Example

Code:

sum1 = 0;
for (k=1; k<=n; k*=2)
    for (j=1; j<=n; j++)
        sum1++;

Complexity:

Code:

sum2 = 0;
for (k=1; k<=n; k*=2)
    for (j=1; j<=k; j++)
        sum2++;

Complexity:

Some Questions

1. Is upper bound the same as worst case?

2. Does lower bound happen with shortest input?

3. What if there are multiple parameters?
   
   Ex: Rank order of p pixels in c colors
   
   for (i = 0; i < c; i++)
       count[i] = 0;
   for (i = 0; i < p; i++)
       count[value(i)]++;
   sort(count)
Space Complexity

Does it matter?

What determines space complexity?

How can you reduce it?

What tradeoffs are involved?

Constants in Bounds

Theorem:
\( O(cf(x)) = O(f(x)) \)

Proof:
- \( T(x) = O(cf(x)) \) implies that there are constants \( c_0 \) and \( n_0 \) such that \( T(x) \leq c_0(cf(x)) \) when \( x \geq n_0 \)
- Therefore, \( T(x) \leq c_1(f(x)) \) when \( x \geq n_0 \) where \( c_1 = c_0c \)
- Therefore, \( T(x) = O(f(x)) \)
Sum in Bounds

Theorem:
Let \( T_1(n) = O(f(n) \) and \( T_2(n) = O(g(n)) \).
Then \( T_1(n) + T_2(n) = O(\max(f(n),g(n))) \).

Proof:
– From the definition of \( O \), \( T_1(n) \leq c_1 f(n) \) for \( n \geq n_1 \) and \( T_2(n) \leq c_2 g(n) \) for \( n \geq n_2 \).
– Let \( n_0 = \max(n_1, n_2) \).
– Then, for \( n \geq n_0 \), \( T_1(n) + T_2(n) \leq c_1 f(n) + c_2 g(n) \).
– Let \( c_3 = \max(c_1, c_2) \).
– Then, \( T_1(n) + T_2(n) \leq c_3 f(n) + c_3 g(n) \)
  \[ \leq 2c_3 \max(f(n), g(n)) \]
  \[ \leq c \max(f(n), g(n)) \]

Products in Bounds

Theorem:
Let \( T_1(n) = O(f(n) \) and \( T_2(n) = O(g(n)) \).
Then \( T_1(n)T_2(n) = O(f(n),g(n)) \).

Proof:
– \( T_1(n) T_2(n) \leq c_1 c_2 f(n) g(n) \) when \( n \geq n_0 \)
– Therefore, \( T_1(n)T_2(n) = O(f(n),g(n)) \).
Polynomials in Bounds

Theorem:
If $T(n)$ is a polynomial of degree $x$, then $T(n) = O(n^x)$.

Proof:
– $T(n) = n^x + n^{x-1} + \ldots + k$ is a polynomial of degree $x$.
– By the sum rule, the largest term dominates.
– Therefore, $T(n) = O(n^x)$.

L’Hospital’s Rule

Finding limit of ratio of functions as variable approaches $\infty$

\[ \lim_{x \to \infty} \frac{f(x)}{g(x)} = \lim_{x \to \infty} \frac{f'(x)}{g'(x)} \]

Use to determine $O$ or $\Omega$ ordering of two functions

$f(x) = O(g(x))$ if $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$

$f(x) = \Omega(g(x))$ if $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$
Polynomials of Logarithms in Bounds

Theorem:
\( \lg^n n = O(n) \) for any positive constant \( k \)

Proof:
- Note that \( \lg^k n \) means \( (\lg n)^k \).
- Need to show \( \lg^k n \leq cn \) for \( n \geq n_0 \). Equivalently, can show \( \lg n \leq cn^{1/k} \)
- Letting \( a = 1/k \), we will show that \( \lg n = O(n^a) \) for any positive constant \( a \). Use L’Hospital’s rule:
  \[
  \lim_{n \to \infty} \frac{\lg n}{cn^a} = \lim_{n \to \infty} \frac{\frac{\lg e}{n}}{a \cdot cn^{a-1}} = \lim_{n \to \infty} \frac{c}{n^a} = 0
  \]
  Ex: \( \lg^{1000000}(n) = O(n) \)

Polynomials vs Exponentials in Bounds

Theorem:
\( n^k = O(a^n) \) for \( a > 1 \)

Proof:
- Use L’Hospital’s rule
  \[
  \lim_{n \to \infty} \frac{n^k}{a^n} = \lim_{n \to \infty} \frac{kn^{k-1}}{a^n \ln a} = \lim_{n \to \infty} \frac{k(k-1)n^{k-2}}{a^n \ln^2 a} = \cdots = \lim_{n \to \infty} \frac{k(k-1)\cdots1}{a^n \ln^k a} = 0
  \]
  Ex: \( n^{1000000} = O(1.00000001^n) \)
Relative Orders of Growth

n (linear)
logₖn for k < 1
constant
n¹⁺k for k > 0 (polynomial)
2ⁿ (exponential)
n log n
logₖn for k > 1
nᵏ for k < 1
log n
List ADT (expanded from Weiss)

A list is a dynamic ordered tuple of homogeneous elements

\[ A_1, A_2, A_3, \ldots, A_N \]

where \( A_i \) is the \( i \)th element of the list

Definition: The position of element \( A_i \) is \( i \); positions range from 1 to \( N \) inclusive

Definition: The size of a list is \( N \) (a list of NO elements is called “an empty list”)

Operations on a List

List() -- construct an empty list
List(const List &rhs) -- construct a list as a copy of rhs
~List() -- destroy the list
const List &operator=(const List &rhs)
    - make this list contain copies of the elements of rhs in the same order
    - elements are deep copied from rhs, not used directly. If \( L_1 = (A_1, A_2, A_3) \) and \( L_2 = (B_1, B_2) \) before the assignment, then \( L_1 = L_2 \) causes \( L_2 = (A_1, A_2, A_3) \)
Operations on a List (cont)

Bool isEmpty() const -- returns true if the list size is zero
void makeEmpty() -- causes the list to become empty
void remove (const Object &x)
   – the first occurrence of x is removed from the list, if it is
     present. If x is not present, the list is unchanged.
   – an occurrence of x is an element $A_i$ of the list such that
     $A_i == x$

Also:
insert
find
findPrevious

Iterators

An *iterator* is an object that provides access to the elements
of a collection (in a specified order) without exposing the
underlying structure of the collection.
   – order dictated by the iterator
   – collection provides iterators on demand
   – each iterator on a collection is independent
   – iterator operations are generic
Iterator Operations

Bool isPastEnd() -- returns true if the iterator is past the end of the list

void advance() -- advances the iterator to the next position in the list. If iterator already past the end, no change.

const Object &retrieve() -- returns the element in the list at the current position of the iterator. It is an error to invoke “retrieve” on an iterator that isPastEnd

List Operations

ListIter<Object> first() -- returns an iterator representing the first element on the list

List Iter<Object> zeroth() -- returns an iterator representing the header of a list

ListIter<Object> find(const Object &x) -- returns an iterator representing the first occurrence of x in the list. If x not present, the iterator isPastEnd.

ListIter<Object> findPrevious(const Object &x) -- returns an iterator representing the element before x in the list. If x is not in the list, the iterator represents the last element in the list. If x is first element (or list is empty), the iterator returned is equal to the one returned by zeroth().
List Operators (cont)

void insert (const Object &x, const listIter<Object> &p)
– inserts a copy of x in the list after the element referred to by p
– if p isPastEnd, the insertion fails without an indication of failure.

Ex: Building a List

List<int> list;  // empty list of int
ListIter<int> iter = list.zeroth();
for (int i=0; i < 5; i++) {
    list.insert(iter);
    iter.advance();
}
Ex: Building a List #2

```c
Ex: Building a List #2

List<int> list;  // empty list of int
ListIter<int> iter = list.zeroth();
for (int i=0; i < 5; i++) {
    list.insert(iter);
}
```