Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a language for talking about program behavior.
  - (e.g., by using big-O notation)
- In real life, many algorithms, though different from each other, fall into one of several paradigms (discussed shortly).
- These paradigms can be studied, and applied to new problems.

Why these particular algorithms??

- In this course, we will discuss problems, and algorithms for solving these problems.
- There are so many algorithms – why focus on the ones in the syllabus?
Why these algorithms (cont.)

1. **Main paradigms:**
   a) Greedy algorithms
   b) Divide-and-Conquer
   c) Dynamic programming
   d) Branch-and-Bound (mostly in AI)
   e) Etc etc.

2. **Other reasons:**
   a) Relevance to many areas:
      • E.g., networking, internet, search engines…
   b) Coolness
   c) Demonstrate important non-trivial ideas that could be used in many other algorithms.
      E.g., Randomized algorithms

---

Criteria that I personally used for topics selection

1. Material that is commonly appearing in this course
2. Materials that are a “must know” after algorithm course. (e.g. Dijkstra Network Flow)
3. Demonstrating very powerful techniques that I found ingenious (e.g. randomness, closest-pair, SkillList).

---

Basic terminology needed from probability

- Random Variables
- Expectation (expected value of a random variable)
- Sum of expectations: \( E(X+Y) = E(X) + E(Y) \)
- We will spend a quick review of these terms.

---

Next – merge sorting (CLRS)

Good occasion to go over merge sort, and to practice basic asymptotic time complexity notations
The problem of sorting

**Input**: sequence \( <a_1, a_2, \ldots, a_n> \) of numbers.

**Output**: permutation \( <a'_1, a'_2, \ldots, a'_n> \) such that \( a'_1 \leq a'_2 \leq \cdots \leq a'_n \).

Example:

**Input**: 8 2 4 9 3 6
**Output**: 2 3 4 6 8 9

---

**Insertion sort**

Invariants: \( A[1 \ldots i-1] \) is sorted

---

Consider \( A[i] = 9 \). Not in the correct place. Need to make room for 9. We shift all elements right, starting from 10.

---

**Example of insertion sort**

8 2 4 9 3 6
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 4 8 9 6 3
**Example of insertion sort**

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

**Example of insertion sort**

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
done

---

**Running time**

- The running time depends on the input: an already sorted sequence is easier to sort.
  - Parameterize the running time by the size of the input $n$.
  - Seek upper bounds on the running time $T(n)$ for the input size $n$, because everybody likes a guarantee.

---

**Kinds of analyses**

**Worst-case:** (usually)
- $T(n) =$ maximum time of algorithm on any input of size $n$.

**Average-case:** (sometimes)
- $T(n) =$ expected time of algorithm over all inputs of size $n$.
- Need assumption of statistical distribution of inputs.

**Best-case:** (bogus)
- Cheat with a slow algorithm that works fast on some input.
**Machine-independent time**

*What is insertion sort’s worst-case time?*
- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

**Big Idea:**
- Ignore machine-dependent constants.
- Look at growth of $T(n)$ as $n \to \infty$.

“*Asymptotic Analysis*”

---

**$\Omega$-notation**

**Math:**
we say that $T(n) = \Omega(g(n))$ iff there exists positive constants $c_2$, and $n_0$ such that
$0 \leq c_2 g(n) \leq T(n)$ for all $n \geq n_0$

**Engineering:**
- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Omega(n^3)$

---

**$O$-notation**

**Math:**
we say that $T(n) = O(g(n))$ iff
there exists positive constants $c_1$, and $n_0$ such that
$0 \leq T(n) \leq c_1 g(n)$ for all $n \geq n_0$

Usually $T(n)$ is running time, and $n$ is size of input

**Engineering:**
- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = O(n^3)$

---

**$O$-notation - cont**

So if $T(n) = O(n^4)$ then we are also sure that
$T(n) = O(n^4)$ and that
$T(n) = O(n^{4.5})$ and
$T(n) = O(2^n)$

But it might or might not be true that $T(n) = O(n^{1.5})$.

However, if $T(n) = \Omega(n^2)$ then it is not true that
$T(n) = O(n^{1.5})$

Proofs: In the Homework
**Θ-notation**

**Math:**
we say that \( T(n) = \Theta(g(n)) \) if there exist positive constants \( c_1, c_2, \) and \( n_0 \) such that
\[ 0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n) \quad \text{for all } n \geq n_0 \]

In other words
\( T(n) = \Theta(g(n)) \) \iff \( T(n) = O(g(n)) \) and \( T(n) = \Omega(g(n)) \)

**Engineering:**
• Drop low-order terms; ignore leading constants.
• Example: \( 3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3) \)

---

**Insertion sort analysis**

**Worst case:** Input reverse sorted.
\[
T(n) = 2c + 3c + 4c + \ldots + c(n-1) = cn(n-1)/2
\]

\[
T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \quad \text{[arithmetic series]}
\]

**Is insertion sort a fast sorting algorithm?**
• Moderately so, for small \( n \).
• Not at all, for large \( n \).

---

**Merge sort**
*(divide-and-conquer algorithm)*

**MERGE-SORT \([1 \ldots n]\)**
1. If \( n = 1 \), done.
2. Recursively sort \( A[1 \ldots [n/2]] \)
   and \( A[[n/2]+1 \ldots n] \).
3. “\textbf{Merge}” the 2 sorted lists.

**Key subroutine:** \textbf{Merge}

**Asymptotic performance**

When \( n \) gets large enough, a \( \Theta(n^2) \) algorithm \textit{always} beats a \( \Theta(n^3) \) algorithm.

• We shouldn’t ignore asymptotically slower algorithms, however.
• Real-world design situations often call for a careful balancing of engineering objectives.
• Asymptotic analysis is a useful tool to help to structure our thinking.
Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2

Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2

Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2

Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2
Merging two sorted arrays

10

20 12 20 12 20 12
13 11 13 11 13 11
7 9 7 9 7 9
2 1 2 2
1 2 7

Merging two sorted arrays

20 12 20 12 20 12
13 11 13 11 13 11
7 9 7 9 7 9
2 1 2 7
1 2 7
9

Merging two sorted arrays

20 12 20 12 20 12
13 11 13 11 13 11
7 9 7 9 7 9
2 1 2 7
1 2 7
9

Merging two sorted arrays

20 12 20 12 20 12
13 11 13 11 13 11
7 9 7 9 7 9
2 1 2 7
1 2 7
9
Merging two sorted arrays
Merging two sorted arrays

Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).

Analyzing merge sort

$T(n) = \Theta(1)$ if $n = 1$;
$2T(n/2) + \Theta(n)$ if $n > 1$.

Recurrence for merge sort

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

Abuse

Sloppiness: Should be $T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$, but it turns out not to matter asymptotically.

CLRS provides several ways to find a good bound on $T(n)$. 
Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log_2 n \cdot \frac{cn}{4}$

$\Theta(1)$

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log_2 n \cdot \frac{cn}{4}$

$\Theta(1)$

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log_2 n \cdot \frac{cn}{4}$

$\Theta(1)$

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log_2 n \cdot \frac{cn}{4}$

$\Theta(1)$
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log n_{cn/4}$

$\Theta(1) \cdot \ldots \cdot \Theta(n)$

#leaves $= n$

Total $= \Theta(n \log n)$

Conclusions

- $\Theta(n \log n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.
- Go test it out for yourself!

More examples – iterative method (1)

```cpp
NoNeed(n){
    • If (n<1) return ;
    • Print(*)
    • NoNeed(n-1)
}
```

Recursion formula: $T(n) = c + T(n-1)$, where $T(1) = c$. We can solve it using the iteration method:

$T(n) = c + T(n-1) =
\ldots
(\text{setting } k = n-1) \ldots
(n-1)c + T(1) = nc$
Examples

Example 1

```c
NoNeed(n) {
    if (n<1) return ;
    Print('*')
    NoNeed(n-1)
}
```

Recursion formula: \( T(n) = c + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = c + T(n-1) =
\]

\[
c + (c + T(n-2)) = 2c + T(n-3) = \ldots = (\text{pick } k < n)
\]

\[
k - 1 + c + T(n-k) = (\text{setting } k = n-1) \ldots
\]

\[
(n-1)c + T(1) = nc
\]

Example 2

```c
NoNeed(n) {
    if (n<1) return ;
    for(i=1 ; i<n ; i++) print('*')
    NoNeed(n-1)
}
```

Recursion formula: \( T(n) = cn + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = cn + T(n-1) =
\]

\[
= c(n + T(n-2)) + T(n-3) = \ldots = (\text{pick } k < n)
\]

\[
= c(n + n-1 + n-2 + ... + n-3 + ... + n-k) + T(n-k-1) =
\]

\[
= \ldots = (\text{setting } k = n-1) \ldots
\]

\[
c(n + n-1 + n-2 + n-3 + ... + 1) + T(1) =
\]

\[
c(1 + 2 + 3 + ... + n) + T(1) = cn(n+1)/2 = \mathcal{O}(n^2).
\]

Example 2

```c
NoNeed(n) {
    if (n<1) return ;
    for(i=1 ; i<n ; i++) print('*')
    NoNeed(n-1)
}
```

Recursion formula: \( T(n) = cn + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = cn + T(n-1) =
\]

\[
= c(n + T(n-2)) + T(n-3) = \ldots = (\text{pick } k < n)
\]

\[
= c(n + n-1 + n-2 + ... + n-3 + ... + n-k) + T(n-k-1) =
\]

\[
= \ldots = (\text{setting } k = n-1) \ldots
\]

\[
c(n + n-1 + n-2 + n-3 + ... + 1) + T(1) =
\]

\[
c(1 + 2 + 3 + ... + n) + T(1) = cn(n+1)/2 = \mathcal{O}(n^2).
\]

Example 3

- Read(n); \( k = 1 \);
- while( \( k \leq n \) ) \( k = 2k \);
- We know that each iteration takes \( O(1) \) times. Need to find the number of iterations.
  - After the first iteration \( k = 2^1 \)
  - After the 2nd iteration \( k = 2^2 \)
  - After the 3rd iteration \( k = 2^3 \)
  - After the \( j \)th iteration \( k = 2^j \)

\[
\text{Assume } j \text{ iterations occur until the loop exits. After the last one we have that } k = 2^j < 2n.
\]

- Taking \( \log_2 \) from both sides, we have that \( \log_2 k = \log_2(2^j) < \log_2(2n) \) or...
- \( j \log_2 2 < \log_2(2n) \) or \( j \log_2 n \) or \( j = \mathcal{O}(\log_2 n) \) or \( T(n) = \mathcal{O}(\log n) \)

- Homework: Prove \( T(n) = \mathcal{O}(\log n) \)
Examples 4

```
read(n);
for(i=1; i < n ; i++)
for( j=i ; j < n ; j+=i )
print("*");
```

- Time Complexity Analysis – first approach:
  - The outer loop (on i) runs exactly n-1 times
  - The inner loop (on j) runs O(n) times.
  - Together T(n)=O(n^2).

- More "sensitive" analysis:
  - For i=1 we run through j=1,2,3,4,...n, total n times.
  - For i=2 we run through j=2,4,6,8,10,...n, total n/2 times.
  - For i=3 we run through j=3,6,9,12,...n, total n/3 times.
  - For i=n we run through j=n, total n times.
  - Summing up: T(n)=n+n/2+n/3+n/4+...n/n = n(1+1/2+1/3+1/4+...1/n) = n \ln n in n
  - Harmonic Sum

Example 5

```
read(n) ;  a=0.31415926
while( n>1 )
  For(j=1; j<n ; j++)
    print("*"))
  n=a*n ;
```

- The first time the outer loop is called, the "print" is called n times.
- The second time the outer loop is called, the "print" is called an times.
- The third time the outer loop is called, the "print" is called a^2n times...

- Let t be the number of iterations of the outer loop. Then the total time
  T(n)=n+an+a^2n+...a^n = n(1+a+a^2+...a^t) < n(1+a+a^2+...a^t) = n/(1-a) = O(n).

- Same analysis holds for any a<1
  
  Recall: \(1+a+a^2+...+a^t = (1-a^{t+1})/(1-a)\).
  
  If a<1 then \(1+a+a^2+...+a^t = 1/(1-a)\)

Properties of big-O

- Claim: if \(T_1(n)=O(g_1(n))\) and \(T_2(n)=O(g_2(n))\) then
  \(T_1(n)+T_2(n)=O(g_1(n)+g_2(n))\)

- Example: \(T_1(n)=O(n^2)\), \(T_2(n)=O(n \log n)\) then
  \(T_1(n)+T_2(n)=O(n^2 + n \log n) = O(n^2)\)

- Proof: We know that there are constants \(n_1, n_2, c_1, c_2\) st.
  - for every \(n > n_1\) \(T_1(n) < c_1 g_1(n)\) (definition of big-O)
  - for every \(n > n_2\) \(T_2(n) < c_2 g_2(n)\) (definition of big-O)

  - Now set \(n = \max \{ n_1, n_2 \}\), and \(c = c_1 + c_2\) then
    - for every \(n > n\) we have that
      - \(T_1(n) + T_2(n) < c_1 g_1(n) + c_2 g_2(n) \leq c' g(n) + c' g(n) = c' (g_1(n) + g_2(n))\)

More properties of big-O

- Claim: if \(T_1(n)=O(g_1(n))\) and \(T_2(n)=O(g_2(n))\) then
  \(T_1(n) T_2(n)=O(g_1(n) g_2(n))\)

- Example: \(T_1(n)=O(n^2)\), \(T_2(n)=O(n \log n)\) then
  \(T_1(n) T_2(n)=O(n^2 \log n)\)

- Similar properties hold for \(\Theta, \Omega\)
More about $\Omega(\cdot)$

Sometimes we would talk about a lower bound on the running time of a specific algorithm.

E.g. The insertion sort might take $\Omega(n^2)$ for some input.

Sometimes we would talk about a lower bound on the running time of a problem.

E.g.

1. Any algorithms that reads all the input (for any problem) requires $\Omega(n)$ time.
2. Any algorithm that stores all the data requires $\Omega(n)$ space.
3. Any algorithm that sort $n$ keys requires $\Omega(n \log n)$
   (disclaimer – could be better if we make some assumptions about the keys or the model. Usually
   * Sorting sort integers takes $\Omega(n)$ (how?)
   * Sorting floats takes $\Omega(n \log n)$

Example 2 (another look)

```c
NoNeed(n) {
  if (n<1) return ;
  for (i=1 ; i<n ; i++)   print(*)
  NoNeed(n-1)
}
```

Recursion formula: $T(n)=cn+T(n-1)$, where $T(1)=c$. We can solve it using the iteration method:

$$T(n)=cn+T(n-1)=cn+(c(n-1)+T(n-2)) = cn+(c(n-1)+(c(n-2)+T(n-3)) = \ldots = (\text{pick } k<n)$$

$$=c[n+n-1+n-2+n-3+\ldots+n-k]+T(n-k-1) = \ldots = c[n+n-1+n-2+n-3+\ldots+1]+T(1)= cn(n+1)/2 = \Omega(n^2).$$

The lower bound trick – Second example

We are about to insert $n$ keys $\{k_1, \ldots, k_n\}$ into an empty AVL tree.

How much time would it take?

Upper bound: When the $i+1$ key is inserted, the tree contains $i$ keys, so its height is $O(\log i)$, and an insert operation takes $O(\log i)$ which is also $O(\log n)$

So the overall running time is 

$$O(\log 1) + O(\log 2) + O(\log 3) + \ldots + O(\log n) \leq O(\log n) + O(\log n) + O(\log n) + \ldots + O(\log n) = O(n \log n)$$

This is an upper bound. What is the lower bound?

- $\Omega(n)$ ?
- $\Omega(n+1)$ ?
- $\Omega(2^n)$ ?
- $\Omega(n \log n)$
The lower bound trick – a less trivial example

We demonstrate this trick by giving an $\Omega(n \log n)$ bound on the time $T(n)$ required to insert $n$ keys into an (initially empty) balanced search tree.

The $i$’th insertion takes $K \log(i)$ time (for a constant $K$, that we ignore). Hence

$$\sum_{i=1}^n \log i =$$

$$\log 1 + \log 2 + \cdots + \log\left(\frac{n}{2} - 1\right) + \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2} + 1\right) + \cdots + \log n$$

$$\geq \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2} + 1\right) + \log\left(\frac{n}{2} + 2\right) + \cdots + \log n$$

$$\geq \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}\right) + \cdots + \log\left(\frac{n}{2}\right) =$$

$$= (\log\left(\frac{n}{2}\right)) \cdot \log(n \log n)$$

Random Variable (light version)

- Assume we perform an experiment (tossing a dice). Let $R$ be the result – one of the number 1,2,3,4,5,6.
- We could define a random variable which (in this course) is a value that depends on the result of the experiment.
- Preferably, set to ‘1’ if some condition is satisfied, and is ‘0’ otherwise.
- Define $X$ to be a random variable, set to 1 iff $R$ is even; ($X=0$ if $R$ is 1,3, or 5).
- Define $Y$ to be another random variable, which is 1 iff $R \geq 2$.
- We could ask what is the probability that $X=1$. Denote $Pr(X=1)$.
- If dice is fair, $Pr(X=1)$ is 0.5, and $Pr(Y=1)=4/6=0.666$.

Random Variable and expectation (light version)

- In many cases, we would like to know what is the expected value of a random var.
- Example: If $Y=1$ we earn a dollar. What is the expected amount we earn in one game.
- Good news (for Boolean vars): $E(Y)$, the expected value of $Y$, is just $Pr(Y=1)$.
- What if we earn $17$ if $Y=1$.
- Lemma: for any constant $a$ it is always true that $E(aY) = aE(Y)=aPr(Y=1)$.

Random Variable and expectation (light version)

- If a random variable could accept several values?

Say define a new random var $Z=X+Y$.

So $Z=0$ if $X=0$ and $Y=0$;

$Z=1$ if $X=1$ and $Y=1$;

etc.

Lets assume that we earn $1$ if $X=1$, and $1$ if $Y=1$. The expected earning is

$$E(Z) = \sum_{j=0}^{\infty} j \cdot Pr(Z = j)$$

But in out setting $E(Z) = 0 \cdot Pr(Z=0) + 1 \cdot Pr(Z=1) + 2Pr(Z=2)$

Lemma: $E(X+Y)=E(X)+E(Y)$

So in out case $E(Z)=0 \cdot 0.5 + 1 \cdot 0.666 + 2 \cdot 0.666$

Lemma for any constants: if $E(aX+bY)=aE(X)+E(bY)$

$E(X+Y+Z)=E(X)+E(Y)+E(Z)$
Analysis of QuickSort

• On whiteboard