This is CS 50.

Harvard College’s Introduction to Computer Science I

COMPUTER SCIENCE 50

WEEK 3

DAVID J. MALAN ’99
malan@post.harvard.edu
Divide and Conquer
Roll Call

1. Stand up.

2. Think to yourself “I am #1”.

3. Pair off with someone standing, add your numbers together, and adopt the sum as your new number.

4. One of you should sit down, the other should go back to step 3.
Running Time

$T(n)$
## Running Time

\[ T(n) \]

<table>
<thead>
<tr>
<th>( \log_2 \log_2 n )</th>
<th>( \log_2 n )</th>
<th>( n )</th>
<th>( n \log_2 n )</th>
<th>( n^2 )</th>
<th>( n^3 )</th>
<th>( 2^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>1.58</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>4096</td>
<td>65536</td>
</tr>
<tr>
<td>2.32</td>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1024</td>
<td>32768</td>
<td>4294967296</td>
</tr>
<tr>
<td>2.6</td>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4096</td>
<td>2.6 \times 10^5</td>
<td>1.85 \times 10^{19}</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>256</td>
<td>2.05 \times 10^3</td>
<td>6.55 \times 10^4</td>
<td>1.68 \times 10^7</td>
<td>1.16 \times 10^{77}</td>
</tr>
<tr>
<td>3.32</td>
<td>10</td>
<td>1024</td>
<td>1.02 \times 10^4</td>
<td>1.05 \times 10^6</td>
<td>1.07 \times 10^9</td>
<td>1.8 \times 10^{308}</td>
</tr>
<tr>
<td>4.32</td>
<td>20</td>
<td>1048576</td>
<td>2.1 \times 10^7</td>
<td>1.1 \times 10^{12}</td>
<td>1.15 \times 10^{18}</td>
<td>6.7 \times 10^{315652}</td>
</tr>
</tbody>
</table>

**Table 7.1** Common Computing Time Functions

Figure from *C++: An Introduction to Data Structures*, by Larry Nyhoff.
Running Time

$T(n)$

Figure from *C++: An Introduction to Data Structures*, by Larry Nyhoff.
Asymptotic Notation

Informally

$O$, $\Theta$, $\Omega$
Asymptotic Notation

Formally

\[ T(n) \in O(f(n)) \]

We say that the running time, \( T(n) \), of an algorithm is “in big \( O \) of \( f \) of \( n \)” iff there exist an integer \( n_0 > 0 \) and a real number \( c > 0 \) such that \( T(n) \leq c \cdot f(n) \) for all \( n \geq n_0 \).

\[ T(n) \in \Theta(f(n)) \]

We say that the running time, \( T(n) \), of an algorithm is “in theta of \( f \) of \( n \)” iff there exist an integer \( n_0 \) and real numbers \( c_1, c_2 > 0 \) such that \( c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n) \) for all \( n \geq n_0 \).

\[ T(n) \in \Omega(f(n)) \]

We say that the running time, \( T(n) \), of an algorithm is “in omega of \( f \) of \( n \)” iff there exist an integer \( n_0 \) and a real number \( c > 0 \) such that \( T(n) \geq c \cdot f(n) \) for all \( n \geq n_0 \).
### In English

- **O(1)**                “constant”  
- **O(log n)**          “logarithmic”  
- **O(n)**                “linear”  
- **O(n log n)**       “supralinear”  
- **O(n^2)**              “quadratic”  
- **O(n^c)**              “polynomial”  
- **O(n!)**                “factorial”
Searching
Linear Search

Pseudocode

On input $n$:

For each element $i$:
  
  If $i == n$:
    
    Return true.

Return false.
Binary Search
Iterative Pseudocode

On input array [0], ... , array [n − 1] and k:
Let first = 0.
Let last = n − 1.
While first <= last:
    Let middle = (first + last ) / 2.
    If k < array [middle] then let last = middle − 1.
    Else if k > array [middle] then let first = middle + 1.
    Else return true.

Return false.
int
sigma(int m)
{
    // avoid risk of infinite loop
    if (m < 1)
        return 0;

    // return sum of 1 through m
    int sum = 0;
    for (int i = 1; i <= m; i++)
        sum += i;
    return sum;
}

see
sigma1.c
```c
int sigma(int m)
{
    // base case
    if (m <= 0)
        return 0;

    // recursive case
    else
        return (m + sigma(m-1));
}
```

see `sigma2.c`
The Stack

Frames

```
main()
main()'s parameters

main()
main()'s parameters

foo()
foo()'s parameters
```
On input array, first, last, and k, define recurse as:

If first > last then return false.

Let middle = (first + last) / 2.

Else if k < array[middle] then

   return recurse(array, first, middle - 1, k).

Else if k > array[middle] then

   return recurse(array, middle + 1, last, k).

Else return true.
Sorting

4 2 6 8 1 3 7 5
Bubble Sort

Pseudocode

Repeat n times:
  For each element i:
    If element i and its neighbor are out of order:
      Swap them.
Selection Sort

Pseudocode

Let $i := 0$.
Repeat $n$ times:
  Find smallest value, $s$, between $i$ and list's end, inclusive.
  Swap $s$ with value at location $i$.
Let $i := i + 1$. 


Sorting
Visualization
Merge Sort

Pseudocode

On input of \( n \) elements:

If \( n < 2 \), return.

Else

Sort left half of elements.
Sort right half of elements.
Merge sorted halves.
Merge Sort

Pseudocode

\[ T(n) = 0, \text{ if } n < 2 \]

\[ T(n) = T(n/2) + T(n/2) + O(n), \text{ if } n > 1 \]
Merge Sort

How long does it take to sort 16 elements?

\[
\begin{align*}
T(16) &= 2T(8) + 16 \\
T(8) &= 2T(4) + 8 \\
T(4) &= 2T(2) + 4 \\
T(2) &= 2T(1) + 2 \\
T(1) &= 0
\end{align*}
\]
Merge Sort

T (1) = 0
T (2) = 2T (1) + 2 = 0 + 2 = 2
T (4) = 2T (2) + 4 = 4 + 4 = 8
T (8) = 2T (4) + 8 = 16 + 8 = 24
T (16) = 2T (8) + 16 = 48 + 16 = 64
Sorting
Visualization
All Sorts of Sorts

- Heap Sort
- Insertion Sort
- Quicksort
- Radix Sort
- Shell Sort
- ...

Tradeoffs

Space, Time, ...