School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Searching + more complexity examples

Overview

- Analysis of recursive algorithms: simple methods in this course
- Examples for finding Complexity
  - GCD
  - Binary Search

Euclid's Algorithm

- Computing the greatest common divisor

```
int recursiveGCD(int a, int b) {
    if (b==0) return a;
    return gcd(b, a%b);
 }
```

```
int gcd(int a, int b){

  while(b != 0){
    int reminder = a % b;
    a = b;
    b = reminder;
  }

  return a;
}
```

GCD

```
int gcd(int a, int b){

  while(b != 0){
    int reminder = a % b;
    a = b;
    b = reminder;
  }

  return a;
}
```

- The number of iterations depends on the values of a and b.
- Values of a and b are monotonically decreasing.
- After 1 iteration we have $a = \min\{a,b\}$.
- We can prove that after two iterations, the value of a is at most half of what it has been before.
  - Therefore, the complexity is $O(\log \min\{a,b\})$.

GCD

Theorem: Let a and b, $a >= b$, be inputs to gcd(int a, int b). Then after at most two iterations of the while-loop we obtain a^* where $a^* <= a/2$.

Sketch of proof by case distinction:

- Value of a is monotonically decreasing and we always have $a >= b$.
- Assume that $b > a/2$ then $b' = a \% b <= a/2$ and $a' = b$ holds in the next iteration, and $a^* = a' \% b' = b \% b' <= a/2$ after two iterations due to % operation.
- Assume $b <= a/2$ then $a^* = b <= a/2$ after one iteration.

Searching an array

- Array access is O(1)
- But if we search for an element in an array
  - what's the worst case?
  - what's the best case?
- What are your assumptions?
- Do these assumptions matter?
- What's the big-O for searching an array, if we can make some assumptions about its contents?

Searching an array

- If we know that the data is sorted then we can make assumptions about where the thing that we're searching for is.
- I have an integer array of unique integers 1,2,3,4,..,10, inserted into locations 0..9 in order.
- What can I say about all the elements from location 0 to location 4?
- What if they weren't in order?

Binary Search

- In binary search, we locate the middle element in our structure or nearest to middle element and look at it.
- Is it what we're looking for? Yes, stop.
- Is it less than what we're looking for? Yes, look at the elements larger than this one.
- Is it greater? Yes, look at the smaller elements.
- Have we run out of elements? Yes, stop!

Binary Search

```
bool binarySearch(int arr[], int obj, int start, int end){

  while (start <= end){
    int middle = (start+end)/2;
    if(arr[middle] == obj)
      return true;
    else if(arr[middle] > obj)
      end = middle-1;
    else
      start = middle +1;
  }

  return false
}
```

Binary Search

- Benefits:
  - We halve the search space each time. Locating the middle element in an array is an O(1) operation, so it doesn't add complexity.
  - We know if the element isn't there without having to search everything.
- What complexity is binary search?

Binary Search

- In binary search we:
  - halve the search space every time
  - don't have to search every element
- Intuitively, this is better than O(n). But what is it?
- We keep halving the search space - so it's better than O(n/2)... $O(\log_2 n) = O(\log n)$, usually we drop the 2
- Remember logarithm rule to change basis from *b* to *c*:

$$\log_c(n) = \log_c(b) * \log_b(n)$$

  - Example $\log_2(n) = \log_2(10) * \log_{10}(n)$
- This is for worst case! Average case is roughly the same.