CSC 240 LECTURE 8

ANALYSIS OF ALGORITHMS

Why might we want to know how fast an algorithm runs?
-to estimate how long the algorithm will run on a given input
 or the amount of resources it needs
-to estimate how large an input it is reasonable to give the algorithm
-to compare the efficiency of different algorithms that solve the same
 problem

The actual running time of an algorithm depends on a number of factors, such as:
the quality of the implementation,
the compiler that is used,
what machine it is implemented on, and
how heavily loaded the machine is.

Thus, when we analyze an algorithm, we can only estimate the running time to within a constant factor.

The running time of an algorithm also depends on the particular input on which it is run.
For an algorithm A, let $t_A(I)$ denote the number of steps the algorithm performs on input I.

What is a step?

examples: assignment, array index, arithmetic operation, comparison, return from a function

Choose one or two operations such that the total number of operations performed by the algorithm you are analyzing is the same as the number of these operations performed, to within a constant factor.

Example
```
LinearSearch(L,x)
i←1
while i ≤ length(L) do
    if L[i] = x then return i
    i ← i+1
end while
return 0
```

What is a good choice for step?
comparison with x

On input L = [2,4,6,8] and x = 2, 1 step is performed.
On input L = [2,4,6,8] and x = 4, 2 steps are performed.
On input L = [2,4,6,8] and x = 8, 4 steps are performed.
On input L = [2,4,6,8] and x = 1, 4 steps are performed.

The running time of an algorithm A usually increases as the size of its input gets larger. Thus, we use a function of the input size n to represent the number of steps it performs on inputs of size n.

Problem: on different inputs of the same size, the algorithm can perform different numbers of steps.

worst case time complexity

$T_A : \mathbb{N} \rightarrow \mathbb{N}$, where $T_A(n) = \max \{t_A(I) \mid I$ is an input to A of size $n\}$.

Example:

When A is LinearSearch and the size of input (L,x) is the length of L, $T_A(n) = n$.

average case time complexity

$T'_A : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$, where $T'_A(n) = E[t_A]$ and

the expectation is taken over a probability space of all inputs of size n.

If all inputs are equally likely, then

$$T'_A(n) = \frac{\sum\{t_P(I) \mid size(I) = n\}}{\#\{I \mid size(I) = n\}}.$$

Like the choice of which steps to count,
input size can depend on the algorithm.

- For an algorithm with a list as input, the input size is usually the length of the list.
- For an algorithm with an integer as input, the input size can be the absolute value of the integer or the number of bits in its binary representation.
- For an algorithm with an m x n matrix as input, the input size can be m x n, the number of elements in the matrix or we can use both m and n to describe the input size.
- For an algorithm with a graph as input, the input size is either the number of vertices, n, the the number of edges, m, or both.

The worst time complexity $T_A(n,m)$ can be a function of two parameters.

Let $u: \mathbb{N} \to \mathbb{N}$.
Algorithm A has worst-case time complexity at most u means $T_A \leq u$.

Let $\mathcal{S}$ denote the set of all inputs to algorithm A.

$\forall n \in \mathbb{N}.(\max \{t_A(I) \mid (I \in \mathcal{S}) \text{ AND } (\text{size}(I) = n)\} \leq u(n))$.

Express this without using max:

$\forall n \in \mathbb{N}. \forall I \in \mathcal{S}. [(\text{size}(I) = n) \text{ IMPLIES } (t_A(I) \leq u(n))]$

$\forall I \in \mathcal{S}. (t_A(I) \leq u(\text{size}(I)))$

To prove that $T_A \leq u$, you must show that for all $n \in \mathbb{N}$ and for all inputs I to A of size n, algorithm A takes at most u(n) steps on input I.

When comparing two algorithms with different rates of growth, constant factors and low order terms may matter when
the problem size is small,
when many instances will be solved, or
when fast response time is essential (for example, in real-time systems).
They do not usually matter when the problem size is large.

Since we only determine the number of operations performed to within a constant factor, we often use O notation to express an upper bound on the time complexity of an algorithm.

It is often much easier mathematically to do an analysis to within a constant factor than to do an exact analysis.

Recall that

$O(f) = \{g \in \mathcal{F} \mid \exists c \in \mathbb{R}^+. \exists b \in \mathbb{N}. \forall n \in \mathbb{N}. [(n \geq b) \text{ IMPLIES } (g(n) \leq c \cdot f(n))]\}$.

Express $T_A \in O(u)$ using predicate logic:

$\exists c \in \mathbb{R}^+. \exists b \in \mathbb{N}. \forall n \in \mathbb{N}. [(n \geq b) \text{ IMPLIES } (T_A(n) \leq c \cdot u(n))]$

Alternatively, without using T_A :

$\exists c \in \mathbb{R}^+. \exists b \in \mathbb{N}. \forall n \in \mathbb{N}. [(n \geq b) \text{ IMPLIES } \forall I \in \mathcal{S}. [(\text{size}(I) = n) \text{ IMPLIES } (t_A(I) \leq c \cdot u(n))]]$

$\exists c \in \mathbb{R}^+. \exists b \in \mathbb{N}. \forall I \in \mathcal{S}. [(\text{size}(I) \geq b) \text{ IMPLIES } (t_A(I) \leq c \cdot u(\text{size}(I)))]$

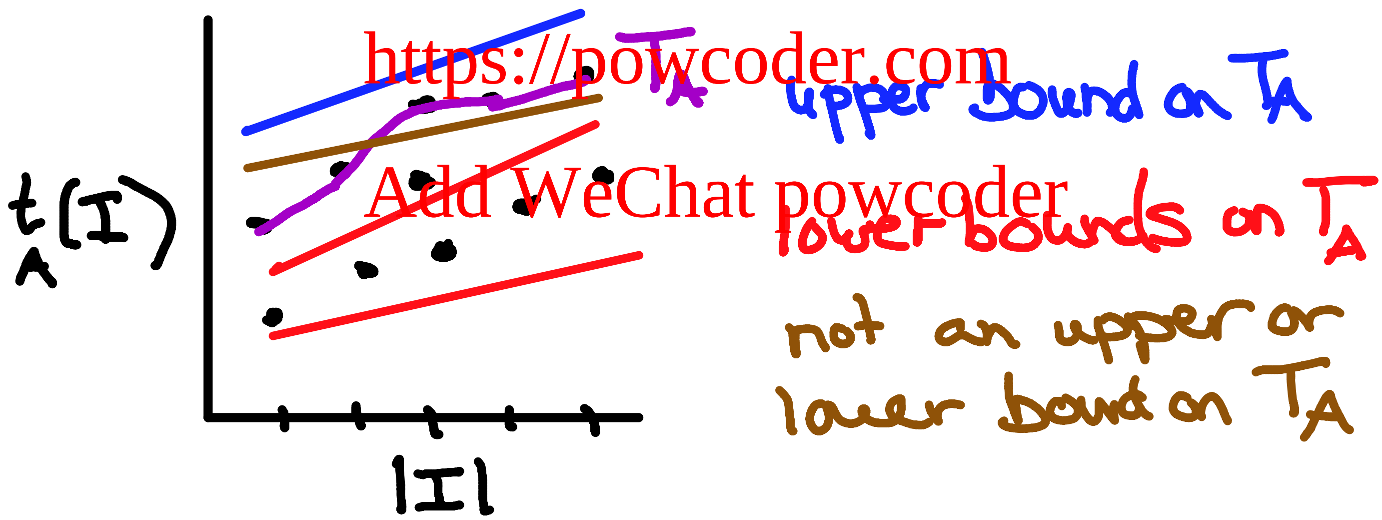
If someone proves $T_A \le u$, it does not mean that the algorithm ever takes that long. The actual running time might be much less.
To know what the worst case time complexity of algorithm A is, we also need to prove a lower bound on T_A.

Let $\ell: \mathbb{N} \to \mathbb{N}$.

Algorithm A has worst-case time complexity at least ℓ means $T_A \ge \ell$.
Express this using predicate logic:
Let $\mathcal{S}$ denote the set of all inputs to algorithm A.
$\forall$ n $\in \mathbb{N}$.(max {t_A(I) I (I $\in \mathcal{S}$) AND (size(I) = n)} $\ge \ell$(n)).
Express this without using max:
$\forall$ n $\in \mathbb{N}$. $\exists$ I $\in \mathcal{S}$. [(size(I) = n) AND (t_A(I) $\ge \ell$(n))]

$\exists$ I $\in \mathcal{S}$. (t_A(I) $\ge \ell$(size(I))) IS INCORRECT

To prove that $T_A \ge \ell$, for all n $\in \mathbb{N}$, you must find an input I to A of size n, and show that algorithm A takes at least ℓ(n) steps on input I.



Bigger lower bounds are better!

Recall that
Ω(f) = {g $\in \mathcal{F}$ I $\exists$ c $\in \mathbb{R}^+$. $\exists$ b $\in \mathbb{N}$. $\forall$ n $\in \mathbb{N}$. [(n $\ge$ b) IMPLIES (g(n) $\ge$ c f(n))]}
Express $T_A \in \Omega(\ell)$ using predicate logic:

$\exists$ c $\in \mathbb{R}^+$. $\exists$ b $\in \mathbb{N}$. $\forall$ n $\in \mathbb{N}$. [(n $\ge$ b) IMPLIES (T_A(n) $\ge$ c $\cdot$ ℓ(n))]

$\exists$ c $\in \mathbb{R}^+$. $\exists$ b $\in \mathbb{N}$. $\forall$ n $\in \mathbb{N}$. [(n $\ge$ b) IMPLIES $\exists$ I $\in \mathcal{S}$. [(size(I) = n) AND
(t_A(I) $\ge$ c $\cdot$ ℓ(n))]]

ANALYZING THE WORST CASE TIME COMPLEXITY OF ITERATIVE ALGORITHMS

Code without loops or procedure calls takes O(1) time.

Loops
If a loop is executed O(f(n)) times and each iteration takes O(g(n)) steps, then the entire loop takes O(f (n) · g(n)) steps.

If Statements
Suppose A, B, and C are blocks of statements that take O(f(n)), O(g(n)), and O(h(n)) steps, respectively. Then
if C
   then A
   else B
takes O(h(n)+ max{f(n),g(n)}) steps.

Procedure and Function Calls
Suppose the worst case time complexity of a procedure (or function) P with input size r is T(r) and suppose T(r) ∈ O(f(r)).
Then a call to P with an input of size g(n) takes O(f(g(n))) steps.

Worst Case Time Complexity of Merging Two Sorted Lists

```
i ← 1
j ← 1
k ← 1
while i ≤ length(A) and j ≤ length(B) do
         if A[i] ≤ B[j]
         then C[k] ←A[i]
                 i ← i + 1
         else C[k] ←B[j]
                 j ← j + 1
         k ← k + 1
if i > length(A)
then while j ≤ length(B) do
         C[k] ←B[j]
          j ← j + 1
         k ← k + 1
else while i ≤ length(A) do
         C[k] ←A[i]
          i ← i + 1
         k ← k + 1
```

What is a good choice for input size?
m = length(A) and n = length(B)

What is a good choice for a step?

- number of assignments to C

What is the worst case time complexity?m+n

- number of comparisons between elements of A and elements of B

What is the worst case time complexity?
min{m,n} X
m+n X
m+n-1 √

This is the number of times the while loop on line 4 is performed.
It terminates as soon as $i > m$ or $j > n$.
So either i has been incremented m times or j has been incremented n times.
Note that exactly one of i and j is incremented each iteration,
so it is not possible for both these conditions to be true.

Thus, in the first case, j has been incremented at most n-1 times and,
in the second case, i has been incremented at most m-1 times.
In both cases, at most m+n-1 comparisons between elements of A and elements of B are performed.

$T_A(m,n) \leq m+n-1$

Why is m+n-1 a lower bound?
For each m and n, what is an input in which m+n-1 comparisons are performed?

A is a list of length m, B is a list of length n and the last element of each list is larger than any other elements in the two lists.

Worst case time complexity of Insertion Sort

```
i ← 1
while i ≤ length(A) do
        j ← i
        while j > 1 and  A[i] < B[j-1] do
                B[j] ←B[j-1]
                j ← j - 1
        B[j] ←A[i]
        i ← i + 1
```

What is a good choice for input size?
n = length(A)

What is a good choice for a step?
- number of assignments
- number of comparisons

What is the worst case time complexity we count both?

Let $T_{IS} : \mathbb{N} \to \mathbb{N}$ be such that
$T_{IS}(n)$ = maximum number of steps taken by Insertion Sort on arrays of length n.

Lemma 1 $T_{IS}(n) \leq 2n^2 + 4n + 2 \in O(n^2)$.

Proof: Let $n \in \mathbb{N}$ be arbitrary and let A be an arbitrary array of length n.

There are exactly n complete iterations of the outer while loop.

Each complete iteration of the outer while loop consists of 4 steps (a comparison on line 2 and assignments on lines 3,7, and 8) plus an execution of the inner while loop.

Each complete iteration of the inner while loop, at most 4 steps are performed (2 comparisons on line 4 and assignments on lines 5 and 6).

During iteration i of the outer loop, at most i – 1 complete iterations of the inner while loop are performed.

The final (incomplete) iteration of the inner while loop takes at most 2 steps (2 comparisons on line 4).

Thus the total number of steps taken by the n complete iterations of the outer while loop is at most

$$\begin{aligned} \sum \{4(i-1) + 2 + 4 \mid 1 \leq i \leq n\} &= 6n + 4 \sum \{k \mid 0 \leq k \leq n-1\} \\ &= 6n + 4(n-1)n/2 \\ &= 2n^2 + 4n. \end{aligned}$$

The final (incomplete) iteration of the outer while loop takes 1 step (line 2).
There is 1 step taken before the outer while loop (line 1).

Thus $T_{IS}(n) \leq 2n^2 + 4n + 2$, since A is an arbitrary array of length n.
Since $n \in \mathbb{N}$ is arbitrary, it follows that, $\forall n \in \mathbb{N} . T_{IS}(n) \leq 2n^2 + 4n + 2$

so $T_{IS} \in O(n^2)$.

Lemma 2 $T_{IS} \in \Omega(n^2)$.

Proof: Let $n \in \mathbb{N}$ be arbitrary.
Consider the input A = [n,n − 1,...,1] of size n.

Let P(k) =
"during iteration k of the outer while loop, 4k + 1 steps are performed, and
after iteration k of the outer while loop,
and
the first k elements of B are n−k+1,...,n".

We will show by induction that P(k) is true for $1 \leq k \leq n$.

During iteration 1 of the outer while loop, 5 steps are performed:
1 step on each of lines 2,3,4,7, and 8.
Afterwards, B[1] = A[1] = n.
Thus, P(1) is true.


```
1   i ← 1
2   while i ≤ length(A) do
3           j ← i
4           while j > 1 and  A[i] < B[j-1] do
5                   B[j] ← B[j-1]
6                   j ← j - 1
7           B[j] ← A[i]
8           i ← i + 1
```


Let $1 \leq k < n$ and suppose that P(k) is true.

Then, during iteration k+1 of the outer while loop,
4 steps are performed (on lines 2, 3, 7 and 8),
in addition to the inner while loop.

Note that i = k+1 and A[i] = n − k.
On line 3, j is set to k+1
and it is decremented during each complete iteration of the inner loop.
Since A[i] is less than the first k elements of B,
there are k complete iterations of the inner while loop.
Each complete iteration of the inner while loop consists of 4 steps
(2 on line 4, 1 on line 5 and 1 on line 6).

In the final (incomplete) iteration of the inner while loop, only 1 step is

performed (j =1).
Then, on line 7, A[i] = n-k becomes the new first element of B, so the first k+1 elements of B are n-k,..., n.

In total, $4k + 5 = 4(k+1) + 1$ steps are performed in iteration k+1 of the outer while loop.

Thus P(k+1) is true.

By induction $\forall\, k \in \{1,...,n\}.P(k)$

Let P(k) =
"during iteration k of the outer while loop, 4k + 1 steps are performed, and after iteration k of the outer while loop,
and the first k elements of B are n–k+1,...,n".

The final (incomplete) iteration of the outer while loop takes 1 step (line 2).
There is also 1 step taken before the outer while loop (line 1).
Thus

$$T_{IS}(n) \geq 1 + 1 + \sum \{4k + 1 \mid 1 \leq k \leq n\} = 2 + n + 4 \sum \{k \mid 1 \leq k \leq n\}$$
$$= 2 + n + 4n(n+1)/2$$
$$= 2n^2 + 3n + 2.$$

Since $n \in \mathbb{N}$ is arbitrary, it follows that,
$\forall n \in \mathbb{N}\ .\ (T_{IS}(n) \geq 2n^2 + 3n + 2)$
Hence $T_{IS} \in \Omega(n^2)$.

Corollary $T_{IS} \in \Theta(n^2)$.

ANALYZING THE WORST CASE TIME OF RECURSIVE ALGORITHMS

```
function square(n)
if n = 1
then return n
else return 2 x n - 1 + square(n-1)
```

What is a good choice for input size? n

What is a good choice for a step?
any arithmetic operation

Define $T_{SQ} : \mathbb{Z}^+ \rightarrow \mathbb{N}$, where $T_{SQ}(n)$ denotes the number of arithmetic operations performed by square(n). Then

$$T_{SQ}(n) = \begin{cases} 0 \text{ if } n = 1 \\ 4 + T_{SQ}(n-1) \text{ if } n > 1. \end{cases}$$

The solution to this recurrence is $T_{SQ}(n) = 4(n-1)$ for $n \geq 1$.

MERGESORT(A)
if length(A) = 1 then return
divide A into two subarrays A' and A'' whose sizes differ by at most 1
MERGESORT(A')
MERGESORT(A'')
A ← MERGE(A', A'')

What is a good choice for input size? n = length(A)

What is a good choice for a step?

- number of comparisons between elements of A
- all comparisons and assignments

Let M denote the worst case time complexity of MERGESORT.
so $M : \mathbb{Z}^+ \rightarrow \mathbb{N}$ and
M(n) is the maximum number of steps taken by MERGESORT(A) for arrays A of size n.

Then

$$M(n) \leq \begin{cases} c \text{ if } n = 1 \\ M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor) + dn \end{cases}$$

where $c, d \in \mathbb{N}$ are constants.

complexity measure 1, c = 0.
complexity measure 2, c = 1.

complexity measure 1, d = 1, since $\lceil n/2 \rceil + \lfloor n/2 \rfloor - 1 = n-1$
complexity measure 2, $d \in \mathbb{Z}^+$

The arrays A' and A'' have sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.
Then first two terms are for the steps taken by MERGESORT(A') and MERGESORT(A'').
The third term is for dividing A into two subarrays
merging the two halves of the array.

The solution to this recurrence is $M(n) \in O(n \log n)$, by the Master Method.

Analysis of Recursive Binary Search

```
RecBinSearch(A, F, L, x)
1 if F = L
2 then if A[F] = x
3       then return F
4       else return 0
5 else m ← (F + L) div 2
6       if A[m] ≥ x
7       then RecBinSearch(A, F, m, x)
8       else RecBinSearch(A, m + 1, L, x)
```

What is a good choice for input size?
n = length(A) X
n = L– F+1√

What is a good choice for a step?

- number of comparisons between x and elements of A

Let $B : \mathbb{Z}^+ \rightarrow \mathbb{N}$,

where $B(n)$ denotes the worst case number of comparisons with x performed by RecBinSearch(A, F, L, x) when $L - F + 1 = n$, over all choices of A, F, L, and x such that $L - F + 1 = n$.

Then, from the code,
$B(1) = 1$ (line 2)
$B(n) \leq 1 + \max\{B(\lfloor n/2 \rfloor), B(\lceil n/2 \rceil)\}$, for $n > 1$ (line 5 and one of lines 7,8)
Note that, if B is a nondecreasing function,
then $\max\{B(\lfloor n/2 \rfloor), B(\lceil n/2 \rceil)\} = B(\lceil n/2 \rceil)$,
so $B(n) \leq 1 + B(\lceil n/2 \rceil)$ for $n > 1$.

The solution to this recurrence is $B(n) \in O(\log n)$, by the Master Method.

When n is a power of 2, $B(n) = 1 + B(n/2)$,
so the solution gives a lower bound as well.