

## Binary Search

The Binary Searching technique is good to use when the data in the list is ordered. The data must be in increasing or decreasing order. To do a binary search, first look at the middle element of the entire list. If this item is not your key, then determine if your key will be to the left or right of this middle position. Once you determine which half it is in, you can ignore half the elements in the list, and constrict your search to the other half. In this half you have left pick the middle element of this part, and repeat the same process as before. Continue breaking the parts to look at into half until you either find the key, or there are no parts to look at. For example:

```
struct DataType
{
    int info;
    int key;
};

typedef apvector<DataType> List;

int BinarySearch(const List & x, int key)
// pre: x is a list containing ordered data, from least to greatest
// post: return the index of where key is found
//       return -1 if key is not found
{
    int left = 0;
    int right = x.length() - 1;
    int middle = (left + right) / 2;
    while (x[middle] != key && left <= right)
    {
        if (key < x[middle])
            right = middle - 1;
        else
            left = middle + 1;
        middle = (left + right) / 2;
    }
    if (key == x[middle])
        return middle;
    else
        return -1;
}
```

The least number of searches this algorithm would take to find a key is 1. That is it would find the key in the middle location of the list.

The most number of searches this algorithm would take to find the key is  $\log_2(n+1)$ . The variable  $n$  is used to indicate the number of elements in the list, in this case it's value is  $x.length()$ .

The average number of searches this algorithm would take to find the key is  $\log_2(n) - 1$ .