Searching with Linear Search

Many times, it is necessary to search an array to find a particular entry.

To study the basics of this issue, we will look at an array of integers (i.e. no ) and search for a particular value called the . The answer is the of the array element that holds the . If the is not in the array, then this should be conveyed.
Linear Search

A simple algorithm for linear (or sequential) search:

Const MAX = 5;
Type Number = Array[1..MAX] of integer;

function linearSearch (A:Number, int key):integer;
{returns the index to where the search value is found}
begin
  var i:integer;
  linearSearch := -1;
  for i := 1 to MAX do
    if (A[i]=key) then linearSearch := i;
  end;
end;

_______________ running time is proportional to the _________________ in the array. O(n)
If entries are ordered, faster algorithms are possible.
Sorting

Many applications using arrays require that the data be sorted.


When the data is sorted, other operations can be done ________________ (e.g., searching).
Insertion Sort

Many sorting algorithms have been devised.

This is a simple one called insertion sort.

• Think of the array as ___________ into the part that is ___________ ___________ and the part that is _________________.

• Initially, only the ________________ of the array, A[1], is sorted. The rest, A[2..n], is not.
Insertion Sort

• Suppose we’ve sorted $A[1..i-1]$. Our task is to insert the $i^{th}$ entry, $v$, into its correct place among the already sorted entries.


sorted $v$ unsorted

To be _________ into __________ array
Insertion Sort

Start ______________ backwards from $A[i-1]$ down to $A[0]$.

________ entries to the _________ until

the proper place for $v$ is ________, and _______ $v$.

```
```

```
< v | > v | v | unsorted
```

$v$ ______ here

Must ______ this part
Insertion Sort

More detail:

• At each entry \( j \) in that range, ________ \( v \) to \( A[j] \).

• If \( A[j] > v \), then \( v \) goes ______________________ \( A[j] \);
  keep shifting -- value in \( A[j] \) is moved to \( A[j+1] \).

• If \( A[j] < v \), then \( v \) goes just after \( A[j] \), so \( v \) is put in \( A[j+1] \).

• If we ____________ the end of the array, then \( v \) is ____________
  than anything in the range, so \( v \) goes in \( A[0] \).
Pascal Code for Insertion Sort

procedure insertionSort(var A:Number);
  var i,j:integer;
  var value:integer;

  begin
    for i := 2 to MAX do
      {start at second element in array}
      begin
        value := A[i]; { current value to insert }
        j := i - 1; { between A[i-1] and A[1] }
        while ((j >= 1) AND (A[j] > value)) do
          begin
            A[j+1] := A[j];  { shift to the right }
            j := j – 1
          end;
        A[j+1] := value  {Place value}
      end
    end;
end;
Insertion Sort

The _________ running time for insertion sort is proportional to the _________ of the ______________________ to be sorted

O(__)

A visualization:
http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/insert/insertionen.htm
http://web.engr.oregonstate.edu/~minoura/cs162/javaProgs/sort/InsertSort.html

Note: The array used by these sites is indexed from 0 to n-1, where the array used in these slides is indexed from 1 to n.
procedure insertionSort(var A:Number);
var i,j:integer;
var value:integer;

begin
  for i := 2 to MAX do
    {start at second element in array}
    begin
      value := A[i]; { current value to insert }
      j := i - 1; { between A[i-1] and A[1] }
      while ((j >= 1) AND (A[j] > value)) do
        begin
          A[j+1] := A[j]; { shift to the right }
          j := j - 1
        end;
      A[j+1] := value  {Place value}
    end
  end;
end;
Modification for Practical usage

procedure insert(value: integer; var A:Number; i: integer);

var j: integer;

begin

j := i - 1;  { between A[i-1] and A[1] }
while ((j >= 1) AND (A[j] > value)) do
begin
  A[j+1] := A[j];  { shift to the right }
  j := j – 1
end;
A[j+1] := value  {Place value}
end;
Modification for Practical usage

procedure insertionSort(var A: Number);

var i, j: integer;
var value: integer;

begin

  for i := 2 to MAX do

    { start at second element in array }
    begin
      value := A[i]; { current value to insert }
      insert(value, A, i);
    end

  end;

end;
Binary Search

Recall Linear Search…

We started at one end and checked each value

With ordered Data we can do ____________.

We don’t have to look at _______ array element.

Let’s look in the _______: is it what we are looking for? NO

Is the _______ value _______ than what we are looking for?

YES

Not here

Not here

Somewhere in here!

Somewhere in here!

sorted
Binary Search

Repeat the process until we find what we are looking for or fail

We continue until we find what we are looking for, or run out of places to look thus failing to find a match.

Let’s look in the ______: is it what we are looking for? NO

Is the ______ value _______ than what we are looking for? YES

<table>
<thead>
<tr>
<th>Not here</th>
<th>Not here</th>
<th>Not here</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sorted

J. Michael Moore
function binarySearch(A: Number; key: integer): integer;
begin
  var low, high, mid: integer;
  var found: boolean;
  found := false;
  low = 1;
  high = MAX;
  while (low <= high) AND (NOT found) do
  begin
    mid = (low + high) div 2;
    if (A[mid] > key) then
      high = mid - 1
    else if (A[mid] < key) then
      low = mid + 1
    else found := true
  end;
  if found then
    binarySearch := mid
  else
    binarySearch := -1;
end;
Binary Search

Runs much _______ than linear search.

The _______ running time for binary search is proportional to the _______ of the ________________ to be searched

O(______)

___ is defined as ______

Linear search: O(______)
Binary search: O(______)

J. Michael Moore