Write down the definition of Omega.

Prove that:

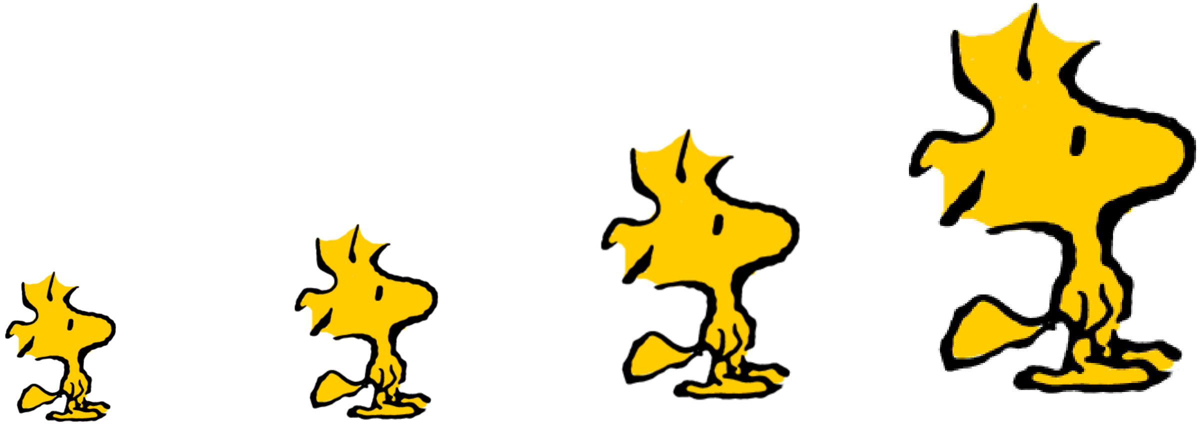
1. $T(n) = n^4 - 10\, n^2 - 100$ is in $\Omega(n^4)$

Assume that T, f and g are functions mapping the natural numbers {0, 1, 2, 3, ...} into the reals.

Definition: "Omega" A function T(n) is in $\Omega(f(n))$ if there exist constants $n_0 \geq 0$, and $c > 0$, such that for all $n \geq n_0$, $T(n) \geq c * f(n)$.

Definition: "Theta" The set $\Theta(g(n))$ of functions consists of $\Omega(g(n)) \cap O(g(n))$.

Max Sort

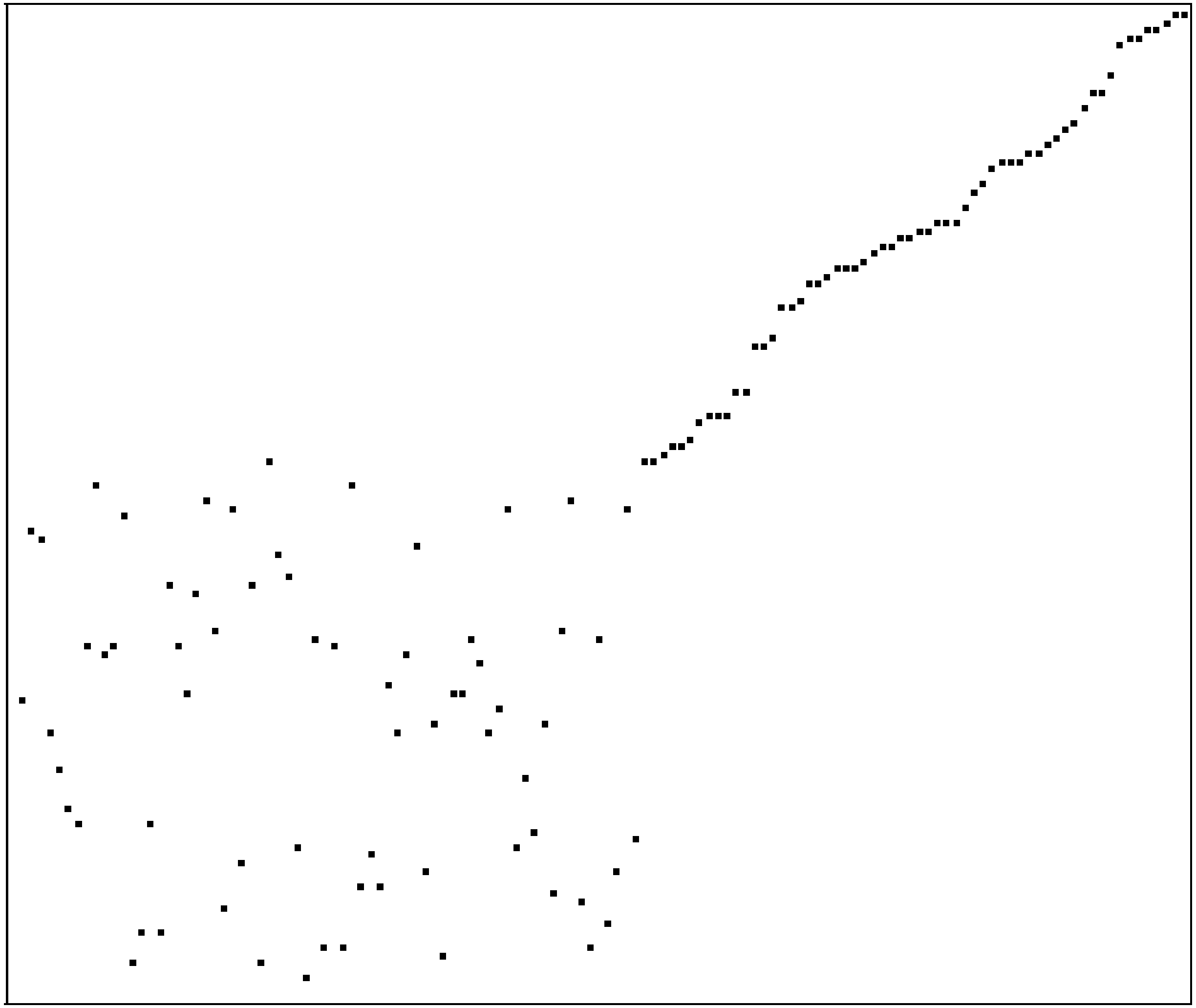


Outline:

This class starts by defining the sorting problem. Max Sort, a very simple selection sort algorithm, is introduced. Its implementation can be iterative or recursive.

The comparison model is presented. It is the basis of the time complexity analyses of the most common sorting algorithms. Because the amount of work these do is proportional to the number of key comparisons and swaps, counting these can provide reliable estimates as to running times of the algorithms on large problems.

Max Sort scatter plot



From software by Kenneth Lambert and Thomas Whaley.

Definition: A Sorting Problem (with integer data)

Given an array of n integers,

$A[0], A[1], \ldots, A[n-1]$,

rearrange the values so they are sorted:

$A[0] \le A[1] \le \ldots \le A[n-1]$.

Inductive definition: Sorted array of size n.

Throughout the term, arrays follow C/Java conventions: *A*[0..n-1].

[Basis] If n = 0 or 1, *A* is sorted.

[Inductive step] Otherwise, *A* is *sorted* if

A[n-1] ≥ *A*[0], *A*[1], ... , and *A*[n-2],

and further, *A*[0..n-2] is a sorted array.

Iterative Maxsort: Pseudocode

Maxsort(*A*[0..(n-1)])

1. for end= n-1 down to 1 do

{

    1.1 Find the position *max_pos* of the maximum element in *A*[0..end].
    1.2 Swap(*A*[max_pos], *A*[end]).

}

8	6	1	4	9	2	5	3	0	7
8	6	1	4	7	2	5	3	0	9
0	6	1	4	7	2	5	3	8	9
0	6	1	4	3	2	5	7	8	9
0	5	1	4	3	2	6	7	8	9
0	2	1	4	3	5	6	7	8	9
0	2	1	3	4	5	6	7	8	9
0	2	1	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Iterative MaxSort:

Implicit variable: this

```
public class Array
 {  int n;  int [] A;

    public void maxSort()
    {     int i, t, end, max_pos;

          for (end= n-1; end > 0; end--)
          {   max_pos=0;
              for (i= 1; i <= end; i++)
                 if (A[i] >= A[max_pos])  max_pos= i;
              t= A[max_pos];
              A[max_pos]= A[end];
              A[end]= t;
           }
     }
```

```
for (end= n-1; end > 0; end--)
{  max_pos=0;
   for (i= 1; i <= end; i++)
   {
      if (A[i] >= A[max_pos] ) max_pos= i;
   }

/*    Swap the max. element with the  end
*/
   t= A[max_pos];
   A[max_pos]= A[end];
   A[end]= t;
}
```

Definition: The Comparison Model.

Problem size: n.

Operations are:

1. Key Comparisons- compare A[i] and A[j] using ≤ or ≥ .
2. Swap(A[i], A[j]).

Not allowed: Hashing, examination of individual bits of the data, ...

In your algorithm, you can use any other variables you need and may manipulate them as you wish as long as only key comparisons and swaps are used when accessing any data values.

```
for (end= n-1; end > 0; end--)
{  max_pos=0;
   for (i= 1; i <= end; i++)
   {
      if (A[i] >= A[max_pos] ) max_pos= i;
   }
```

KEY COMPARISON

```
/*    Swap the max. element with the  end
*/
```

SWAP

```
   t= A[max_pos];
   A[max_pos]= A[end];
   A[end]= t;
}
```

Recursive Maxsort:

Maxsort(*A*[0..(n-1)])

1. [Base case] If n= 0 or 1, return.

2. Otherwise, find the position *max_pos* of the maximum element in *A*[0..(n-1)].

3. Swap(*A*[max_pos], *A*[n-1]).

4. MaxSort(*A*[0..(n-2)]).

Recursive MaxSort:

```
public void maxSort(int size)
{     int i, t, maxPos;

      if (size <= 1) return;    // Base case
      maxPos=0;
      for (i=1; i < size; i++)
            if (A[i] >= A[maxPos]) maxPos=i;
      t= A[maxPos];
      A[maxPos]= A[size-1];
      A[size-1] = t;
      maxSort(size-1);
   }
```

On a problem of size n:

How many key comparisons does MaxSort do?

How many swaps does it do?

How much time does it take to run?

How does Max Sort compare to Merge Sort on big problems?

Running time estimates:

- Home pc executes 10^8 comparisons/second.
- Supercomputer executes 10^{12} comparisons/second.

Insertion Sort (N^2)				Mergesort (N log N)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 sec	18 min
super	instant	1 second	1.6 weeks	instant	instant	instant

Table taken from notes by Robert Sedgewick and Kevin Wayne.