Science 1000: Lecture #2 (Wareham):

The Way Things Work:
Computing with Algorithms

Got problems?
Try algorithms!
Much gets solved.

Problems

List Search:

Input: A list L of n elements and a value t.
Output: The position of the element in L with value t if such an element exists and -1 otherwise.

List Sort:

Input: A list L of n elements.
Output: The sorted version of L.

Bin Packing:

Input: A list L of the sizes of n items and a numbers B.
Output: The smallest number of bins of size B that can hold the the items in L.

List Search (Linear)

Intuition:

"Well, if I don't know anything else about the list except that it has n elements, I suppose I'll have to look at each element in the list and see if it is equal to the target-value. If I find such an element, I can stop and print that element's position; otherwise, I print -1 after I've look at all elements in the list. Sounds like a lot of work. Bummer."

List Search (Linear) [Cont'd]
Algorithm

```
tpos = -1
i = 1
while (i <= n) and (tpos == -1)) do
      if (L[i] == t) then
            tpos = i
      i = i + 1
print tpos
```

List Search (Binary)

Intuition:

"Hmmm ... Suppose this time I know L is sorted. Whenever I look at $L[i]$ where i is the middle of the list and $L[i]$'s not equal to the target-value, as L is sorted, I know that the target-value must be either above or below i in the list (depending on whether the target-value is greater or less than $L[i]$). I can keep repeating this in a loop until I either find the target-value or run out of list to search. Cool!"

List Search (Binary) [Cont'd]
Algorithm (Version #1)

```
set the current list to L
while we haven't found t in the list
      and there's still a current list
      to search do
      if t isn't the middle element of
            the current list then
            if t > middle element then
                  set current list to upper
                  part of current list
            else
                  set current list to lower
                  part of current list
```

List Search (Binary) [Cont'd]
Algorithm (Version #2)

```
t_pos = -1
left = 1
right = n
while ((t_pos == -1) and
      (left <= right)) do
      t_pos = (left + right) / 2
      if (L[t_pos] != t) then
            if (t > L[t_pos]) then
                  left = t_pos + 1
            else
                  right = t_pos - 1
            t_pos = -1
print t_pos
```

List Sort

Intuition:

"The first element in a sorted list is the smallest in the list, the second element is the smallest among the remaining elements in the list, and so on. Perhaps we could use a find-list-minimum algorithm in a loop!"

List Sort [Cont'd]
Algorithm (Version #1)

```
for i = 1 to n - 1 do
      find minimum element in L[i ..  n]
      swap minimum element and element i
```

List Sort [Cont'd]
Algorithm (Version #1)

```
for i = 1 to n - 1 do
      min_pos = i
      for scan = i + 1 to n do
            if (L[scan] < L[min_pos]) then
                  min_pos = scan
      temp = L[min_pos]
      L[min_pos] = L[i]
      L[i] = temp
```

Bin Packing

Intuition #1:

"Well, if I have at most n items, I'll need at most n bins. How about I try all possible ways of dividing the items in L among n or less bins, and then check each packing to make sure that no bin has items that are too big for their bin?"

Intuition #2:

"That sounds way too hard. How about I just do it like Doug at Sobey's – take each item in L in turn and add it to the current bin, and if that item is too large, make a new bin and add it to that one?"

Types of Algorithms

- If an algorithm always produces the answer you want, it is an **exact algorithm**; otherwise, it is a **heuristic algorithm.**
- If a heuristic produces an answer that is provably close to the one you want, it is an **approximation algorithm.**
- Each problem has many algorithms; which one should we use? Exact algorithms may not run quickly and quick heuristic or approximation algorithms may not be exact.

HOW DO WE SHOW ALGORITHMS RUN QUICKLY?

Science 1000: Lecture #2 (Wareham):

The Way Things Work:
Computing with Algorithms

Got problems?
Try algorithms!
Much gets solved.