CS 173: Discrete Structures

Eric Shaffer
Office Hour: Wed. 12-1, 2215 SC
shaffer1@illinois.edu

Section 3.1

Algorithms

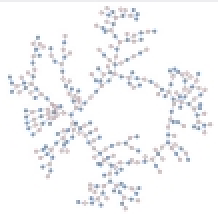
"complexity" ≡ "efficiency"

An algorithm is a clearly defined, step-by-step procedure for solving a problem.

Properties of an algorithm:

- Input and output domains are specified.
- Each step is precisely described.
- Each step is executable in finite time.
- Typically applicable to a range of inputs, not just a specific one.
- Must halt with output in finite time.

Determining if a given algorithm halts is an unsolvable problem. (meaning, we cannot write an algorithm to do it.)



Searching and Sorting Algorithms

Given a list A of ***n*** items: $a_1, a_2, \ldots, a_n$

Search(A,x): report if x in A (i.e. if $x = a_i$ for some $1 \le i \le n$)

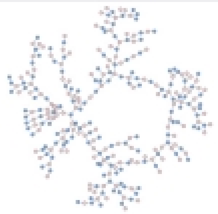
Sort(A): reorder A so that $a_1 \le a_2 \le \ldots \le a_n$

Analyze the running time

- We count the number of instructions executed
  - As a function of input size
  - Use Big-O (**We disregard constants)**

Two types of analysis:

1. Worst case analysis
2. Average case analysis



Linear Search

Linear search

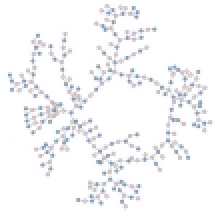
Input: an unsorted list of integers $a_1, a_2, \ldots a_n$ and integer x

Output: position i where $a_i = x$, if x is in the list

```
i:=1
position:=0
while ((i≤n) and (x ≠ a_i))
    i:=i+1
if (i≤n)then position:=i
{the variable position is location of x in the list}
{position=0 means x isn't in the list}
```

$$\Theta(1) + \Theta(n) + \Theta(1) = \Theta(n)$$

Count the instructions executed in the worst case



Algorithm Complexity

How long does the "linear search algorithm" take?

Suppose the data is (2, 8, 3, 9, 12, 1, 6, 4, 10, 7)
- Count the number of comparisons if we're looking for "4":
- How about "2"? "

The running time of the algorithm depends on the particular input to the problem. In this case we have two different complexity measures:

- Worst case complexity - running time on worst input
- Average case - average running time among all inputs

Worst case for linear search is time n. $O(n)$

Algorithm Complexity

In the average case, how long does the "linear search algorithm" take?

Assume that the number x is in the list, each spot equally likely

Average case for linear search is time
(1+2+...+n)/n = n(n+1)/2n = (n+1)/2.

$$\sum_{i=1}^{n} i \Big/ n$$

O(n)

If we allow the number x to not be in the list
-- we need to know the probability of it not being in the list
-- but in the end, the average case run time will still be O(n)

pos. of x	# items compared
1	1
2	2
⋮	⋮
n	n

Algorithm Complexity

range we're searching a_i up a_j

How long does the "binary search algorithm" take?

Binary search
Input: a **sorted** array of integers $a_1, a_2, \ldots a_n$ and an integer x
Output: position i where $a_i = x$, if x is in the list

```
i := 1
j := n
while (i < j)
    m := ⌊(i + j)/2⌋ {midpt of range (i,j)}
    if x > $a_m$ then i := m + 1
    else j := m
if x = $a_i$ then position:=i
else position:=0 {not in list}
```

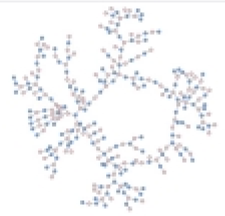
$O(1)$

$O(1)$

a_1 a_2 a_3 a_4

times through loop $\cdot O(1)$

The main point of showing you this code is to remind you that the range is cut in half at every iteration.

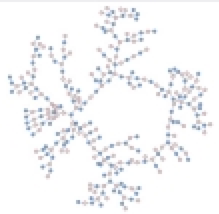


Algorithm Complexity

```
i := 1
j := n
while (i < j)
   m := ⌊(i + j)/2⌋ {midpt of range (i,j)}
   if x > $a_m$ then i := m + 1
   else j := m
if x = $a_i$ then position:=i
else position:=0 {not in list}
```

Binary search 4,7,8,10,12,14,20 for 8:

i	j	m
1	7	4
1	4	2
3	4	3
3	3	



Algorithm Complexity

How long does the "binary search algorithm" take?

If $\underbrace{\text{n is a power of 2,}}_{n = 2^k}$ $n = 2^k$, then $\boxed{k = O(\lg n)}$ iterations occur.

If n is not a power of 2, let k be the number so that $\boxed{2^k < n < 2^{k+1}}$, ~~and~~ imagine that the array has 2^{k+1} elements. Then $k+1 < \cancel{\lg n + 1} = \boxed{O(\lg n)}$