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 positive 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 $\Omega(g(n)) \cap O(g(n))$.

Announcements

Office hours this week:

T 12:30, 2:30

W 12:30, 1:30

F 12:30, 1:30

Please let me know if you plan to attend at one of these.

Assignment #2 parts A (due Fri. Oct. 4) and B (due Tues. Oct. 8) are posted. Read through them and let me know if you have any questions.

Relevant sections of text:

1.1: Java review.

1.2-1.3: Programming basics review.

1.4: Algorithm analysis.

We will cover 1.5 later when we do graph algorithms.

Now: Ch. 2: Sorting.

For recurrences/induction: Use a Math 122 text.


SCRATCH
Teach kids in grades 6-9 to code!
Winners will be PAID to develop games that we'll host on csc.uvic.ca
PITCH & PLAY
PITCH YOUR SCRATCH PROJECT IDEA
FRIDAY SEPTEMBER 27 at 2:30 IN ECS 660

$T_0(n) = n + 2\,T(n/2),\ T(1) = 1.$

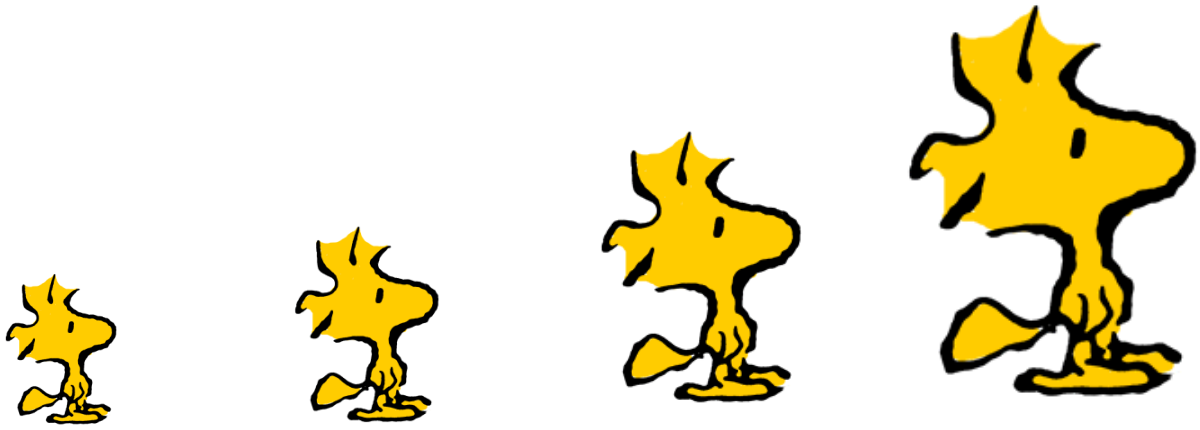
$T_1(n) = 10n + 20 + 2\,T(n/2),\ T(1) = 30$

$T_2(n) = 10n + 20n + 2\,T(n/2),\ T(1) = 30$

T_0: Used for the time complexity of mergeSort.
T_1: If for any n, mergeSort does at most 10n + 20 machine instructions at the top level of recursion (ignoring those done by a recursive call), $T_1(n)$ is an upper bound on the actual number of machine instructions.
T_2 (n): Upper bound on $T_1(n)$.

$T_0(n) \le T_1(n) \le T_2(n) = 30 * T_0(n)$ for all $n \ge 1$.
So the actual number of machine instructions $T_1(n)$ is in $O(T_0(n))$.

Max Sort

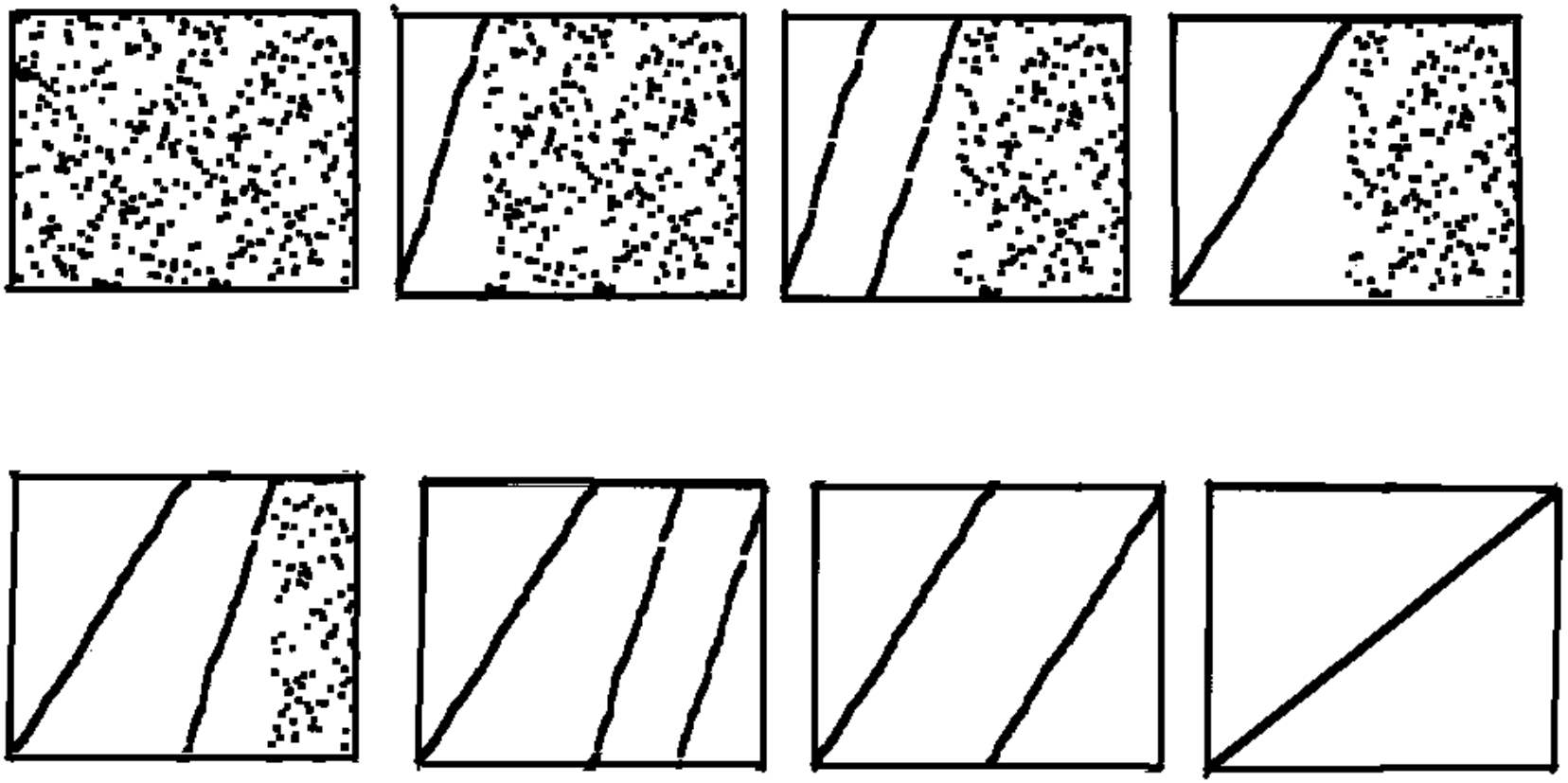


Outline:

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.

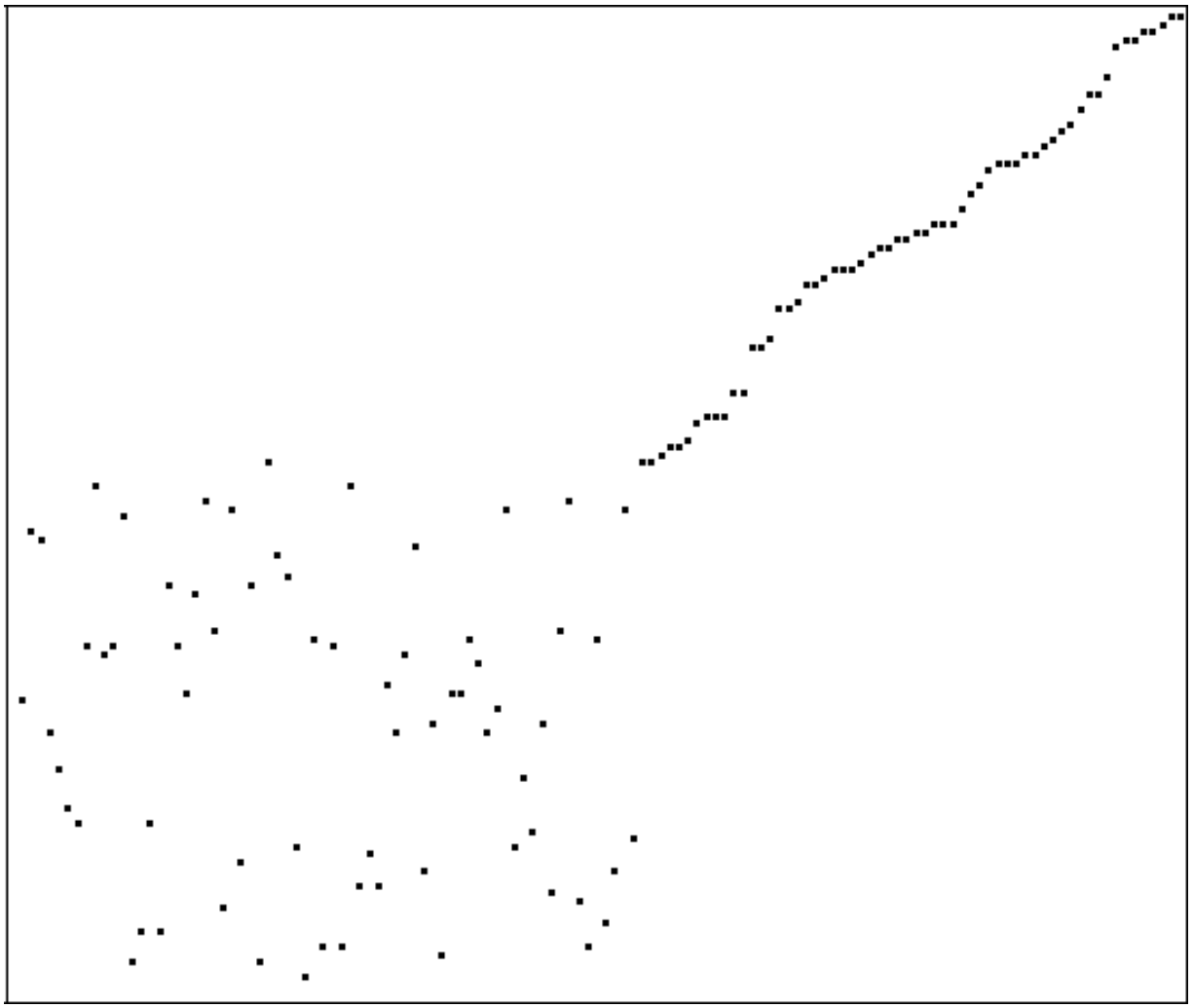
The comparison model is presented. It is the basis of the time complexity analyses of the most common sorting algorithms. Because the amount of work these do is proportional to the number of key comparisons and swaps, counting these can provide reliable estimates as to running times of the algorithms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

Scatter Plots for Merge Sort:



Taken from: Algorithms in C++ by Sedgewick.

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:

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

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

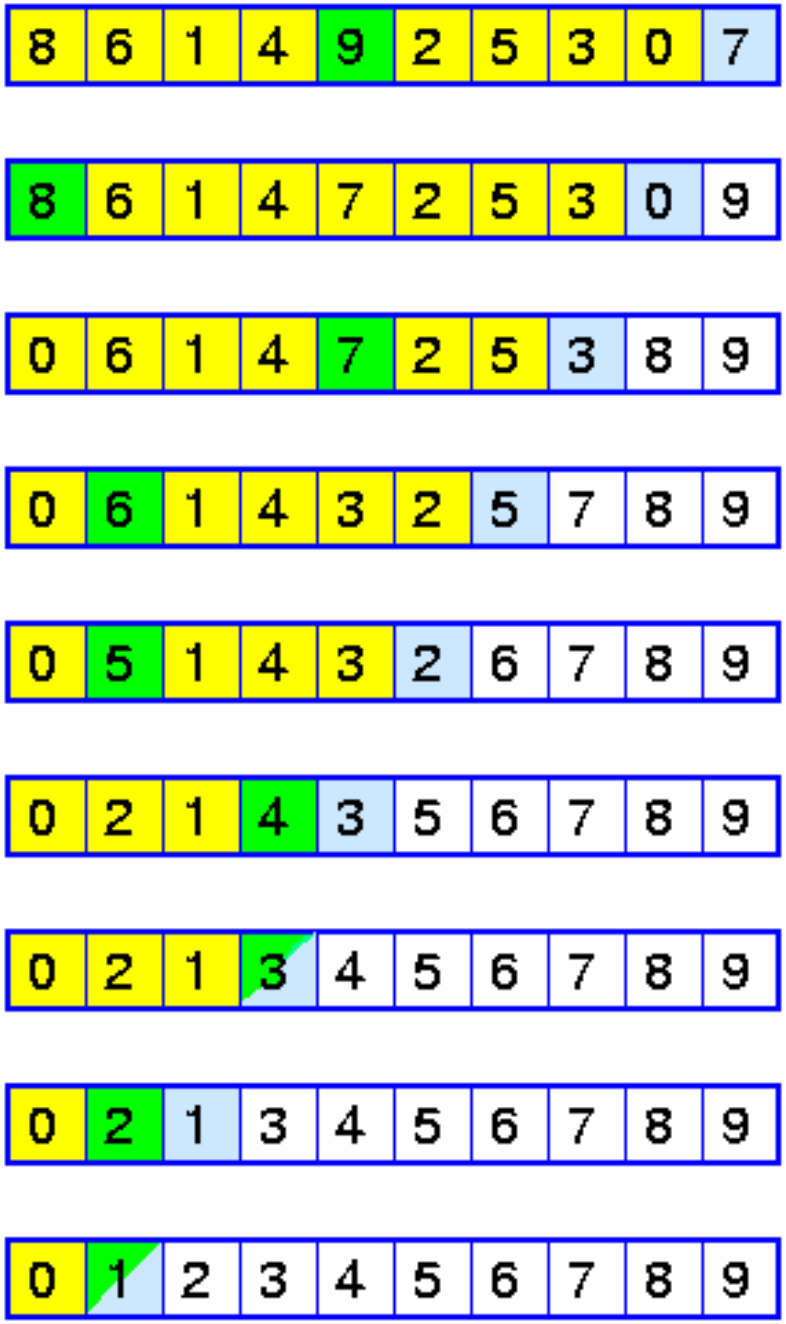
$A[n-1] \geq A[0], A[1], \ldots$, and $A[n-2]$,

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:

```
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;
        }
   }
```

Implicit variable: this

```
   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 permitted 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);
  }
```

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)

computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

Table taken from notes by Robert Sedgewick and Kevin Wayne.