What can $S$ be?

$S = \sum_{j=2}^{n} t_j$

- **Best case:**
- **Worst case:**
- **Average case:**

Inner loop stops when $A[i] \leq key$, or $i = 0$
Best case

Inner loop stops when A[i] <= key, or i = 0

- Array already sorted
- $S = \sum_{j=2}^{n} t_j$
- $t_j = 1$ for all j
- $S = n-1$  $\mathcal{T}(n) = \Theta(n)$
Worst case

- Array originally in reverse order sorted
- $S = \sum_{j=2}^{n} t_j$
- $t_j = j$
- $S = \sum_{j=2}^{n} j = 2 + 3 + \ldots + n = (n-1) (n+2) / 2 = \Theta (n^2)$

Inner loop stops when $A[i] \leq key$
Average case

- Array in random order
- \( S = \sum_{j=2}^{n} t_j \)
- \( t_j = j / 2 \) on average
- \( S = \sum_{j=2}^{n} j/2 = \frac{1}{2} \sum_{j=2}^{n} j = (n-1) (n+2) / 4 = \Theta (n^2) \)

Inner loop stops when \( A[i] \leq key \)

![Diagram showing array in random order with inner loop stopping condition](image)
Analyzing Insertion Sort

- \( T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4S + c_5(S - (n-1)) + c_6(S - (n-1)) + c_7(n-1) \)
- \( = c_8S + c_9n + c_{10} \)

What can \( S \) be?

- **Best case -- inner loop body never executed**
  - \( t_j = 1 \Rightarrow S = n - 1 \)
  - \( T(n) = an + b \) is a linear function

- **Worst case -- inner loop body executed for all previous elements**
  - \( t_j = j \Rightarrow S = 2 + 3 + \ldots + n = n(n+1)/2 - 1 \)
  - \( T(n) = an^2 + bn + c \) is a quadratic function

- **Average case**
  - Can assume that on average, we have to insert \( A[j] \) into the middle of \( A[1..j-1] \), so \( t_j = j/2 \)
  - \( S \approx n(n+1)/4 \)
  - \( T(n) \) is still a quadratic function

\( \Theta(n) \)

\( \Theta(n^2) \)

\( \Theta(n^2) \)
Asymptotic Analysis

- Ignore actual and abstract statement costs
- **Order of growth** is the interesting measure:
  - Highest-order term is what counts
  - As the input size grows larger it is the high order term that dominates
Comparison of functions

<table>
<thead>
<tr>
<th></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>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>10</td>
<td>33</td>
<td>$10^2$</td>
<td>$10^3$</td>
<td>$10^3$</td>
<td>$10^6$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>6.6</td>
<td>$10^2$</td>
<td>660</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>$10^{30}$</td>
<td>$10^{158}$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>10</td>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td>13</td>
<td>$10^4$</td>
<td>$10^5$</td>
<td>$10^8$</td>
<td>$10^{12}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^5$</td>
<td>17</td>
<td>$10^5$</td>
<td>$10^6$</td>
<td>$10^{10}$</td>
<td>$10^{15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^6$</td>
<td>20</td>
<td>$10^6$</td>
<td>$10^7$</td>
<td>$10^{12}$</td>
<td>$10^{18}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For a super computer that does 1 trillion operations per second, it will be longer than 1 billion years.
Order of growth

$1 \ll \log_2 n \ll n \ll n \log_2 n \ll n^2 \ll n^3 \ll 2^n \ll n!$

(We are slightly abusing of the ‘‘$\ll$’’ sign. It means a smaller order of growth).
Exact analysis is hard!

- Worst-case and average-case are difficult to deal with precisely, because the details are very complicated

It may be easier to talk about upper and lower bounds of the function.
Your turn
Fibonacci numbers

The Fibonacci numbers:
0, 1, 1, 2, 3, 5, 8, 13, 21, ...

The Fibonacci recurrence:
\[ F(n) = F(n-1) + F(n-2) \]
\[ F(0) = 0 \]
\[ F(1) = 1 \]
Solution

1
int fib(int n)
{
    if (n <= 2) return 1
    else return fib(n-1) + fib(n-2)
}

2
int fib(int n)
{
    int f[n+1];
    for (int i = 3; i <= n; i++)
        f[i] = f[i-1] + f[i-2];
    return f[n];
}