

## ICS4U – Linear Search

In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection or end of the array.

## Algorithm

```
Linear Search ( Array A, Value x)
```

```
Step 1: Set i to 1
```

```
Step 2: if i > n then go to step 7
```

```
Step 3: if A[i] = x then go to step 6
```

```
Step 4: Set i to i + 1
```

```
Step 5: Go to Step 2
```

```
Step 6: Print Element x Found at index i and go to step 8
```

```
Step 7: Print element not found
```

```
Step 8: Exit
```

## ICS4U – Binary Search

Binary search is a fast search algorithm. This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection must be sorted.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array until the size of the sub-array reduces to zero.

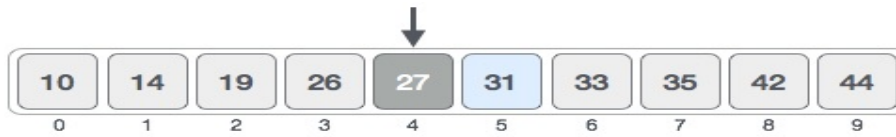
For a binary search to work, it is mandatory for the target array to be sorted. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.



First, determine half of the array by using the formula

```
mid = low + (high - low) / 2
```

Here it is,  $0 + (9 - 0) / 2 = 4$  (integer value of 4.5). So, 4 is the mid of the array.



Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



We change our low to mid + 1 and find the new mid value again.

```
low = mid + 1  
mid = low + (high - low) / 2
```

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



The value stored at location 7 is not a match, rather it is less than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very small numbers.