# Binary Search

A binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

A binary search halves the number of items to check with each iteration, so locating an item (or determining its absence) takes logarithmic time. A binary search is a divide and conquer search algorithm.

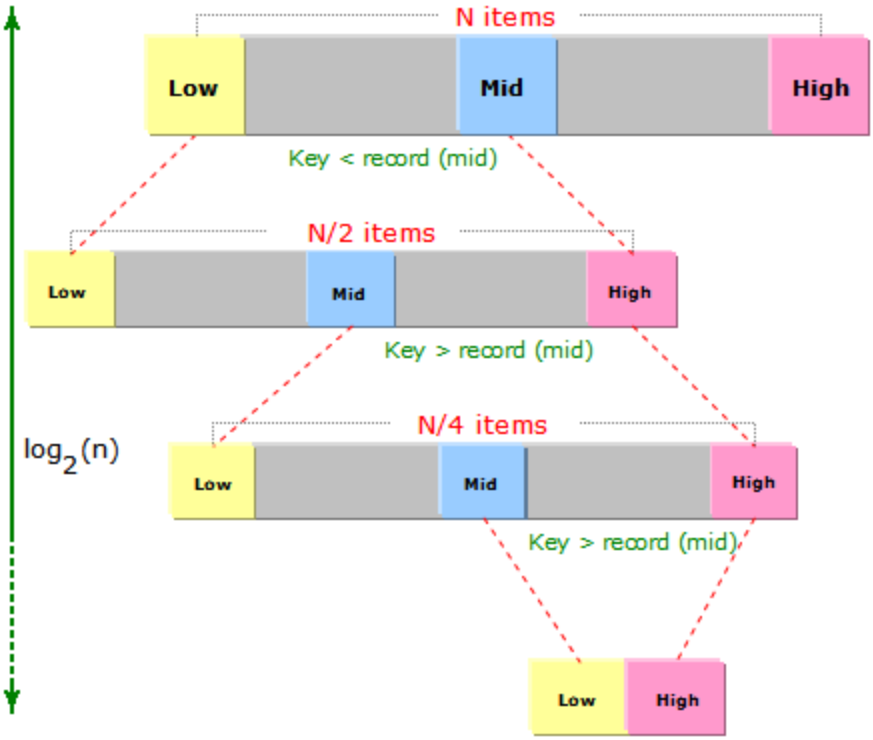
Example: The list to be searched: L = 1 3 4 6 8 9 11. The value to be found: X = 4.

```
Compare X to 6. X is smaller. Repeat with L = 1 3 4.
Compare X to 3. X is bigger. Repeat with L = 4.
Compare X to 4. They are equal. We're done, we found X.
```

## Recursive Algorithm

```
int binary_search(int A[], int key, int imin, int imax)
{
  if (imax < imin) // test if array is empty
  // set is empty, so return value showing not found
  return KEY_NOT_FOUND;
 else
  { int imid = midpoint(imin, imax);//calculate midpoint

   if (A[imid] > key)     // key is in lower subset
        return binary_search(A, key, imin, imid - 1);
   else if (A[imid] < key) // key is in upper subset
        return binary_search(A, key, imid + 1, imax);
   else                   // key has been found
       return imid;
  }
}
```




## Equivalent Iterative Program

```
int binary_search(int A[], int key, int imin, int imax)
{
 // continue searching while [imin, imax] is not empty
 while (imax >= imin)
  {
   // calculate the midpoint for roughly equal partition
   int imid = midpoint(imin, imax);
   if(A[imid] == key)      // key found at index imid
    return imid;    // determine which subarray to search
   else if (A[imid] < key)
   // change min index to search upper subarray
    imin = imid + 1;
   else         // change max index to search lower subarray
    imax = imid - 1;
  }
 // key was not found
 return KEY_NOT_FOUND;
}
```

## Time Complexity

| | |
|---|---|
| Worst case performance | $O(log\ n)$ |
| Best case performance | $O(1)$ |
| Average case performance | $O(log\ n)$ |
| Worst case space complexity | $O(1)$ |