**Binary Search**

* It is a **recursive** technique that splits a problem in half.
* ***Binary search*** is a better alternative to a sequential search.

**Relating To a Dictionary**

* Binary search, in theory, is similar to the way in which you would actually use a dictionary.
* You open the dictionary—maybe to a point near its middle—and by glancing at the page, determine which “half” of the dictionary contains the desired word.
* Parts of this solution are intentionally vague:
  + How do you scan a single page?
  + How do you find the middle of the dictionary?
  + Once the middle is found, how do you determine which half contains the word?
* The answers to these questions are not difficult, but they would only obscure the solution strategy right now.
* The following pseudocode is a first attempt to formalize the binary search process:

Text

Description automatically generated

* The previous search strategy reduces the problem of searching the entire dictionary for a word to a problem of only searching half of the dictionary for the word, as Figure 2-1 illustrates.

Diagram

Description automatically generated

**Notice two important points.**

1. First, once you have divided the dictionary in half, you can use the exact same strategy that you used to search the original dictionary.
   1. This is the heart of recursion. A function calls itself, passing in smaller parameters to shrink the problem.
2. Second, note that there is a special case that is different from all the other cases:
   1. After you have divided the dictionary so many times that you are left with only a single page, the halving ceases.
   2. At this point, the problem is sufficiently small that you can solve it directly by scanning the single page that remains for the word.
   3. This special case is called the **base case** (or **basis** or **degenerate case**).

**Divide and Conquer**

* Binary search is an example of a strategy called ***divide and conquer***.
* You solve the dictionary search problem by first ***dividing***the dictionary into two halves and then ***conquering***the appropriate half.
* You solve the smaller problem by using the same divide-and-conquer strategy.
* The dividing continues until you reach the base case.

search(aDictionary: Dictionary, word: string)

**if** (aDictionary *is one page in size*)

*Scan the page for* word

**else**

{

*Open* aDictionary *to a point near the middle*

*Determine which half of* aDictionary *contains* word

**if** (word *is in the first half of* aDictionary)

search( *first half of aDictionary*, word)

**else**

search(*second half of* aDictionary, word)

}

Writing the solution as a function allows several important observations:

1. **A recursive function calls itself**
   1. One of the actions of the function is to call itself; that is, the function search() calls the function search().
   2. This action is what makes the solution recursive.
   3. The solution strategy is to check the base case, determine which half to follow recursively (splitting the problem in half), and apply the same strategy to the appropriate half.
      1. The solution strategy for the dictionary problem is to split aDictionary in half, determine which half contains word, and apply the same strategy to the appropriate half.
2. **Each recursive call simplifies the problem, while the objective stays the same**
   1. The function solves the search problem by solving another search problem that is identical in nature but smaller in size.
   2. Each call to the function search() made from within the function search() passes a dictionary that is one-half the size of the previous dictionary.
   3. That is, at each successive call to search(aDictionary, word), the size of aDictionary is cut in half.
3. **There is a base case (check) that you handle differently from other checks** 
   1. When aDictionary contains only a single page, you use another approach: You scan the page directly.
   2. Searching a one-page dictionary is the base case of the search problem.
   3. When you reach the base case, the recursive calls stop and you solve the problem directly.
4. **You are guaranteed to reach the base case as long as you diminish the problem on each function call**
   1. The manner in which the size of the problem diminishes ensures that you will eventually reach the base case.