Find maximum element in finite sequence

given input

integers: $a_1, a_2, \ldots, a_n$

n, i

$max := a_1$

```
for i := 2 to n
    { if max < ai, than max := ai
      i := i+1
    }
return (max)
```

Given $a_1 = 2$, $a_2 = 1$, $a_3 = 5$, $a_4 = 3$, $a_5 = 2$:

$n = 5$, $max = a_1 = 2$

Step 1: $i = 2$ (to 5):
is $max = 2 < a_2 = 1$? NO

Step 2 $i = 3$: is $max = 2 < a_3 = 5$ YES: set $max = a_3 = 5$

Step 3 $i = 4$ is $max = 5 < a_4 = 3$ NO

Step 4 $i = 5$ is $max = 5 < a_5 = 2$ NO

↓ Exit for loop

return $max = 5$

Complexity of find maximum in list algorithm

Examining algorithm for each integer $2, \ldots, n$ there are 2 comparisons - one to determine if $i \leq n$ and another to determine if $max < a_i$.

Between 2 and n there are $n-1$ instances of above.

When $i = n+1$ there is a final extra comparison in order to exit

Thus an then a total of $2(n-1)+1 = 2n-1$ comparisons. Thus algorithm's time complexity can be then measured by the function $f(n) = 2n-1$

And $f(n)$ is $O(n)$ - why? what is K? what is c?

Linear search

input: integers: n, x, a list $a_1, a_2, \ldots, a_n$ in increasing numeric order

find: the position of the integer x in list.
if x not present return 0.

local variables: integers $i := 1$, $location := 0$

```
while (i ≤ n)
{ if (x = ai), location := i
  i := i+1
}
return(location)
```

Given: $a_1 = 1, a_2 = 3, a_4 = 6, a_5 = 8, a_6 = 9, a_7 = 10, a_8 = 12$
$x = 6$

$i = 1$ does $x = a_1$? NO
$i = 2$ does $x = a_2$? NO
$i = 3$ does $x = a_3$? NO
$i = 4$ does $x = a_4$? YES
location = 4
$i = 5$ does $x = a_5$? NO
$i = 6$ does $x = a_6$? NO
$i = 7$ does $x = a_7$? NO
$i = 8$ does $x = a_8$? NO

We now drop out of while loop with location = 4.

Time complexity can be examined in same way as that of the "find maximum element of list" algorithm. Do it!

Binary Search

Input:
integers: x, and $a_1, a_2, \ldots, a_n$ for increasing order

find where in the list $a_1, \ldots, a_n$ one finds integer x. If x not in list return 0.

Local variables:
integers: $i := 1, j := n$; location $:= 0, m$

$n = 2^3 = 2^x$

```
while (i < j)
{ m := ⌊ (i+j)/2 ⌋
   if (x > am), i := m+1
   else j := m
}
if (x = ai), location := i
return (location)
```

Given $a_1 = 1, a_2 = 3, a_3 = 4, a_4 = 6, a_5 = 8, a_6 = 9, a_7 = 10, a_8 = 12$

$x = 6$

Note $8 = 2^3$

Step 1: $i = 1$ and $j = 8$
$m = 4$, is $x > a_4$?, NO
thus $j = 4$.
) 2 comparisons

Step 2: $i = 1$ and $j = 4$
$m = 2$, is $x > a_2$? YES
thus $i = 3$
) 2 comparisons

Step 3: $i = 3$ and $j = 4$
$m = 3$ and is $x > a_3$? YES
thus $i = 4$
) 2 comparisons

Loop is now finished, with $x = a_4 = 6$. Location number $i = 4$ is returned

Conclusion

(1) if $n = 2^{K-1}$ there are $2K+1$ comparisons

(2) if $n = 2^{K+1}$ there are $2(K+1)+1$ comparisons
$= 2K+3$

(3) if n is in between,
there are: minimum $2K+1$ comparisons
maximum $2K+3$ comparisons

* Looking at the worst case when $n = 2^{K+1}$
there are $\boxed{2\log_2 n + 1 \leq 2K+3}$ comparisons

* thus we can conclude that the time complexity is $\boxed{O(\log_2 n)}$

Justify the latter.

Time Complexity of of Binary Search

Notice that if the length of the list, $a_1, a_2, \ldots, a_n$, is a power of 2 say 2^k, then there are precisely k times that the list can successively be divided by 2.
So the previous example with $2^3 = 8$ and $k = 3$.

* As in the example, for each of k divisions of the list, there would be 2 comparisons:
  (1) in the "while" loop comparison it's
  (2) $x > a_m$

There is one final comparison to end the "while" loop

* Thus if $n = 2^k$ we conclude there are
  → $2k+1$ comparisons

* Now suppose the length of the list, n, is not a power of 2, then suppose:

$$2^k < n < 2^{k+1}$$

so that if we take $\log_2$ of the inequality we get

$$k < \log_2 n < k+1$$

If we multiply each term of the above by 2 and add 1, we get

$$2k+1 < 2\log_2 n + 1 < 2(k+1)+1$$

Bubble Sort Time Complexity

Given sequence $a_1, a_2, a_3, \ldots, a_n$

```
for(i := 1 to n-1)
{   for (j := 1 to n-i)
    {   if aj > aj+1 interchange aj and aj+1
            j := j+1
    }
    i := i+1
}
```

For $i=1$: there is the i-index comparison
for each j from 1 to n-1, j index comparisons
and the comparison $a_j > a_{j+1}$?
thus $(n-1)+(n-1) = 2(n-1)$ comparisons
finally one comparison to leave j- for loop
There are then $1 + 2(n-1) + 1 = 2n$ comparisons

$i=2$: Similary there is one i-index comparison
$2(n-2)$ j index and sequence term comparisons
1 comparison to leave j- for loop
There are then $1 + 2(n-2)+1 = 2(n-1)$ comparisons

$\vdots$

$i = n-1$: There is one i-index comparison
one j-index comparison
one $a_j > a_{j+1}$? comparison
one leave the j- for loop comparison
for a total of 4 comparisons

In general for any i, $1 \le i \le n-1$, there are $2(n-(i-1)) = 2(n-i+1)$ comparisons.

Adding the comparisons within the i-for-loop we have

$2(n + (n-1) + (n-2) + \cdots + 2)$ such comparisons

Using the result that

$$n + (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \frac{n(n+1)}{2}$$

The above becomes

$$2\left(\frac{n(n+1)}{2} - 1\right) = n(n+1) - 2 = n^2 + n - 2$$

If we add the final i index comparison when leaving the i-for-loop we have a total of

$f(n) = n^2 + n - 1$ comparisons

Noting that:

$$f(n) = n^2 + n - 1 < n^2 + n^2 = 2n^2$$

we see that f is $O(n^2)$ with $c = 2$ and $n = 1$

Insertion Sort Time Analysis

```
for (j := 2 to n)
{   i := 1
    while (aj > ai)
    { i := i+1
    }
    m := aj
    for (k := 0 to j-i-1)
    { aj-k := aj-k-1
      k := k+1
    }
    ai := m
    j := j+1
}
```

Consider sequence $a_1=2, a_2=1, a_3=5, a_4=6, a_5=7, a_6=4$

For $j=2$ and $i=1$
* $a_2 > a_1$ NO so drop out of while-loop
  $m := a_2$
* Since $j-i-1=0$ k-for-loop executes
* The k-for-loop is exited with one further comparison

Number of comparisons = 3

$j=3$ $i=1$
* $a_3 > a_1$ Y
  $i=2$
* $a_3 > a_2$ Y
  $i=3$
* $a_3 > a_3$ N exit while-loop

$m := a_3$

In the k-for-loop $j-i-1=-1$ so the loop does not execute but the index
* comparison was made

Number of comparisons = 4

New Sequence $a_1=1, a_2=2, a_3=5, a_4=6, a_5=7, a_6=4$

The sequence now is

$a_1 = 1,\ a_2 = 2,\ a_3 = 5,\ a_4 = 6,\ a_5 = 7,\ a_6 = 4$

For $j = 5$

- $i = 1$: $a_1 < a_5$ Y
- $i = 2$: $a_2 < a_5$ Y
- $i = 3$: $a_3 < a_5$ Y
- $i = 4$: $a_4 < a_5$ Y
- $i = 5$: $a_5 < a_5$ N

The k-for-loop does not execute but the index comparison is made

Number of comparisons = 6

$j = 6$

- $i = 1$: $a_1 < a_6$ Y
- $i = 2$: $a_2 < a_6$ Y
- $i = 3$: $a_3 < a_6$ N

$m := a_6$

In the k-for-loop $j - i - 1 = 2$

- $k = 0$: $a_6 := a_5$
- $k = 1$: $a_5 := a_4$
- $k = 2$: $a_4 := a_3$

There is a final k-for_loop index comparison to exit

Total number of comparisons = 7

Abstracty given the sequence

$a_1, a_2, a_3, \ldots, a_i, \ldots, a_j \ldots$

Suppose a_i is the first term of the sequence for which $a_j < a_i$. This is discovered with
* i comparisons - as in the example with $j=6$
The k for loop is then executed from
* $k=0$ to $j-i-1$ for a total of $j-i$ index comparisons.
* There is then a final k loop index comparison to exit the loop.

Total number of comparisons = $\boxed{i+j-i+1 = j+1}$

In summary as j goes from 2 to n, there are

$f(n) = 3+4+5+\cdots+n+(n+1)$ comparisons

Using the identity $1+2+3+\cdots+n = \frac{n(n+1)}{2}$

see section 5.1 p 316

then:

$$f(n) = \frac{(n+1)(n+2)}{2} - 3 = \frac{(n+1)(n+2)}{2} - 6 = \frac{n^2+3n+2}{2} - 6$$

$$= \frac{n^2+3n-4}{2}$$

$$\leq \tfrac{1}{2}(4n^2) - 2$$

$$\leq 24n^2$$

We then see that $f(n)$ is $O(n^2)$ with $C = 2$ and $k = 1$