# Chapter 10

# Searching

## 10.1 Sequential Search

A simple algorithm that search for a specific item inside a list. It operates looping on each element $O(n)$ until a match occurs or the end is reached.

```
1) algorithm SequentialSearch($list$, $item$)
2)     Pre: $list \neq \emptyset$
3)     Post: return $index$ of item if found, otherwise $-1$
4)     $index \leftarrow 0$
5)     while $index < list$.Count and $list[index] \neq item$
6)         $index \leftarrow index + 1$
7)     end while
8)     if $index < list$.Count and $list[index] = item$
9)         return $index$
10)    end if
11)    return $-1$
12) end SequentialSearch
```

## 10.2 Probability Search

Probability search is a statistical sequential searching algorithm. In addition to searching for an item, it takes into account its frequency by swapping it with it's predecessor in the list. The algorithm complexity still remains at $O(n)$ but in a non-uniform items search the more frequent items are in the first positions, reducing list scanning time.

Figure 10.1 shows the resulting state of a list after searching for two items, notice how the searched items have had their search probability increased after each search operation respectively.

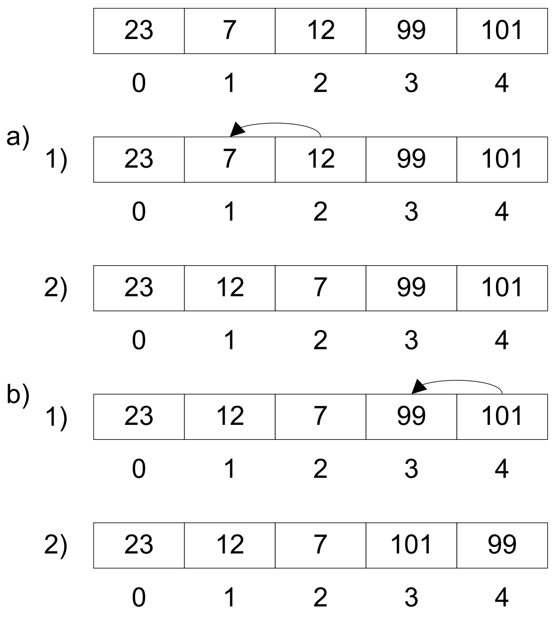

Figure 10.1: a) Search(12), b) Search(101)

```
1) algorithm ProbabilitySearch(list, item)
2)     Pre: list ≠ ∅
3)     Post: a boolean indicating where the item is found or not;
             in the former case swap founded item with its predecessor
4)     index ← 0
5)     while index < list.Count and list[index] ≠ item
6)         index ← index + 1
7)     end while
8)     if index ≥ list.Count or list[index] ≠ item
9)         return false
10)    end if
11)    if index > 0
12)        Swap(list[index], list[index − 1])
13)    end if
14)    return true
15) end ProbabilitySearch
```

## 10.3 Summary

In this chapter we have presented a few novel searching algorithms. We have presented more efficient searching algorithms earlier on, like for instance the logarithmic searching algorithm that AVL and BST tree's use (defined in §3.2). We decided not to cover a searching algorithm known as binary chop (another name for binary search, binary chop usually refers to its array counterpart) as the reader has already seen such an algorithm in §3.

Searching algorithms and their efficiency largely depends on the underlying data structure being used to store the data. For instance it is quicker to determine whether an item is in a hash table than it is an array, similarly it is quicker to search a BST than it is a linked list. If you are going to search for data fairly often then we strongly advise that you sit down and research the data structures available to you. In most cases using a list or any other primarily linear data structure is down to lack of knowledge. Model your data and then research the data structures that best fit your scenario.