- Last time
  - Review induction proofs
  - Loop invariants
- Today
  - Theoretical efficiency metrics
    - Basic notations
  - A direct analysis of selection sort and insertion sort
  - Asymptotic Notation

Elementary Algorithmics

- Given a problem
  - What's an instance
  - Instance size
- What does efficiency mean?
  - Time
  - Space

Instance of a problem

- Instance: problem + input
- Problem: calculate Fibonacci(n)
  - Fibonacci(45) is an ***instance*** of the problem
- ***Domain of definition*** of a problem: the set of instances to be considered
  - A correct algorithm should work for every instance

Efficiency of an algorithm

- Efficiency
  - **Time**, space, energy
  - Measured as a function of the size of the instances considered
- Input Size
  - The ***size*** of an instance/input
    - corresponds formally the number of the bits needed to represent the instance on a computer
    - A less formal definition: any integer that in some way measures the number of components in an instance
      - For example, sorting, graphs
    - For problems involving integers, we use ***value*** rather than size
- Running time
  - The number of primitive operations executed in terms of input size.

Approaches to measure efficiency

- Empirical Approach
  - Experiments through limited instances
- Theoretical Approach (one focus of this course)
  - Determines mathematically the quantity of resources needed by an algorithm
- Hybrid approach
  - Given an implementation in a machine, predict the efficiency of an instance using limited experiments

Average, best, and worst case analysis

- How to compare two algorithms
  - Worst case, average, best case
- Worst case
  - Appropriate for an algorithm whose response time is critical
- Average
  - For an algorithm which is to be used many times on many different instances
  - Harder to analyze, need to know the distribution of the instances
- Best case

Machine Model and Elementary (Primitive) Operation

- Assuming RAM (random-access machine) model
  - Instructions and costs are well-defined
  - Realistic
  - No concurrent operations
- An elementary (primitive) operation is one whose execution time can be bounded above by a constant depending only on the particular implementation—the machine, the programming language, etc.
- Example
  - X = Sum{A[i] | 1 <= i <= n}
  - Fibonacci sequence, addition may not be an elementary operation

Insertion sort vs. Selection sort

```
void insertionSort(int A[], int n)
{
  int i, j, tmp;

  for (i=1; i<n; i++) {
    tmp=A[i];
    j = i-1;
    while (j>=0 && tmp<A[j]) {
      A[j+1] = A[j];
      j--;
    }
    A[j+1] = tmp;
  }
}
```

```
int selectionSort(int A[], int n)
{
  int i, j, minj, minv;

  for (i=0; i<n-1; i++) {
    minj=i; minv=A[i];
    for (j=i+1; j<n; j++) {
      if (A[j]<minv) {
        minv = A[j];
        minj = j;
      }
    }
    A[minj] = A[i];
    A[i] = minv;
  }
}
```

For best-case and worst-case, consider:
- A is in ascending order
- A is in descending order

A detailed worst–case analysis of selection sort

Assumption!

```
int selectionSort(int A[], int n)
{
  int i, j, minj, minv;

  for (i=0; i<n-1; i++) {            First time 3, later 2
    minj=i; minv=A[i];               1 + 2
    for (j=i+1; j<n; j++) {          First time 3, later 2
      if (A[j]<minv) {               2
        minv = A[j];                 2
        minj = j;                    1
      }
    }
    A[minj] = A[i];                  3
    A[i] = minv;                     2
  }
}
```

n-i-1 times

n-1 times

Total elementary operations:

$$1+\sum_{i=0}^{n-2}(2+3+3+2+1+\sum_{j=i+1}^{n-1}(2+2+2+1))=1+\sum_{i=0}^{n-2}(11+\sum_{j=i+1}^{n-1}7)=\frac{7}{2}n^2+\frac{15}{2}n-10$$

Insertion sort analysis

code	cost	times
`void insertionSort(int A[], int n)`		
`{`		
`int i, j, tmp;`		
`for (i=1; i<n; i++) {`	c_1	n
`tmp=A[i];`	c_2	n-1
`j = i-1;`	c_4	n-1
`while (j>=0 && tmp<A[j]) {`	c_5	$\sum_{i=1}^{n-1} t_i$
`A[j+1] = A[j];`	c_6	$\sum_{i=1}^{n-1} (t_i-1)$
`j--;`	c_7	$\sum_{i=1}^{n-1} (t_i-1)$
`}`		
`A[j+1] = tmp;`	c_8	n-1
`}`		
`}`		

best case $t_i = 1$

worst case $t_i = i + 1$

Insert sort cost

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{i=1}^{n-1}t_i+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7\sum_{i=1}^{n-1}(t_i-1)+c_8(n-1)$$

best case $t_i = 1$

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{i=1}^{n-1}t_i+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7\sum_{i=1}^{n-1}(t_i-1)+c_8(n-1)$$

$$=c_1 n+c_2(n-1)+c_4(n-1)+c_5(n-1)+c_8(n-1)$$

$$=(c_1+c_2+c_4+c_5+c_8)n-(c_2+c_4+c_5+c_8)$$

worst case $t_i = i + 1$

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{i=1}^{n-1}t_i+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7\sum_{i=1}^{n-1}(t_i-1)+c_8(n-1)$$

$$=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{i=1}^{n-1}(i+1)+c_6\sum_{i=1}^{n-1}i+c_7\sum_{i=1}^{n-1}i+c_8(n-1)$$

$$=\frac{c_5+c_6+c_7}{2}n^2+(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6+c_7}{2}+c_8)n-(c_2+c_4+c_5+c_8)$$

Asymptotic Notation

- What does “the order of” mean
- Big O, Ω, and Θ notations
- Properties of asymptotic notation
- Limit rule

A notation for "the order of"

- We'd like to measure the efficiency of an algorithm
  - Determine mathematically the resources needed
- There is no such a computer which we can refer to as a standard to measure computing time
- We introduce "asymptotic" notation
  - An asymptotically superior algorithm is often preferable even on instances of moderate size (We saw this when comparing two Fibonacci algorithms)

Efficiency of selection sort

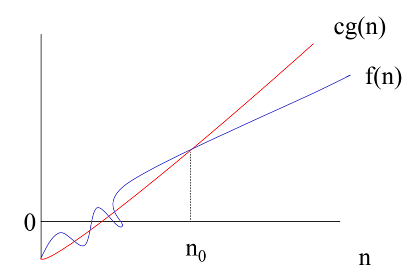
- T(n) is the time taken by selection sort
  - We like to know the dominant factor in T(n)
- $T(n) = 7/2*n^2+15n/2-10 <= 11*n^2$
- We claim that T(n) is in the order of n^2 or $O(n^2)$

Definition of big O

$$O(g(n)) = \{f(n) \mid (\exists c \in R^+, n_0 \in N)(\forall n \geq n_0)[0 \leq f(n) \leq cg(n)]\}$$

- Typically used for *asymptotic upper bound*
- Attention
  - O(f(n)) is a **set** of functions
- Pitfall
  - Traditionally we say $n^2 = O(n^2)$ as used in our text book
  - It really means $n^2 \in O(n^2)$

A graphical view of asymptotic definition



Example

- Prove that following statements

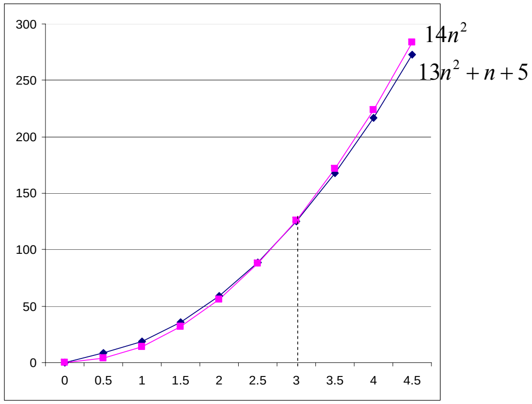
$$13n^2 + n + 5 \in O(n^2)$$

$$13n^2 + n + 5 \in O(n^2 \log n)$$

$$f(n) \in O(n) \rightarrow f^2(n) \in O(n^2)$$

$$O(n) \subset O(n^2)$$

$13n^2 + n + 5 \in O(n^2)$ $\forall n \geq 3, \quad 13n^2 + n + 5 \leq 14n^2$



What are c and n_0?

Several notations

- Logarithm time O(log n)
- Linear time O(n)
- Quadratic time $O(n^2)$
- Cubic time $O(n^3)$
- Exponential time $O(c^n)$, c>1

- Order of growth

$$O(\lg n) \subset O(n^c) \subset O(n^c \lg n) \subset O(n^{c+\varepsilon} \lg n) \subset O(d^n) \qquad c, \varepsilon > 0, d > 1$$

The Maximum rule

- Let $f, g : N \rightarrow R^{\geq 0}$,

  then $O(f(n) + g(n)) = O(\max(f(n), g(n)))$

- Proof
  - the key is max(f(n),g(n)) <= f(n)+g(n) <= 2*max(f(n),g(n))

- Examples
  - $O(12n^3-5n+n\log n+36)$
- The maximum rule let us ignore lower-order terms

Example

- True or false
  - ? $5 \in O(\log n)$
  - ? $\log n \in O(5)$
  - ? $O(n) \subset O(n^{0.6}\log n)$
  - ? $O(n^{0.6}\log n) \subset O(n)$
  - ? $O(n^8) = O((n^2-3n+5)^4)$

Definition of Ω

$$\Omega(g(n)) = \{f(n) \mid (\exists c \in R^+, n_0 \in N)(\forall n \geq n_0)[f(n) \geq cg(n) \geq 0]\}$$

- Ω is typically used to describe *asymptotic lower bound*
- Ω for algorithm complexity
  - We use it to give the lower bounds on the intrinsic difficulty of solving problems
  - Example, any comparison-based sorting algorithm takes time $\Omega(n\log n)$

The Θ notation

Definition:

$$\Theta(g(n)) = \{f(n) \mid (\exists c_1, c_2 \in R^+, n_0 \in N)(\forall n \geq n_0)[0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)]\}$$

Equivalent to: $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

- Used to describe *asymptotically tight bound*

The Limit Rule

- Let $f, g : N \rightarrow R^{\geq 0}$, then

1. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+$ then $f(n) \in \Theta(g(n))$
2. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) \in O(g(n))$ but $f(n) \notin \Omega(g(n))$
3. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$ then $f(n) \in \Omega(g(n))$ but $f(n) \notin O(g(n))$

Example

$$(n^c)' = cn^{c-1}$$

$$(\ln n)' = \frac{1}{n}$$ (ln n means $\log_e n$, the text use log)

When c>0

$$\lim_{n\to\infty} \frac{\ln n}{n^c} = \lim_{n\to\infty} \frac{(\ln n)'}{(n^c)'} = \lim_{n\to\infty} \frac{1/n}{cn^{c-1}} = \lim_{n\to\infty} \frac{1}{cn^c} = 0$$

$$\ln n \in O(n^c)$$ for any $c>0$

Semantics of big-O and Ω

- When we say an algorithm takes worst-case time $t(n) \in O(f(n))$, then there exist a real constant c such that $c*f(n)$ is an upper bound for any instances of size of sufficiently large n
- When we say an algorithm takes worst-case time $t(n) \in \Omega(f(n))$, then there exist a real constant d such that there exists at least one instance of size n whose execution time >= $d*f(n)$, for any sufficiently large n
- Example
  - Is it possible an algorithm takes worst-case time $O(n)$ and $\Omega(n\log n)$?

Practice Problems

```
anAlgorithm( int n)
{
  // if (x) is an elementary
  // operation
  if (x) {
    some work done
    by n^2 elementary
    operations;
  } else {
    some work done
    by n^3 elementary
    operations;
  }
}
```

- True or false
  - The algorithm takes time in $O(n^2)$
  - The algorithm takes time in $\Omega(n^2)$
  - The algorithm takes time in $O(n^3)$
  - The algorithm takes time in $\Omega(n^3)$
  - The algorithm takes time in $\Theta(n^3)$
  - The algorithm takes time in $\Theta(n^2)$
  - The algorithm takes worst case time in $O(n^3)$
  - The algorithm takes worst case time in $\Omega(n^3)$
  - The algorithm takes worst case time in $\Theta(n^3)$
  - The algorithm takes best case time in $\Omega(n^3)$

Definition of o and ω

- Definition

$$o(g(n)) = \{f(n) \mid (\forall c \in R^+, \exists n_0 \in N, \forall n \geq n_0)[0 \leq f(n) < cg(n)]\}$$

$$\omega(g(n)) = \{f(n) \mid (\forall c \in R^+, \exists n_0 \in N, \forall n \geq n_0)[f(n) > cg(n) \geq 0]\}$$

- Denote upper/lower bounds that are not asymptotically tight
- Example $1000n \in o(n^2)$; $1000n^2 \notin o(n^2)$
  $1000n^2 \in \omega(n)$; $1000n^2 \notin \omega(n^2)$
- Properties

$$f(n) \in o(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) \in \omega(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Relational Properties

- Transtivity: *O, o, Ω, ω,Θ*
- Reflexity: *O, Ω, Θ*
- Symmetry: $f(n) = \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$
- Transpose symmetry (Duality)

$$f(n) = O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$$
$$f(n) = o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$$

- Analogy

$$f(n) \in O(g(n)) \approx a \leq b$$
$$f(n) \in \Omega(g(n)) \approx a \geq b$$
$$f(n) \in \Theta(g(n)) \approx a = b$$
$$f(n) \in o(g(n)) \approx a < b$$
$$f(n) \in \omega(g(n)) \approx a > b$$