An array has the benefits of:

* O(\_\_1\_\_\_) retrieval and storage

But, when we want to use an array for searching...

* when an array is unsorted, we must perform a \_\_linear O(n)\_\_\_ search

To perform efficient search of a data structure...

* we wish to apply \_\_binary\_\_\_\_ search, which has O(\_\_log n\_\_\_) complexity
* in order to perform this efficient search, the data must be \_\_\_\_sorted\_\_\_\_\_\_\_\_\_ (sorted or unsorted?)

Unfortunately, an array does not automatically maintain a sorted order of its elements. We could implement our own code to handle this, where, every time we add a new element to the array...

* we perform a binary search to find the index the new value should be inserted at, but...
  + inserting into an array is O(\_\_\_n\_\_\_\_\_)

We know of another structure that has faster insertion...

* A \_linked\_\_ list, which provides O(\_\_\_\_\_\_1\_\_\_\_\_\_\_) insertion

We could maintain the sorted order of elements in this list, but for every insertion...

* we must traverse the list linearly to find the right position to insert at, which is O(\_\_\_\_\_\_\_\_\_n\_\_\_\_\_\_\_\_\_\_)

In other words, in order to search efficiently, we need a sorted structure. And we need a structure that can maintain the sorted order of its elements efficiently.

A Binary Search Tree is a non-linear data structure that provides a solution to the constraints above. It provides:

* O(\_\_\_\_log n\_\_\_\_\_\_) insertion of a new node
* O(\_\_\_\_\_log n\_\_\_\_\_) search, access, and deletion of a node

**It provides efficient search and insertion *while maintaining sorted order*, at the cost of performing not-quite-as-well with other operations.** However...

* Compared to an *array*, a BST provides \_\_\_\_\_worse\_\_\_\_\_\_\_\_\_(better or worse?) access to any one element in the structure.
  + So, if my programming situation requires me to always traverse my data structure linearly, rarely insert/delete, and quickly access each element, I might reach for an \_\_\_array\_\_\_\_ instead of a BST.
* Compared to a *linked list*, a BST provides \_\_worse\_\_\_\_\_\_\_\_\_\_\_ (better or worse?) insertion into the structure.
  + So, if my programming situation doesn't expect search or maintaining sorted order, or if my situation involves lots of insertions/deletions, I might reach for a \_linked list\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ instead of a BST.

In summary, the benefits of a BST are \_\_\_organizing large data sets\_\_\_\_\_\_\_\_\_\_\_\_\_, while slightly sacrificing performance for \_\_\_complex\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ operations. It maintains the sorted order of its elements, at a better cost than an array or linked list can, in order to provide efficient \_\_searching algorithms on large data sets\_\_\_\_\_\_\_\_\_\_\_\_\_\_.