people who searched for “abortion” also searched for “adoption.” But Amazon agreed to make the *ad hoc* change to its search algorithm to treat the term “abortion” as a special case. In so doing, the company unintentionally confirmed that its algorithms sometimes incorporate elements of human bias.

Market forces are likely to drive commercially viable search engines toward the bias of the majority, and also to respond to minority interests only in proportion to their political power. Search engines are likely to favor fresh items over older and perhaps more comprehensive sources, because their users go to the Internet to get the latest information. If you rely on a search engine to discover information, you need to remember that others are making judgment calls for you about what you are being shown.

Not All Search Engines Are Equal

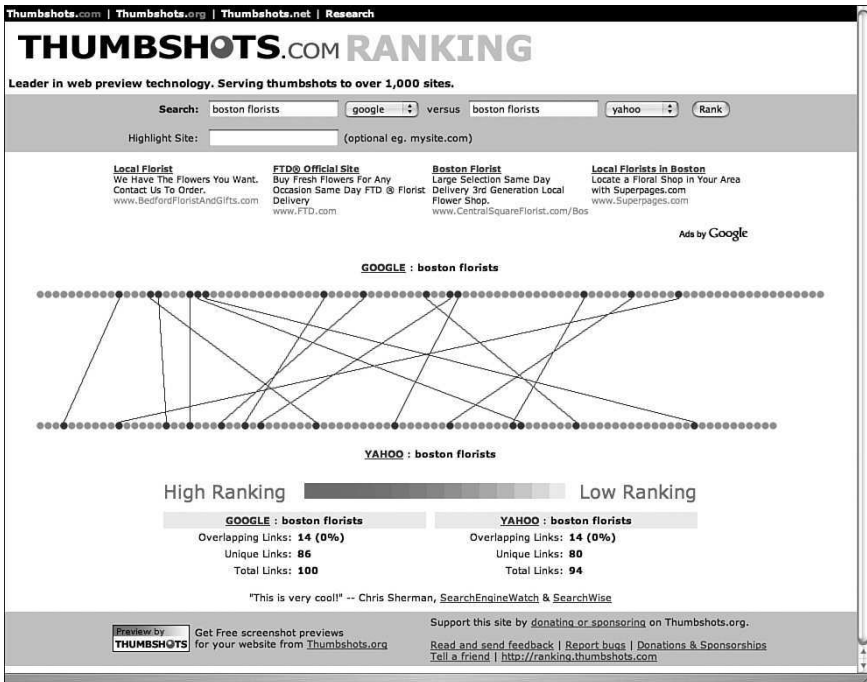
When we use a search engine, we may think that what we are getting is a representative sample of what’s available. If so, what we get from one search engine should be pretty close to what we get from another. This is very far from reality.

A study comparing queries to Google, Yahoo!, ASK, and MSN showed that the results returned on the first page were unique 88% percent of the time. Only 12% of the first-page results were in common to even two of these four search engines. If you stick with one search engine, you could be missing what you’re looking for. The tool ranking.thumbshots.com provides vivid graphic representations of the level of overlap between the results of different search engines, or different searches using the same search engine. For example, Figure 4.9 shows how little overlap exists between Google and Yahoo! search results for “boston florist.”

Each of the hundred dots in the top row represents a result of the Google search, with the highest-ranked result at the left. The bottom row represents Yahoo!’s results. A line connects each pair of identical search results—in this case, only 11% of the results were in common. Boston Rose Florist, which is Yahoo’s number-one response, doesn’t turn up in Google’s search at all—not in the top 100, or even in the first 30 pages Google returns.

Ranking determines visibility. An industry research study found that 62% of search users click on a result from the first page, and 90% click on a result within the first three pages. If they don’t find what they are looking for, more

than 80% start the search over with the same search engine, changing the keywords—as though confident that the search engine “knows” the right answer, but they haven’t asked the right question. A study of queries to the Excite search engine found that more than 90% of queries were resolved in the first three pages. Google’s experience is even more concentrated on the first page.



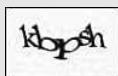
Reprinted with permission of SmartDevil, Inc.

FIGURE 4.9 Thumbshots comparison of Google and Yahoo! search results for “boston florists.”

Search engine users have great confidence that they are being given results that are not only useful but authoritative. 36% of users thought seeing a company listed among the top search results indicated that it was a top company in its field; only 25% said that seeing a company ranked high in search results would not lead them to think that it was a leader in its field. There is, in general, no reason for such confidence that search ranking corresponds to corporate quality.

CAT AND MOUSE WITH BLOG SPAMMERS

You may see comments on a blog consisting of nothing but random words and a URL. A malicious bot is posting these messages in the hope that Google's spider will index the blog page, including the spam URL. With more pages linking to the URL, perhaps its PageRank will increase and it will turn up in searches. Blogs counter by forcing you to type some distorted letters—a so-called *captcha* ("Completely Automated Public Turing test to tell Computers and Humans Apart"), a test to determine if the party posting the comment is really a person and not a bot. Spammers counter by having their bot take a copy of the captcha and show it to human volunteers. The spam bot then takes what the volunteers type and uses it to gain entry to the blog site. The volunteers are recruited by being given access to free pornography if they type the captcha's text correctly! Here is a sample captcha:



This image has been released into the public domain by its author, Kruglov at the wikipedia project. This applies worldwide.

Search Results Can Be Manipulated

Search is a remarkable business. Internet users put a lot of confidence in the results they get back from commercial search engines. Buyers tend to click on the first link, or at least a link on the first page, even though those links may depend heavily on the search engine they happen to be using, based on complex technical details that hardly anyone understands. For many students, for example, the library is an information source of last resort, if that. They do research as though whatever their search engine turns up must be a link to the truth. If people don't get helpful answers, they tend to blame themselves and change the question, rather than try a different search engine—even though the answers they get can be inexplicable and capricious, as anyone googling "kinderstart" to find kinderstart.com will discover.

Under these circumstances, anyone putting up a web site to get a message out to the world would draw an obvious conclusion. Coming out near the top of the search list is too important to leave to chance. Because ranking is algorithmic, a set of rules followed with diligence and precision, it must be possible to manipulate the results. The Search Engine Optimization industry (SEO) is based on that demand.

Search Engine Optimization is an activity that seeks to improve how particular web pages rank within major search engines, with the intent of

increasing the traffic that will come to those web sites. Legitimate businesses try to optimize their sites so they will rank higher than their competitors. Pranksters and pornographers try to optimize their sites, too, by fooling the search engine algorithms into including them as legitimate results, even though their trappings of legitimacy are mere disguises. The search engine companies tweak their algorithms in order to see through the disguises, but their tweaks sometimes have unintended effects on legitimate businesses. And the tweaking is largely done in secret, to avoid giving the manipulators any ideas about countermeasures. The result is a chaotic battle, with innocent bystanders, who have become reliant on high search engine rankings, sometimes injured as the rules of engagement keep changing.

Google proclaims of its PageRank algorithm that “Democracy on the web works,” comparing the ranking-by-inbound-links to a public election. But the analogy is limited—there are many ways to manipulate the “election,” and the voting rules are not fully disclosed.

The key to search engine optimization is to understand how particular engines do their ranking—what factors are considered, and what weights they are given—and then to change your web site to improve your score. For example, if a search engine gives greater weight to key words that appear in the title, and you want your web page to rank more highly when someone searches for “cameras,” you should put the word “cameras” in the title. The weighting factors may be complex and depend on factors external to your own web page—for example, external links that point to your page, the age of the link, or the prestige of the site from which it is linked. So significant time, effort, and cost must be expended in order to have a meaningful impact on results.

Then there are techniques that are sneaky at best—and “dirty tricks” at worst. Suppose, for example, that you are the web site designer for Abelson’s, a new store that wants to compete with Bloomingdale’s. How would you entice people to visit Abelson’s site when they would ordinarily go to Bloomingdale’s? If you put “We’re better than Bloomingdale’s!” on your web page, Abelson’s page might appear in the search results for “Bloomingdale’s.” But you might not be willing to pay the price of mentioning the competition on Abelson’s page. On the other hand, if you just put the word “Bloomingdale’s” *in white text on a white background* on Abelson’s page, a human viewer wouldn’t see it—but the indexing software might index it anyway. The indexer is working with the HTML code that generates the page, not the visible page itself. The software might not be clever enough to realize that the word “Bloomingdale’s” in the HTML code for Abelson’s web page would not actually appear on the screen.

A huge industry has developed around SEO, rather like the business that has arisen around getting high school students packaged for application to college. A Google search for “search engine optimization” returned 11 sponsored links, including some with ads reading “Page 1 Rankings Guarantee” and “Get Top Rankings Today.”

Is the search world more ethical because the commercial rank-improving transactions are indirect, hidden from the public, and going to the optimization firms rather than to the search firms? After all, it is only logical that if you have an important message to get out, you would optimize your site to do so. And you probably wouldn't have a web site at all if you thought you had nothing important to say. Search engine companies tend to advise their web site designers just to create better, more substantive web pages, in much the same way that college admissions officials urge high school students just to learn more in school. Neither of the dependent third-party “optimization” industries is likely to disappear anytime soon because of such principled advice.

And what's “best”—for society in general, not just for the profits of the search companies or the companies that rely on them—can be very hard to say. In his book, *Ambient Findability*, Peter Morville describes the impact of search engine optimization on the National Cancer Institute's site, www.cancer.gov. The goal of the National Cancer Institute is to provide the most reliable and the highest-quality information to people who need it the most,

GOOGLE BOMBING

A “Google bomb” is a prank that causes a particular search to return mischievous results, often with political content. For example, if you searched for “miserable failure” after the 2000 U.S. presidential election, you got taken to the White House biography of George Bush. The libertarian Liberty Round Table mounted an effort against the Center for Science in the Public Interest, among others. In early 2008, www.libertyroundtable.org read, “Have you joined the Google-bombing fun yet? Lob your volleys at the food nazis and organized crime. Your participation can really make the difference with this one—read on and join the fun! Current Target: Verizon Communications, for civil rights violations.” The site explains what HTML code to include in your web page, supposedly to trick Google's algorithms.

Marek W., a 23-year-old programmer from Cieszyn, Poland, “Google bombed” the country's president, Lech Kaczyński. Searches for “kutas” using Google (it's the Polish word for “penis”) returned the president's web site as the first choice. Mr. Kaczyński was not pleased, and insulting the president is a crime in Poland. Marek is now facing three years in prison.

often cancer sufferers and their families. Search for “cancer,” and the NCI site was “findable” because it appeared near the topic of the search page results. That wasn’t the case, though, when you looked for specific cancers, yet that’s exactly what the majority of the intended users did. NCI called in search engine optimization experts, and all that is now changed. If we search for “colon cancer,” the specific page on the NCI site about this particular form of cancer appears among the top search results.

Is this good? Perhaps—if you can’t trust the National Cancer Institute, who *can* you trust? But WebMD and other commercial sites fighting for the top position might not agree. And a legitimate coalition, the National Colorectal Cancer Roundtable, doesn’t appear until page 7, too deep to be noticed by almost any user.

Optimization is a constant game of cat and mouse. The optimizers look for better ways to optimize, and the search engine folks look for ways to produce more reliable results. The game occasionally claims collateral victims. Neil Montcrief, an online seller of large-sized shoes, prospered for a while because searches for “big feet” brought his store, 2bigfeet.com, to the top of the list. One day, Google tweaked its algorithm to combat manipulation. Montcrief’s innocent site fell to the twenty-fifth page, with disastrous consequences for his economically marginal and totally web-dependent business.

Manipulating the ranking of search results is one battleground where the power struggle is played out. Because search is the portal to web-based information, controlling the search results allows you, perhaps, to control what people think. So even governments get involved.

Search Engines Don’t See Everything

Standard search engines fail to index a great deal of information that is accessible via the Web. Spiders may not penetrate into databases, read the contents of PDF or other document formats, or search useful sites that require a simple, free registration. With a little more effort than just typing into the search window of Google or Yahoo!, you may be able to find exactly what you are looking for. It is a serious failure to assume that something is unimportant or nonexistent simply because a search engine does not return it. A good overview of resources for finding things in the “deep web” is at Robert Lackie’s web site, www.robertlackie.com.

Search Control and Mind Control

To make a book disappear from a library, you don’t have to remove it from the bookshelf. All you need to do is to remove its entry from the library

All algorithms are not equal

This activity introduces the concept that algorithms are used to develop and express solutions to computational problems. It focus, in part, on the following learning objectives:

- 15: The student can develop an algorithm.
- 16: The student can express an algorithm in a language.
- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Comparing Algorithms

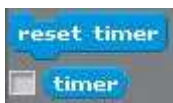
We all know that computers can be really fast at solving problems: feed a computer a problem that adds several million-digit numbers, and the answer comes out faster than a human has finished the first ten digits. (Well, unless you're one of those mental calculators (http://en.wikipedia.org/wiki/Mental_calculator).) In fact, that is one of the main motivations behind building computers: there are problems that humans can solve for small values, but once the values become large, it is better to ask a computer to do them instead, since a computer can crunch through calculations faster. Nonetheless, for many applications, speed is essential, and we can't trust a human to perform the calculations: GPS (http://en.wikipedia.org/wiki/Global_Positioning_System), for example, would be a very different (maybe non-existent) system if a human were continuously determining the position of a plane or of a car.

In these applications, the faster, the better, so if we're faced with two algorithms to perform the same problem, we want to be able to compare which one performs better. Since computers are more useful for large amounts of data, we also want to compare the performances for large inputs. One obvious way to do this is to run both algorithms on large inputs, time each of the algorithms from start to finish, and see which one performs better. (We can also see which algorithm takes up more *space* in memory, but for today's lab, we will only focus on time.)

Can you think of any other tasks where speed is essential? In other words, can you think of any task that probably would not exist (or that would be very tedious!) if a human were in charge of performing it?

Time is of the Es-sense

We have seen how BYOB can be used to wait for a certain time before reporting anything. BYOB also allows us to do the converse: to report how long a program takes to finish. In the `Sensing` menu, you will see a command called `reset timer` and a reporter called timer:



Activate (tick-mark) the timer reporter and you should see a timer ticking away above Alonzo. It's been ticking ever since you opened up BYOB, and counts in tenths of a second:

A small rectangular timer widget with a blue background and rounded corners. It has the word "timer" in white on the left and the number "134.0" in white on the right.

We are going to be performing timing experiments in the following sections, so this timer will prove useful. Add the following timing framework script into the space for BYOB scripts:



The script will reset the timer whenever the green flag is pressed. Alonzo will then say Hello! for 1 second, and soon after that, he will say the current time, less 1 second to account for how long he said Hello!. If we replace Hello! with a reporter, then Alonzo will say the answer of the reporter, and one second later, how long the reporter took to generate the answer: this is the timing information that we need. Save the script with the name `TimingFramework`.

Do You Have Time to Add?

Let us start our timing experiments with something simple: how long does it take for a computer to add 1 to (or *increment*) a number? Replace Hello! in the timing script with an addition operator from the `Operators` menu, and add 1 to 100000 (that's five zeros!). Run the script a few times (around four) to get an idea of the average approximate time (in seconds) it takes for the computer to increment 100000. Run the script repeatedly by double-clicking on it; do not place the script inside a repeat or a repeat until block, since we are interested in knowing the approximate time the script takes to run once.

Now, pause here and think: what's your gut feeling for how much longer the computer would take if we doubled 100000? Test your intuition: Get an average approximate time for how long it takes the computer to increment numbers that you progressively double: 200000, 400000, 800000, and 1600000. Remember that for each number, you need to run the timing script multiple times (around four) to get an idea of the average approximate time (you don't have to be precise). What do you observe?

Take another huge leap and find out how long it (approximately) takes for the computer to increment numbers that you progressively scale by 10: 160000000, 1600000000 (that's eight zeros), and maybe 16000000000 (that's nine zeros). What do you observe?

Constant-Time

You may have noticed that the computer takes approximately the same time to increment a number, even though the number was made progressively larger. Computer scientists (programmers and theorists) thus call incrementing a number a **constant-time** operation. In fact, any basic arithmetic operation (addition, subtraction, multiplication, division, and exponentiation) is considered to be a constant-time operation.

Notice that we call these operations *constant-time* operations, but we don't actually say how much time they take. Different computers will take different amounts of time to perform the same operation. To report the exact time of any algorithm, we would have to also report the physical configurations of the computer that the algorithm was run on. This gets very difficult and annoying very fast, especially with the almost infinite variety of computers available today. Instead, we focus on how the running time of an algorithm scales as we scale its inputs to larger and larger sizes, because this is a property of the algorithm itself, and is independent of the computer that it is run on.

Why did the computer take approximately the same amount of time to increment a number, even though that number was getting larger? Think about how you would add one to a number, back from your elementary school days; this is similar to how a computer does its arithmetic (ignoring technical details). The elementary school way of adding numbers goes digit by digit, and so the amount of time it takes for you to add two numbers depends on how many digits each number has. As we doubled the number we were incrementing, we didn't consistently add digits to it, and so the computer took approximately the same time. Even as we began scaling the number by ten, the computer (and you!) takes a relatively really small amount of time to account for the extra digit, so the total time remains approximately constant.

Constant-time operations are the Holy Grail of computer science algorithms, and unfortunately, most algorithms are not constant-time, as we will soon see.

All the Numbers, All the Time

For this section, we will be using this timing framework (`./code/timing.ypr`). It is very similar to the framework you constructed earlier, but it also has the stubs for a few blocks that we will be filling in. The first block that we will fill in is already on stage:

insert all numbers between 1 and 10 into 

This block should fill the input list with all of the numbers from start to end. If working correctly, the version used on stage should generate all of the numbers from 1 to max, where max is a number that we will change when running timing experiments, and should add them to the numbers list.

First, talk to your partner and discuss an algorithm that you can use to fill the input list with all of the numbers from start to end. Once you both have decided on an algorithm, complete the body of the block. Now, run the script with max set to 10. Run it a few times to get an idea of the average time the computer takes to generate this list (to the nearest tenth of a second); you don't need to be exact! Repeat the experiment with max set to 20, 40, 100, 200, and 1000. Do you see a pattern?

Linear Time

You may have noticed that if you increased max by a factor of two, then the computer took approximately twice as much time to fill the numbers list. Similarly, if you increased max by a factor of ten, then the computer took approximately ten times as much time; if you increased max by a factor of 100, then the computer ran approximately 100 times longer. In general, as we scale the size of the input by a certain amount, we also scale the running time by the same amount. We call these algorithms **linear-time** algorithms, because if we were to plot the runtime of one such algorithm against the size of its input, we would get a line.

Why is the algorithm a linear-time algorithm? When max was set to 20, we had to add 20 numbers to the numberslist; when max was set to 40, we had to add 40 numbers to the number list. In other words, as we scaled max, we also scaled how many numbers we had to add to the numbers list by the same amount. Linear-time algorithms are also much sought-after in computer science, and for many problems, they are the fastest algorithms you can find.

Algorithm Analysis

This lecture focuses on algorithm analysis, focusing on the following learning objectives:

- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Introduction

An algorithm's **correctness** refers to whether or not it contains errors. We will talk about testing procedures below.

An algorithm's **clarity** refers to how well it is expressed and described. Can you understand what it does and how it does it? We will begin using **comments** to document our code.

An algorithm's **efficiency** refers generally to how efficiently it uses two key resources, time and space – i.e., the *processor or central processing unit* (CPU), and the computer's *main memory*, often called *random access memory* (RAM).

The faster the algorithm – the quicker it finishes its task – the more efficient it is with respect to time.

The less memory it uses, the more efficient it is with respect to space.

Sorting and Searching Algorithms

In discussing this topic we will talk about *sorting* and *searching* algorithms. A **sort** algorithm arranges data in some order – e.g., numeric or alphabetic order. A **search** algorithm looks up some target data – e.g., a name in the phone book. You can think of the data as being contained in a **list**.

Efficiency and Input Size

Efficiency issues only come into play when there are large amounts of **data**. An algorithm's *input size* is often abbreviated with N . As N grows, the differences between efficient and inefficient algorithms become clear.

For example, this table shows the differences in running time in seconds for the same sort algorithm on different input sizes.

List Size, N	Older Computer	Newer Computer
125	12.5	2.8
250	49.3	11.0
500	195.8	43.4
1000	780.3	172.9
2000	3114.9	690.5

Computer scientists like to *abstract away* the efficiency differences that are caused by different hardware and software – e.g., faster vs. slower computer.

Here are some old data obtained on different machines when sorting 2000 integers using the same algorithm:

Type of computer	Time
Home PC	51.915

Type of computer	Time
Desktop PC	11.508
Minicomputer	2.382
Mainframe	0.431
Supercomputer	0.087

Empirical Analysis: The data in both of these tables were gathered **empirically** or **experimentally** – i.e., by running and timing the actual program on different machines and inputs. Abstract Analysis

We can also analyze algorithm efficiency more abstractly, by comparing an algorithm to well known mathematical functions.

Consider the following graph with four functions : the logarithmic, linear, quadratic, and exponential functions:

- Logarithm function: $y = \log_2 x$
- Linear function: $y = 5x$
- Quadratic function: $y = x^2$
- Exponential function: $y = 2^x$

Graph

As you can see, as X increases, the running time increases. But look at the differences. Look at how slowly running time increases for the logarithmic function compared to the exponential function.

In terms of these four types of functions, we arrange the them in the following order in terms of how fast they increase as the size of, x , increases:

logarithmic	linear	quadratic	exponential
$\log_2 x$	x	x^2	2^x

Algorithm Efficiency

Computer scientists like to characterize algorithm efficiency in terms of **how fast the running time will increase as the size of the inputs, n , increases**.

To see the dramatic differences, consider some numbers for these for types of functions:

Inputs	Logarithmic	Linear	Quadratic	Exponential
n	\log_2	n	n^2	2^n
8	3	8	64	256
16	4	16	256	65,536
32	5	32	1,024	4,296,967,296
64	6	64	4,096	1.84×10^{19}
128	7	128	16,384	3.40×10^{38}
256	8	256	65,536	1.15×10^{77}
512	9	512	262,144	1.34×10^{154}

So, for 512 inputs, a linear algorithm could perform some task in 512 seconds. Whereas an exponential algorithm would take 1.34×10^{154} seconds.

Algorithms and Problems

Computer scientists like to analyze problems by asking, **what type of algorithm do we need to solve this problem?** Or, what's the fastest type of algorithm that can solve this problem?

A Linear Search Algorithm

For example, to find the number 100 in the following lists of numbers, we would have to go through each number in the list, from left to right.

List 1 (16 numbers): (15 20 4 2 1 17 19 25 100 65 78 19 20 15 0 72)

List 2 (16 numbers): (15 20 4 2 1 17 19 25 65 78 19 20 15 0 72 75)

We can use a linear search algorithm:

```
# Search for X in List L and return its Index, indx, in L
# Or return -1 if X is not in the List

To Search for X in List L, DO:
    Set global indx to 1
    For each number, num, in the list, L:
        If num = X:
            Return indx      Set indx to indx + 1
    Return -1                # The number is not in the list so return -1
```

TODO: Hand trace this algorithm on List 1 and List 2 and count how many times you have to go through the loop.

QUESTION: Suppose there 512 numbers in our list. In the **worst case** – i.e., when X is not in the list – how many times would we go through the while loop?

This algorithm (and the problem it solves) are said to be **linear** because in the **worst case** it takes N iterations of the loop to find something in a list of N items.

A Logarithmic Search Algorithm

If our list were sorted – e.g., like the telephone book – then it wouldn't make sense to do a linear search – i.e., start at page 1.

List 1 (16 numbers): (1 2 5 6 9 12 15 16 32 64 100 128 256 299 512 568)

List 2 (16 numbers): (1 2 5 6 9 12 15 16 32 64 99 128 256 299 512 568)

Instead we could guess the approximate location of the number in the list and look there. In the most general case, we could guess that the X was the middle value in the list. If X was greater (or smaller) than the middle value, we would search the top (bottom) half of the list.

This is known as the binary search algorithm. For example, to guess a secret number between 1 and N:

```
Guess the middle value in the range 1..N
If the guess is too high,
    cut off the top of the range.
If the guess is too low,
    cut off the bottom of the range.
Repeat until there's only 1 number left.
```

QUESTION: How many guesses to find the secret number between 1 and 100? Between 1 and 500?

You should be able to see that as N grows larger, the binary search will grow very slowly, like a logarithmic function.

A Quadratic Sort Algorithm

Let's look at an example of an algorithm that performs like a quadratic function. Sort a list of N numbers from low to high using bubble sort.

```
List 1 (5 numbers): (10 5 4 4 1)
```

NOTE: Bubble sort is probably the world's worst sorting algorithm. And there are many sorting algorithms that are much more efficient. But here is bubble sort:

```
# Sort the N numbers in List L into ascending order

For a list of N items, repeat N-1 times.      # N-1 steps
  For each adjacent pair of items                # N-1 pairs
    If the pair is out of order, swap the numbers
```

This algorithm contains a **nested loop**. The outer loop repeats $N-1$ times. And on each repetition the inner loop does $N-1$ comparisons of adjacent numbers. So that is $N^2 - 2N + 1$ comparisons. That's quadratic.

Clock analogy: Think of the relationship between the second hand and the minute hand on a clock. The second hand ticks 60 times each minute. To time 1 hour, the minute hand would tick 60 times, which is 3600 seconds.

TODO: Hand trace this algorithm on List 1 and count how many times you have perform the IF statement?

Question: Suppose there are 512 numbers in the list and it took 1 second to compare and swap a pair of numbers. Approximately how many seconds would it take to sort the list?

Efficient Sorting

Bubble sort is an example of an in-place sorting algorithm. It doesn't require much more memory than the list itself. The most efficient in-place sorting algorithms can sort N numbers in time that is proportional to an $n \log n$ function. For 512 numbers that would require time proportional to $512 \times 8 = 8,192$ seconds as opposed to $512 \times 512 = 262,144$ seconds.

A Linear Sorting Algorithm

What if we didn't care about conserving space? How fast could we sort N numbers?

```
# Sort the list of N numbers in List L in ascending order. Suppose
# the list contained numbers whose values ranged between 1 and M.
# For example the numbers might be between 1 and 1000.

(1) Make a list, LL, with M "buckets", each containg an empty list.
(2) For each of the N numbers in list L, put it in its proper bucket.
(3) Traverse through the list, LL, and empty the buckets back into the list, L.
```

For example, for our original list, (10 5 4 4 1) and assuming that the numbers range between 1 and 10:

```
(0)  L = ( 10 5 4 4 1)
(1)  LL = ( () () () () () () () () () )
(2)  LL = ( (1) () () (4 4) (5) () () () (10) )
(3)  L = ( 1 4 4 5 10 )
```

Questions

- **Space efficiency:** In terms of space, this algorithm requires space for the N numbers in L and the M buckets, so $N + M$. How much more space is this? Linearly more? Quadratically more?
- **Time efficiency:** In terms of our N inputs, this algorithm grows linearly as the size of N grows.

This shows that for a given problem, different algorithms can have different efficiencies.

An Exponential Algorithm

The Traveling Salesman Problem (<http://turing.cs.trincoll.edu/~ram/cpsc110/inclass/alg-analysis/>) is an example of a problem that can only be solved using an *exponential algorithm*. Given N cities, find the shortest path (least costly path) through all N cities.

The only known way to solve this problem is by **brute force**. List all possible paths through the cities. For 3 cities we can easily do this:

```
A B C
A C B
B A C
B C A
C B A
C A B
```

In general there are $N!$ ways to arrange N cities. So there are $3 \times 2 \times 1 = 6$ ways to arrange 3 cities. And in general $N!$ is much greater than $2N$:

N	N!	2N
3	6	8
4	24	16
5	120	32
6	720	64

Problems that can only be solved by *exponential algorithms* are known as **intractable problems**. There is no general, optimal solution for the traveling salesman problem that runs in a reasonable amount of time.

There are **heuristic** solutions. A *heuristic* is a rule-of-thumb that gives a pretty good (but not always optimal) solution. For the traveling salesman problem we can use the nearest-neighbor heuristic – i.e., select the nearest neighboring city as the next city. Computability

Theoretically speaking, not practically, are there problems that cannot, in principle, be computed? We're not talking here about *practical* limits to computation.

The answer is "No". In 1936 British Mathematician Alan Turing (http://en.wikipedia.org/wiki/Alan_Turing) proved that there is a set of problems that cannot be solved by any algorithm or computational procedure.

Example: The Halting Problem (http://en.wikipedia.org/wiki/Halting_problem). Given a description of a computer

program, determine whether the program halts or continues running forever. In other words, given a program and an set of inputs, decide whether the program will eventually halt given that set of inputs.

Turing proved that there is no general algorithm that can solve the halting problem for any program and any inputs.

We won't go into the details here, but it's important that you know that computers do have this limitation.

Exercises

Do the following exercises and add your answers to your Portfolio page for today's assignment.

For each of the following problems, decide whether it can be solved by a *linear* algorithm and explain briefly why or why not.

- Compute the sum of a list containing N random integers.
- Determining if duplicate numbers occur in a list of N random integers.

For each of the following problems, decide whether it can be solved by a *logarithmic algorithm* and explain briefly why or why not.

- Guessing a number between 1 and 100 if your guesses are characterized as "too high" or "too low".
- Guessing a number between 1 and 100 if your guesses are characterized as "wrong" or "right".

For each of the following problems, decide whether or not it is *intractable* and explain briefly why or why not.

- Computing the sum of all the numbers in a square matrix of dimension $N \times N$.
 - Given n integers does there exists a subset of them that sum exactly to B ? For example, suppose the integers are $\{4, 5, 8, 13, 15, 24, 33\}$. If $B = 36$ then the answer is yes (and 4, 8, 24 is a solution). If $B = 14$ the answer is no.
-

SEARCHING & SORTING

Computer Science Principles

LEARNING OBJECTIVES

- 1: The student can use computing tools and techniques to create artifacts
- 3: The student can use computing tools and techniques for creative expression.
- 16: The student can express an algorithm in a language.
- 21: The student can evaluate a program for correctness.
- 22: The student can develop a correct program.
- 23: The student can employ appropriate mathematical and logical concepts in programming.

MORE ALGORITHMS

THE STATE GUESSING GAME

- The Rules
 - I will choose a state
 - I will answer questions in such a way that my answers to your questions are true/false (yes/no)
 - You may have up to 5 guesses to guess the state.



Map from www.infoplease.com

SEARCHES

- The strategy of trying to divide choices in half, then again and again until the answer is found is called a Binary Search.
 - Remember NoseGuy and the Guessing Game?
- We are going to look at strategies (or Algorithms) for searching for the “correct” answer.

SEARCHING VS. SORTING

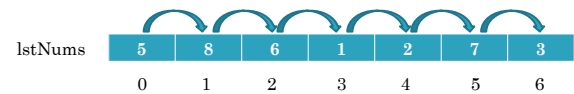
- When you search a list, it is to determine if a specified element is present.
 - There are two primary searching algorithms
 1. Linear Search
 2. Binary Search
- Sorting is done to order the values in the list based upon some key value.
 - There are three primary sorting algorithms
 1. Selection Sort
 2. Insertion Sort
 3. Merge Sort

SEARCHING ALGORITHMS

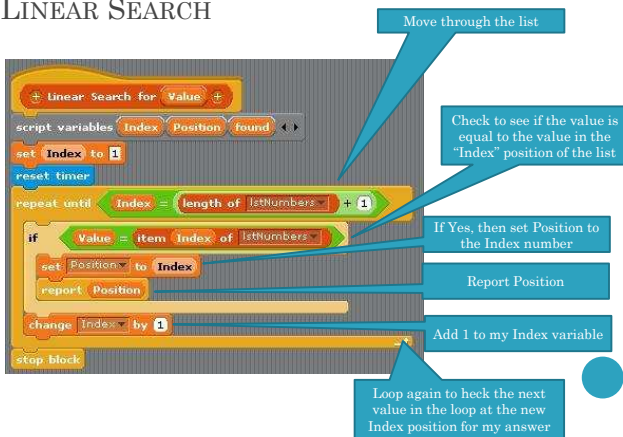
Where is that element?

LINEAR SEARCH

- A linear search algorithm searches the list in a sequential manner.
- The algorithm moves through the list, comparing the key value with the values of the elements. If it does not find the key value, it simply moves to the next element.



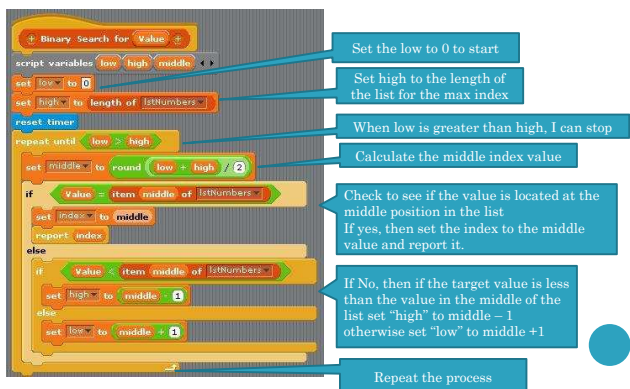
LINEAR SEARCH



BINARY SEARCH

- The binary search algorithm is more efficient than the linear search algorithm.
- Binary search requires that the array be sorted first.
- The algorithm splits the array and checks the middle value.
 - If it is not found it compares the values.
 - If the search value is higher than the middle value, the algorithm moves to the upper half (now a subarray). (Lower – it moves to the lower half.
 - It splits the subarray in half, checks the middle for a match.
 - If not found, it checks to see if it is higher/lower and moves to appropriate subarray.
 - This continues until it has no more values or finds a match.

BINARY SEARCH



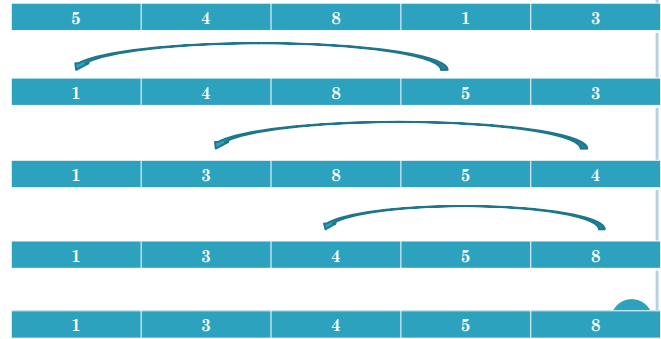
SORTING ALGORITHMS

C# Programming

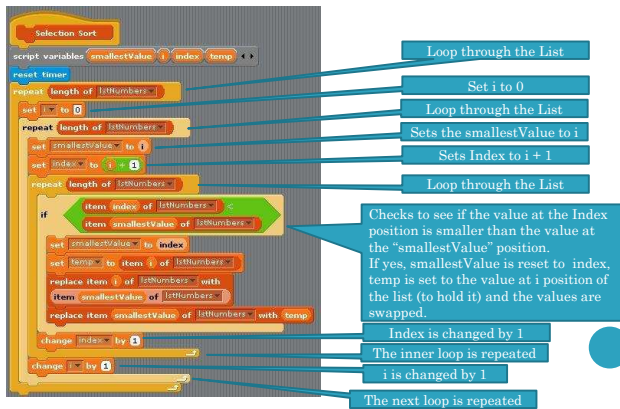
SELECTION SORT

- This is a simple sorting algorithm.
- It moves through the array looking for the lowest value, then moves it to the front.
- The second pass through the array, it looks for the second lowest and moves it to the second position.
- It keeps passing through the array until the last iteration.

SELECTION SORT



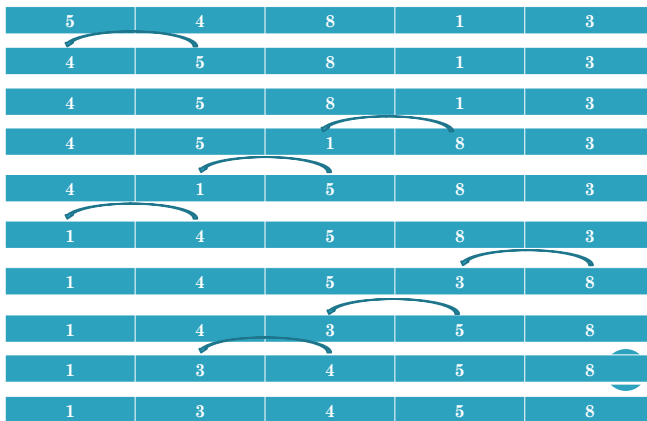
SELECTION SORT



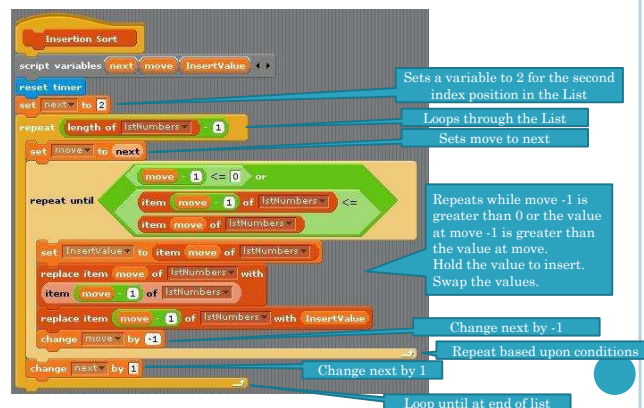
INSERTION SORT

- Another simple sorting algorithm.
- 1st Iteration – Compares element 1 & 2 – Swaps if element 1 > element 2.
- 2nd Iteration – Looks at element 3 – Inserts it in position given element 1 & 2.
- It keeps comparing and inserting until the end.

INSERTION SORT



INSERTION SORT



MERGE SORT

- The merge sort algorithm sorts by splitting the array into two subgroups, sorting the subgroups, then merging them back together sorted.

56	18	65	17	35	29	44
56	18	65	17	35	29	44
56	18	65	17	35	29	44
56	18	65	17	35	29	44
18	56	17	65	29	35	44
17	18	56	65	29	35	44
17	18	29	35	44	56	65

MEASURING EFFICIENCY

- How efficiency an algorithm is can be measured using Big-O notation. (Also called Landau's symbol)
- It tells you how fast a function grows or declines.
- Big-O notation measures the worst-case runtime for an algorithm.

EFFICIENCY OF SEARCHING

- Linear Search
 - Worst Case time $O(N)$ - It goes through entire list
- Binary Search
 - Worst Case time $O(\log N)$ – The number of times N can be divided in half before there is nothing left
 - This is better than the linear search.
- Selection Sort & Insertion Sort
 - Worst Case time $O(N^2)$
- Merge Sort
 - Worst Case time $O(N \log N)$
 - This is a little better than the selection and insertion sorts.

Activity - Searching

We're going to step away from the computer for this activity. First things first: divide up into groups. Let's examine a common computer science problem: finding a number in a list of numbers. We can assume that the user will provide a specific number to look for, and we need to develop an algorithm that will find what location the number they're searching for is located in (or if it's absent). A computer can only check one position in a list at a time, and checking positions takes time. Let's say we're searching through a list where the numbers are random (within a range) and unique (the same number won't be in there twice). As you might imagine, there are several different ways that you can find a particular number in this type of list, but we know that many algorithms have advantages over others. We'll use a set of numbered cups in this activity to represent different pieces of data. Let's call the number of cups N .

Our first goal is to find a specific number among a bunch of randomly chosen cups. Using your set of cups as a testbed, work with your group to **develop an algorithm (remember: set of steps) to do this**. Your algorithm should work for any set of random numbers. **Discuss the following ideas with your group to help your thought process:**

- What strategies exist for finding a particular number in the fewest number of location checks?
 - How many guesses would it take on average to find a number?
 - How many guesses would it take to determine that a number didn't exist in the list at all?
 - How much additional work will it be to find a number if N were to increase?
-

A (Non-Video) Game

Let's play a number-guessing game (you have already seen the Scratch version in lab 4.01).

Between your partner and yourself, decide who will be person G, the guesser: the other person (person P, for picker) picks a number between 1 and 50. Person G asks person P questions to determine the chosen number; the only answers that person P can give are "yes" or "no". The guesser should try two different strategies to playing this game:

- Guessing in sequence: "Is the number 1?", "Is the number 2?", "Is the number 3?", and so on.
- Guessing halfway: You know the number must be between 1 and 50, so you start off by asking "Is the number greater than 25?". If the answer is "yes", you instantly know that it must be between 26 and 100, so you ask "Is the number greater than 50?"; otherwise, the number must be between 1 and 25, so you ask "Is the number greater than 12?" Keep asking questions involving the halfway point.

Play several (around 3) games: person P should pick a wide variety of numbers, while person G should use each strategy alternately. As you play each game, think about which strategy involves fewer questions. Which strategy would be better if, say, the guessing game involved numbers from 1 to 1000? From 1 to a million? Discuss why one strategy better than the other?

[Curriculum \(/bjc-course/curriculum\)](#) / [Unit 6 \(/bjc-course/curriculum/06-searching-sorting\)](#) /[Lab 2 \(/bjc-course/curriculum/06-searching-sorting/labs/03-update-guessing-game\)](#) /

Lab: Add Binary Search to the Guessing Game

Using the starting code provided in lab 5.01, upload your expanded guessing game.

Your Noseguy sprite should utilize binary search in his Guess function to guess the number in as few guesses as possible.

The best route for this would be to set a maximum and minimum value, and adjust them as Noseguy is told “higher” or “lower” by BYOB. Think how you would play as a human. You should have Noseguy able to guess the number in 4 guesses or less.
