The Sequential Search (Linear Search)

Dave Clausen

La Cañada High School

The Sequential Search Description

The Sequential (or Linear) Search examines the first element in the list and then examines each "sequential" element in the list (in the order that they appear) until a match is found. This match could be a desired word that you are searching for, or the minimum number in the list.

The Sequential Search Variations

Variations on this include: searching a **sorted** list for the first occurrence of a data value, searching a **sorted** list for all occurrences of a data value (or counting how many matches occur: inventory), or searching an **unsorted** list for the first occurrence or every occurrence of a data value.

You may indicate that a match has been found, the number of matches that have been found, or the indices where all the matches have been found.

A Sequential Search Algorithm

- Set index to 0 (zero)
- Set found to false
- while index < length and not found do
  - if list[index] is equal to target then
    - set found to be true
  - else
    - Increment the index by 1 (one)
- if found then
  - return index
- else
  - return -1 (negative one)

A Sequential Search C + +

```
int Sequential_Search(int target, apvector <int> &list, int length)
{
    int index = 0;
    bool found = false;
    while((index < length) && ! found)
        if (list[index] = = target)
            found = true;
        else
            ++index;
    if (found)
        return index;
    else
        return -1
}
```

Revised Sequential Search Algorithm

Set index to 0
Set found to false
While index < length and not found do
  If v[index] is equal to target then
    Return index
  Else
    Increment the index by 1
Return -1

Revised Sequential Search C + +

```
int search(int target, const apvector<int> &v)
{
   int index = 0;
   while (index < v.length())
      if (v[index] = = target)
         return index;
      else
         ++index;
   return -1;
}
```

The Sequential Search C + + Variation #1

If the list is sorted, we can improve this code by adding the following extended if statement:

```
if (list[index] = = target)
        found = true:
else if (list[index] > target)     //target is not in list
            index = length;
else
        ++index;
```

The Sequential Search C + + Variation #2

Whether the list is sorted or not, we can return the number of occurrences of the target in the list:

```
int Occurrences_Of (int target, const apvector <int> &list)
{
    int count = 0;
    for(int index = 0; index < list.length(); ++index)
        if (list[index] = = target)
            + + count;
    return count;
}
```

The Sequential Search C + + Variation #3

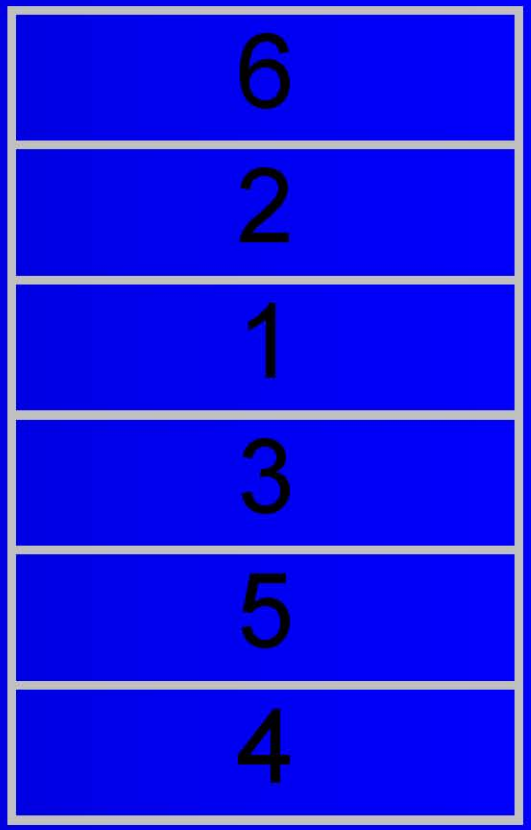
Whether the list is sorted or not, we can return the indices of occurrences of the target in the list:

void Indices_Of (int target, const apvector<int> &list)

```
{
for(int index = 0; index < list.length(); ++index)
        if (list[index] = = target)
            cout<< target << " located at index #  "
                <<index<<endl;
}
```

A Sequential Search Example

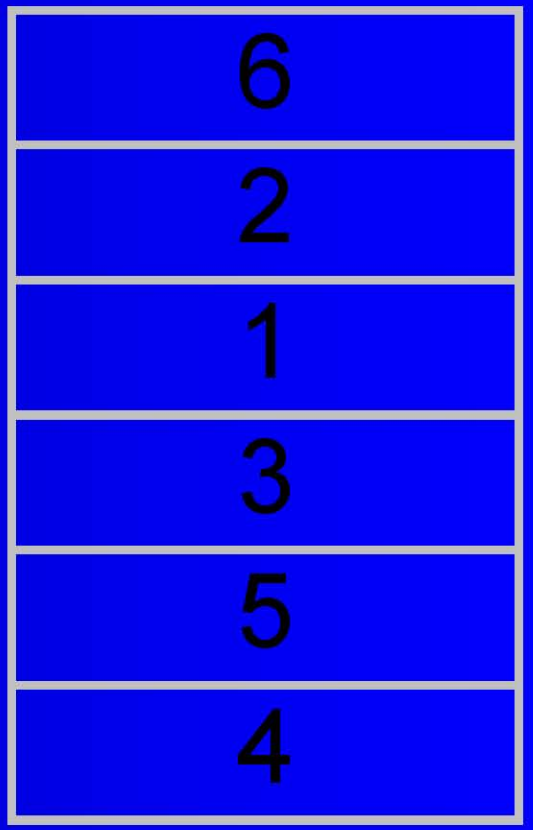
Target ?



We start by searching for the target at the first element in the List and then proceed to examine each element in the order in which they appear.

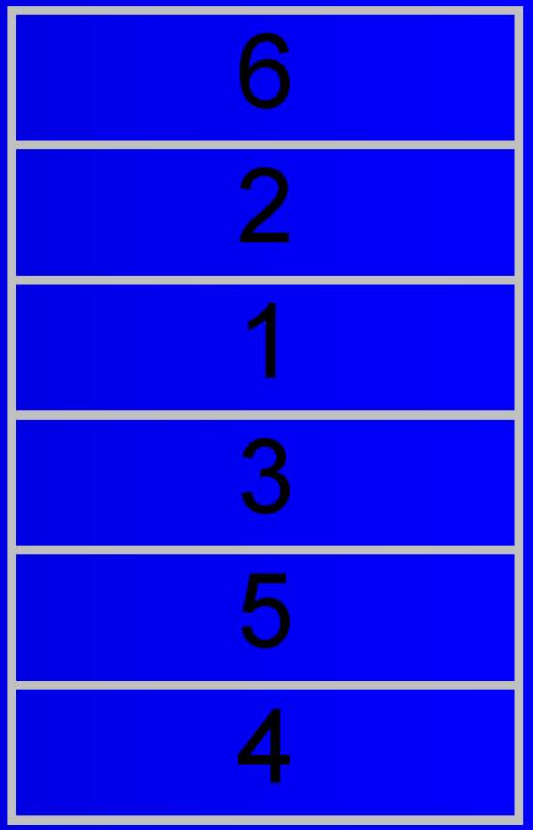
A Sequential Search Example

Target ?



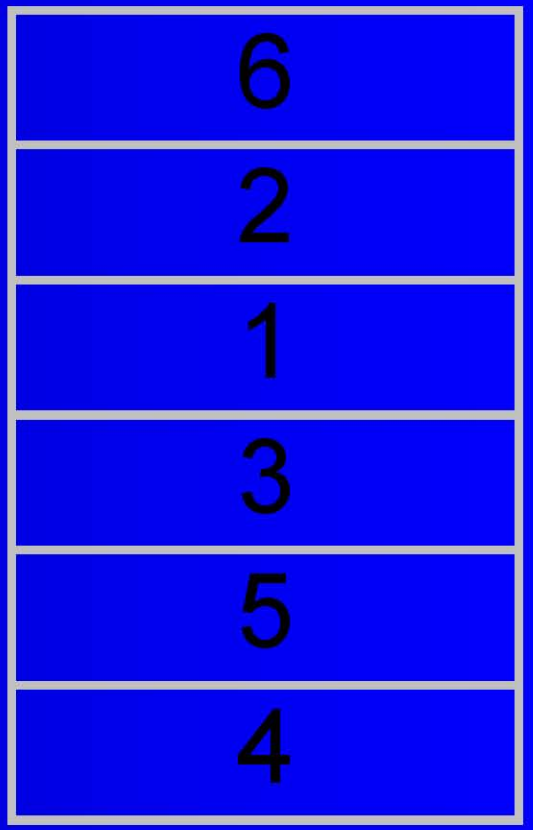
A Sequential Search Example

Target ?



A Sequential Search Example

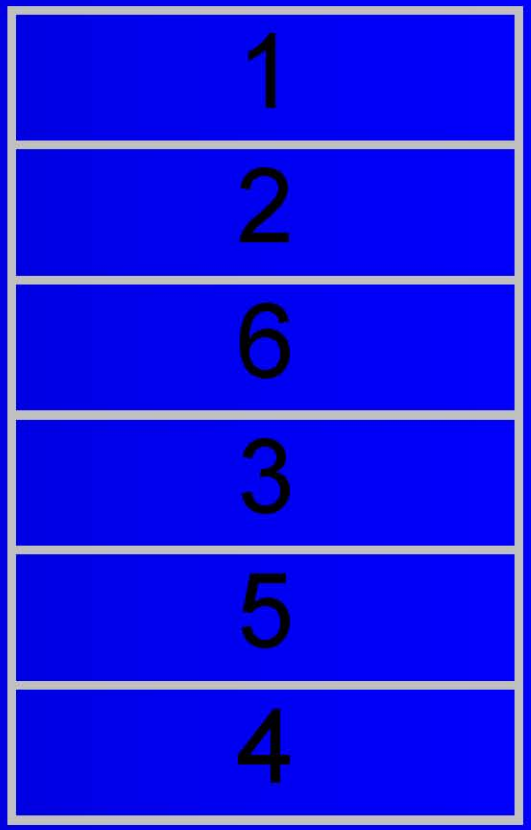
Target ?



A Sequential Search Example

Once the target data item has been found, you may return a Boolean true, or the index where it was found.

Target !



Big - O Notation

Big - O notation is used to describe the efficiency of a search or sort. The actual time necessary to complete the sort varies according to the speed of your system. Big - O notation is an approximate mathematical formula to determine how many operations are necessary to perform the search or sort. The Big - O notation for the Sequential Search is O(n), because it takes approximately n passes to find the target element.