CS2010: ALGORITHMS AND DATA STRUCTURES

Lecture 3.1: Examples using Cost Models

Vasileios Koutavas

School of Computer Science and Statistics
Trinity College Dublin
→ Estimate the performance of algorithms by
  → **Experiments** & Observations
  → **Precise** Mathematical Calculations
  → **Approximate** Mathematical Calculations using **Cost Models**
    → Every basic operation costs 1 time unit
    → Keep only the higher-order terms
    → Count only some operations
  → Classification according to running time **order of growth**
Common order-of-growth classifications

**Good news.** The set of functions

\[ 1, \log N, N, N \log N, N^2, N^3, \text{ and } 2^N \]

suffices to describe the order of growth of most common algorithms.
# Common order-of-growth classifications

<table>
<thead>
<tr>
<th>order of growth</th>
<th>name</th>
<th>typical code framework</th>
<th>description</th>
<th>example</th>
<th>$T(2N) / T(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>constant</td>
<td>$a = b + c;$</td>
<td>statement</td>
<td>add two numbers</td>
<td>1</td>
</tr>
</tbody>
</table>
| $\log N$        | logarithmic  | while $(N > 1)$
{  
  $N = N / 2$;  
  ...  
}
                                                | divide in half  | binary search                     | $\sim 1$         |
| $N$             | linear       | for (int $i = 0$; $i < N$; $i++$)
{  
  ...  
}
                                                | loop             | find the maximum                   | 2                |
| $N \log N$      | linearithmic | [see mergesort lecture]                                                                | divide and conquer| mergesort                        | $\sim 2$      |
| $N^2$           | quadratic    | for (int $i = 0$; $i < N$; $i++$)
for (int $j = 0$; $j < N$; $j++$)
{  
  ...  
}
                                                | double loop      | check all pairs                    | 4                |
| $N^3$           | cubic        | for (int $i = 0$; $i < N$; $i++$)
for (int $j = 0$; $j < N$; $j++$)
for (int $k = 0$; $k < N$; $k++$)
{  
  ...  
}
                                                | triple loop      | check all triples                  | 8                |
| $2^N$           | exponential  | [see combinatorial search lecture]                                                     | exhaustive search | check all subsets                | $T(N)$         |
Q. Approximately how many array accesses as a function of input size \( N \)?

A. \( \sim \frac{1}{2} N^3 \) array accesses.

- Count only array accesses
- Cost of each array access: 1 time unit
- use tilde notation

Order of Growth: \( N^3 \)
→ Examples:

→ Binary Search

→ Insertion Sort
Binary search demo

Goal. Given a sorted array and a key, find index of the key in the array.

Binary search. Compare key against middle entry.
- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33
Binary search: Java implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bug in Java's Arrays.binarySearch() discovered in 2006.

```java
public static int binarySearch(int[] a, int key) {
    int lo = 0, hi = a.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

Invariant. If key appears in the array a[], then a[lo] ≤ key ≤ a[hi].
Binary search: mathematical analysis

**Proposition.** Binary search uses at most $1 + \lg N$ key compares to search in a sorted array of size $N$.

**Def.** $T(N) = \#$ key compares to binary search a sorted subarray of size $\leq N$.

**Binary search recurrence.** $T(N) \leq T(N/2) + 1$ for $N > 1$, with $T(1) = 1$.

**Pf sketch.** [assume $N$ is a power of 2]

\[
\begin{align*}
T(N) & \leq T(N/2) + 1 & \text{[ given ]} \\
& \leq T(N/4) + 1 + 1 & \text{[ apply recurrence to first term ]} \\
& \leq T(N/8) + 1 + 1 + 1 & \text{[ apply recurrence to first term ]} \\
& \vdots & \\
& \leq T(N/N) + 1 + 1 + \ldots + 1 & \text{[ stop applying, $T(1) = 1$ ]} \\
& = 1 + \lg N
\end{align*}
\]
Example: 3-SUM

Q. Approximately how many array accesses as a function of input size $N$?

A. \( \frac{1}{2} N^3 \) array accesses.

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        for (int k = j+1; k < N; k++)
            if (a[i] + a[j] + a[k] == 0)
                count++;
```

\[
\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \\
\sim \frac{1}{6} N^3
\]
An $N^2 \log N$ algorithm for 3-SUM

**Algorithm.**

- Step 1: Sort the $N$ (distinct) numbers.
- Step 2: For each pair of numbers $a[i]$ and $a[j]$, binary search for $-(a[i] + a[j])$.

**Input**

$$30 \ -40 \ -20 \ -10 \ 40 \ 0 \ 10 \ 5$$

**Sort**

$$-40 \ -20 \ -10 \ 0 \ 5 \ 10 \ 30 \ 40$$

**Binary Search**

$$
\begin{align*}
(-40, -20) & : 60 \\
(-40, -10) & : 50 \\
(-40, \ 0) & : 40 \quad \text{(circled)} \\
(-40, \ 5) & : 35 \\
(-40, \ 10) & : 30 \quad \text{(circled)} \\
\vdots & : \vdots \\
(-20, \ -10) & : 30 \quad \text{(circled)} \\
\vdots & : \vdots \\
(-10, \ 0) & : 10 \quad \text{(circled)} \\
\vdots & : \vdots \\
(\ 10, \ 30) & : -40 \\
(\ 10, \ 40) & : -50 \\
(\ 30, \ 40) & : -70
\end{align*}
$$

What is the order of growth?

---

only count if $a[i] < a[j] < a[k]$ to avoid double counting
Algorithm.

- Step 1: Sort the $N$ (distinct) numbers.
- Step 2: For each pair of numbers $a[i]$ and $a[j]$, binary search for $- (a[i] + a[j])$.

Analysis. Order of growth is $N^2 \log N$.

- Step 1: $N^2$ with insertion sort.
- Step 2: $N^2 \log N$ with binary search.

Remark. Can achieve $N^2$ by modifying binary search step.

<table>
<thead>
<tr>
<th>input</th>
<th>30 -40 -20 -10 40 0 10 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>-40 -20 -10 0 5 10 30 40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>binary search</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-40, -20) 60</td>
</tr>
<tr>
<td>(-40, -10) 50</td>
</tr>
<tr>
<td>(-40, 0) 40</td>
</tr>
<tr>
<td>(-40, 5) 35</td>
</tr>
<tr>
<td>(-40, 10) 30</td>
</tr>
<tr>
<td>(-20, -10) 30</td>
</tr>
<tr>
<td>(-10, 0) 10</td>
</tr>
<tr>
<td>(10, 30) -40</td>
</tr>
<tr>
<td>(10, 40) -50</td>
</tr>
<tr>
<td>(30, 40) -70</td>
</tr>
</tbody>
</table>

only count if $a[i] < a[j] < a[k]$ to avoid double counting
Comparing programs

**Hypothesis.** The sorting-based $N^2 \log N$ algorithm for 3-SUM is significantly faster in practice than the brute-force $N^3$ algorithm.

<table>
<thead>
<tr>
<th>$N$</th>
<th>time (seconds)</th>
<th>$N$</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.1</td>
<td>1,000</td>
<td>0.14</td>
</tr>
<tr>
<td>2,000</td>
<td>0.8</td>
<td>2,000</td>
<td>0.18</td>
</tr>
<tr>
<td>4,000</td>
<td>6.4</td>
<td>4,000</td>
<td>0.34</td>
</tr>
<tr>
<td>8,000</td>
<td>51.1</td>
<td>8,000</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16,000</td>
<td>3.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32,000</td>
<td>14.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64,000</td>
<td>59.16</td>
</tr>
</tbody>
</table>

*ThreeSum.java*  

*ThreeSumDeluxe.java*

**Guiding principle.** Typically, better order of growth $\Rightarrow$ faster in practice.