Questions from last time

CS 361, Lecture 2
Jared Saia
University of New Mexico

Express the following in $O$ notation

- $n^3/1000 - 100n^2 - 100n + 3$
- $\log n + 100$
- $10 \times \log^2 n + 100$
- $\sum_{i=1}^{n} i$

Today’s Outline

- Asymptotic Analysis

Computing big-O of an Algorithm

- Write down a formula, $f(n)$, which gives the number of elementary operations performed by the algorithm as a function of the input size, $n$
- Compute the big-O value for $f(n)$
An Example

Consider the following (silly) algorithm:

**Alg 1 (int n)**

For i=1 to n
   For j=1 to i
      print "hi"

Examples from last class

Following are some formulas that represent the number of operations of some algorithm. Give the big-O notation for each.

- E.g., $n$, $10,000n - 2000$, and $.5n + 2$ are all $O(n)$
- $n + \log n$, $n - \sqrt{n}$ are $O(n)$
- $n^2 + n + \log n$, $10n^2 + n - \sqrt{n}$ are $O(n^2)$
- $n \log n + 10n$ is $O(n \log n)$
- $10 \cdot \log^2 n$ is $O(\log^2 n)$
- $n\sqrt{n} + n \log n + 10n$ is $O(n \sqrt{n})$
- $10,000$, $2^{50}$ and 4 are $O(1)$

An Example (II)

- First we write down the formula $f$ giving the number of basic operations the algorithm performs: $f = \sum_{i=1}^{n^2} i = (n + 1)n/2$
- Next we compute the big-O value for $f$: $(n + 1)n/2$ is $O(n^2)$

We can then say that Alg1 takes $O(n^2)$ time. Or, for short, we just say Alg1 is $O(n^2)$

Computing big-O of an Algorithm

Following is a shorter way to compute big-O for an algorithm:

- "Atomic operations" Constant time
- Consecutive statements Sum of times
- Conditionals Larger branch time plus test time
- Loops Sum of iterations
- Function Calls Time of function body
- Recursive Functions Solve Recurrence Relation
**Alg: Linear Search**

```cpp
bool LinearSearch (int arr[], int n, int key){
    for (int i=0;i<n;i++){
        if (arr[i]==key)
            return true;
    }
    return false;
}
```

**Linear Search Analysis**

- To analyze the linear search algorithm, we consider the worst case.
- The worst case occurs when the key is the very last element in the array.
- In this case, the algorithm takes $O(n)$ time.
- Thus we say that the run time of Linear Search is $O(n)$
- (Note that the average time of Linear Search is also $O(n)$)

**Alg: Binary Search**

```cpp
bool BinarySearch (int arr[], int s, int e, int key){
    if (e-s<=0) return false;
    int mid = (e-s)/2;
    if (arr[key]==arr[mid]){
        return true;
    }else if (key < arr[mid]){  
        return BinarySearch (arr,s,mid,key);}
else{ 
    return BinarySearch (arr,mid,e,key)}
}
```

**Binary Search Analysis**

- Note that even in the worst case, the size of the array we search is being split in half in each call.
- Thus if $x$ is the number of recursive calls, and $n$ is the original size of the array, $n(1/2)^x = 1$ in the worst case.
- This implies that $2^x = n$
- Taking log of both sides, we get $x = \log n$, which means that there are $\log n$ recursive calls in the worst case.
- Since each invocation of the function takes $O(1)$ time (minus the recursive calls), and the total number of invocations is at most $\log n$, the running time is $O(\log n)$.
Comparison

- Linear Search is $O(n)$ time
- Binary Search is $O(\log n)$ time
- Binary Search is a much faster algorithm, particularly for large input sizes

A digression on logs

It rolls down stairs alone or in pairs,
and over your neighbor's dog,
it's great for a snack or to put on your back,
it's log, log, log!
- "The Log Song" from the Ren and Stimpy Show
- The log function shows up very frequently in algorithm analysis
- As computer scientists, when we use log, we'll mean $\log_2$ (i.e. if no base is given, assume base 2)

Definition

- $\log_x y$ is by definition the value $z$ such that $x^z = y$
- $x^{\log_x y} = y$ by definition

Examples

- $\log 1 = 0$
- $\log 2 = 1$
- $\log 32 = 5$
- $\log 2^k = k$

Note: $\log n$ is way, way smaller than $n$ for large values of $n$
Examples

- $\log_3 9 = 2$
- $\log_5 125 = 3$
- $\log_4 16 = 2$
- $\log_{24} 24^{100} = 100$

Facts about logs

To prove both equations, raise both sides to the power of 2, and use facts about exponents

- Fact 1: $\log(xy) = \log x + \log y$
- Fact 2: $log_a c = \log c / \log a$

Facts about exponents

Recall that:

- $(x^y)^z = x^{yz}$
- $x^y x^z = x^{y+z}$

From these, we can derive some facts about logs

Incredibly useful fact about logs

- Fact 3: $\log_c a = \log a / \log c$

To prove this, consider the equation $a = c^{\log_c a}$, take log of both sides, and use Fact 2.
Log facts to memorize

Memorize these facts

- Fact 1: \( \log(xy) = \log x + \log y \)
- Fact 2: \( \log a^c = c \log a \)
- Fact 3: \( \log_c a = \log a / \log c \)

These facts are sufficient for all your logarithm needs. (You just need to figure out how to use them)

Take Away

- All log functions of form \( k_1 \log k_2 k_3^* n^{k_4} \) for constants \( k_1, k_2, k_3 \) and \( k_4 \) are \( O(\log n) \)
- For this reason, we don’t really “care” about the base of the log function when we do asymptotic notation
- Thus, binary search, ternary search and k-ary search all take \( O(\log n) \) time

Logs and \( O \) notation

- Note that \( \log_8 n = \log n / \log 8 \)
- Note that \( \log_{600} n^{200} = 200 \times \log n / \log 600 \)
- Note that \( \log_{1000000} 30^*n^2 = 2 \times \log n / \log 1000000 + \log 30 / \log 1000000 \)
- Thus, \( \log_8 n, \log_{600} n^{600}, \) and \( \log_{1000000} 30^*n^2 \) are all \( O(\log n) \)
- In general, for any constants \( k_1 \) and \( k_2, \log_{k_1} n^{k_2} = k_2 \log n / \log k_1, \) which is just \( O(\log n) \)

Important Note

- \( \log^2 n = (\log n)^2 \)
- \( \log^2 n \) is \( O(\log^2 n), \) not \( O(\log n) \)
- This is true since \( \log^2 n \) grows asymptotically faster than \( \log n \)
- All log functions of form \( k_1 \log k_2 k_3^* n^{k_4} \) for constants \( k_1, k_2, k_3, k_4 \) and \( k_5 \) are \( O(\log^{k_2} n) \)
In-Class Exercise

Simplify and give \( O \) notation for the following functions. In the big-\( O \) notation, write all logs base 2:

- \( \log 10n^2 \)
- \( \log_5(n/4) \)
- \( \log^2 n^4 \)
- \( 2\log_4 n \)
- \( \log \log \sqrt{n} \)

Another Interview Question

- The Question: Design an algorithm to return the largest sum of contiguous integers in an array of ints
- Example: if the input is \((-10, 2, 3, -2, 0, 5, -15)\), the largest sum is 8, which we get from \((2, 3, -2, 0, 5)\).

Does big-\( O \) really matter?

Let \( n = 100000 \) and \( \Delta t = 1\mu s \)

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log n )</td>
<td>( 1.2 \times 10^{-5} ) seconds</td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( 3.2 \times 10^{-4} ) seconds</td>
</tr>
<tr>
<td>( n )</td>
<td>.1 seconds</td>
</tr>
<tr>
<td>( n \log n )</td>
<td>1.2 seconds</td>
</tr>
<tr>
<td>( n\sqrt{n} )</td>
<td>31.6 seconds</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>2.8 hours</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>31.7 years</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>&gt; 1 century</td>
</tr>
</tbody>
</table>

(from Classic Data Structures in C++ by Timothy Budd)

A Naive Algorithm

MaxSeq1 (int arr[], int n)

```c
int max = 0;
for (int i = 0; i<n; i++)
    for (int j=i; j<n; j++)
        int sum = 0;
        for (int k=i; k<=j; k++)
            sum += arr[k];
        if (sum > max)
            max = sum;
return max;
```
Analysis

- Need to count the total number of operations of MaxSeq1
- Might as well assume time to do the inner loop is 1 (since it's a constant and therefore $O(1)$)
- Let $f(n)$ be the runtime for an array of size $n$

\[
    f(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} 1
    \]

\[
    = \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1)
    \]

\[
    = \sum_{i=1}^{n} \sum_{j=i}^{n+1} j
    \]

\[
    = \sum_{i=1}^{n} ((n - i + 1)/2)(n - i + 2)
    \]

Challenge

- MaxSeq1 is very slow
- This kind of algorithm won’t impress an interviewer
- Can you do better?

Analysis (II)

\[
    f(n) = \sum_{i=1}^{n} ((n - i + 1)/2)(n - i + 2)
    \]

\[
    = \sum_{i=1}^{n} (i/2)(i + 1)
    \]

\[
    = 1/2 * \sum_{i=1}^{n} (i^2 + i)
    \]

\[
    = 1/2 * \left( \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \right)
    \]

\[
    = 1/2 * (O(n^3) + O(n^2))
    \]

\[
    = O(n^3)
    \]

Todo

- Finish pretest, due next Tuesday!
- Sign up for the class mailing list (cs361)
- Read Chapter 3 (Growth of Functions) in textbook