3.2 big-O and sorting, lecture 2

Thursday, September 15, 2022

Reminders:

- test 1 in lab next week
- if you have an accommodations letter, inform your instructor!
- git add, git commit, git push

TODAY: theoretical analysis
- versions 1, 2, 3 of is_sorted

big O definition
- big O proofs
- sorting algorithms
  - selection sort
  - merge sort

Version 1 of is_sorted   size $= n$

i loop goes from $i=0$ to $i=$ size -1
  j loop goes from $j=i+1$ to $j=$ size -1
    comparison

How many comparisons are done in total?

$$\underbrace{(n-1)}_{1^{st}\text{ iteration}} + \underbrace{(n-2)}_{2^{nd}\text{ iter.}} + \underbrace{(n-3)}_{3^{rd}\text{ iter}} + \cdots + 3 + 2 + 1 = \text{total}$$

$$1 + 2 + 3 + \cdots + (n-3) + (n-2) + (n-1) = \text{total}$$

$$n + n + n + \cdots + n + n + n = 2 \cdot \text{total}$$

$$(n-1)n = 2 \cdot \text{total} \qquad \text{total} = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Version 2:

i loop goes from $i=0$ to size -2
  comparison

How many comparisons are done in total? $n-1$

version 3

i loop goes from $i=0$ to size -2
  10k comparisons

How many comparisons are done in total? $10{,}000(n-1)$
$= 10{,}000n - 10{,}000$

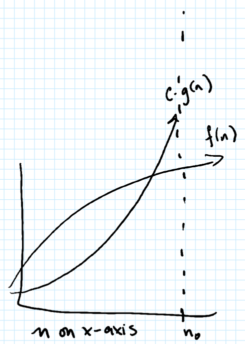
version 1: $\frac{n^2}{2} - \frac{n}{2}$
version 2: $n-1$ ← best
version 3: $10000n - 10000$

version 3 seems better than version 2 for bigger inputs

Definition of big-O:

Let $f(n)$ and $g(n)$ be functions.
We say that $f(n)$ is $O(g(n))$

if there exist a constant $c > 0$
and a constant $n_0 \geq 1$ such that
$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



We say $f(n)$ is asymptotically upper-bounded by $g(n)$.

version 1: $\frac{n^2}{2} - \frac{n}{2}$ comparisons

Want to show this is $O(n^2)$

$c = $ 1 $c = 100$
$n_0 = $ 1

need to check:
for $n \geq n_0 = 1$, $\frac{n^2}{2} - \frac{n}{2} \overset{?}{\leq} 1 \cdot n^2$ $100 \cdot n^2$

$$\frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2} \leq n^2 = 1 \cdot n^2 \leq 100n^2 \checkmark$$

↑ b/c $n \geq 1$, so $-\frac{n}{2} \leq 0$

Version 2: $n-1$ comparisons

outline
- review classes of algorithms
- review def of big O
- proofs of big O
- sorting
  - selection sert
  - mergesort
  - big O analysis of each

first test -- weeks 1 and 2, C++ but not bigO
(we only test you on things where you did the lab and got a grade back)
(the study guide auto-hides answers so you can check yourself)

Review classes of algorithms
1. what class grows proportionally to the size of the problem? linear $O(n)$
2. what's the fastest class of algorithms? constant time $O(1)$
3. what's an example of a quadratic algorithm? selection sort, bubble sort, insertion sort $O(n^2)$
4. which class of algorithms is the slowest? factorial $O(n!)$
5. what class do the fastest sorting algorithms fall into? $O(n \log n)$

Definition of big-O
We say that $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Proofs

Example 1: Show that $f(n) = n^2 + 3n + 1$ is $O(n^2)$.
We know $n > 0$ as it's the size of the problem, and n is an integer.
We also know $n^2 \leq n^2$ and $3n \leq 3n^2$ and $1 \leq n^2$, so
$n^2 + 3n + 1 \leq n^2 + 3n^2 + n^2 = 5n^2$
... so as long as we choose a constant $c \geq 5$ and $n_0 \geq 1$, we can say that $f(n)$ is $O(g(n))$
because $n^2 + 3n + 1 \leq 5n^2$.

Example 2: Show that ...

Example 3: Show that $f(n) = 4n^4 - 5n^3 + 6n - 7n + 8$ is $O(n^4)$ assuming $n > 0$.
We know $0 > -5n^3$ and $0 > -7n$.
So $4n^4 - 5n^3 + 6n^2 - 7n + 8 \leq 4n^4 - 0 + 6n^4 - 0 + 8n^4 = 18n^4$.
So as long as $c \geq 18$ and $n_0 \geq 1$, we know that $f(n)$ is $O(n^4)$.

Example 4: Is $f(n) = 4n^2$....
$O(n^2)$? yes
$O(n^3)$? yes
$O(n^{100})$? yes

Remember that big O provides an upper bound, but the definition does *not* require it to be a *tight* upper bound.
Obviously we prefer tighter upper bounds because they give us a better sense of the algorithm's efficiency.
Example 5: suppose $f(n) = n^2$ and $g(n) = 7n^2 + 3n$, which of the following is the best answer:
[discussion about why we do this, is this good, generally we just say $O(n^2)$ and not anything weirder like $O(7n^2 + 4)$.
Goal: take an unsorted array of elements and rearrange them such that they are in ascending order.

Sanity check:
  if $n = 1000$ & then $n = 3000$,
  how much longer will this take?
  If our analysis is right, a problem 3 times
  larger should take 9 times longer

onscreen demo

./sortTest 1000 selectSort
output shows the last 10 elements as a sanity check for "did it sort correctly?"

time ./sortTest 1000 selectSort
takes .009s
triple it
takes .02s
10,000 took 0.1s
30,000 took 0.9s
so you have to sometimes pick a big enough input size to have the difference in runtime actually show.

Mergesort

- example of a divide-and-conquer style algorithm (think: like binary search)
- typically have **recursive** solutions
  - base case, where no recursion is necessary
  - recursive cases, where we use the same approach on smaller versions of the problem
- **conquer** by putting separate results from recursion back together

reminder: a **recursive** function is a function that calls itself

```
mergeSort(array, size):
    if size < 2:
        return // you're all done! that array is definitely sorted
    copy the first half of the array into a new array B
    copy the second half of the array into a new array C
    mergeSort(B,size/2)
    mergeSort(C,size/2)

    merge(B,C, array) // helper function merge takes two arrays and puts them back in sorted order in
    the original array.
```

$\left.\right\}$ this is the DIVIDING

merge function:

$c = 1$ $c = 100$

$n_0 = 1$

$\frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2} \leq n^2 = 1 \cdot n^2 \leq 100n^2$ ✓

b/c $n \geq 1$, so $-\frac{n}{2} \leq 0$

Version 2 : $n-1$ comparisons

want to show this is $O(n)$

$c = 1$ $c=5$

$n_0 = 1$ $n = 18$

need to check:

for $n \geq n_0 = 1$, $n-1 \overset{?}{\leq} 1 \cdot n$

$n - 1 \leq n$ for $n \geq 1$ ✓

for $n \geq 18$, $n-1 \leq 5n$ ✓

version 3: $10{,}000n - 10{,}000$ comparisons

want to show this is $O(n)$

$c = 10{,}001$ $c = 8$ ✗ no good

$n_0 = 1$

need to check:

for $n \geq n_0 = 1$ $10000n - 10{,}000 \overset{?}{\leq} 10{,}001 \cdot n$ $8 \cdot n$

$10{,}000n - 10{,}000 \leq 10{,}000n \leq 10{,}001 \cdot n$ ✓

$(10{,}000 - 8)n \overset{?}{\leq} 10{,}000$

Version 1: $O(n^2)$

version 2: $O(n)$

version 3: $O(n)$

Classes of algorithms by runtime (from fastest to slowest)

- constant $O(1)$
  - ex: return the last element in an array
- logarithmic $O(\log_2 n)$
  - ex: binary search
- linear $O(n)$
  - ex: is_sorted version 2, 3 ; linear
- $O(n \log n)$
  - ex: mergesort
- quadratic $O(n^2)$ ex: is_sorted version 1, selection sort, bubblesort, insertion sort
- $O(n^3)$
- exponential $O(2^n)$
- factorial $O(n!)$

big O practice:

Definition of big-O:

Let $f(n)$ and $g(n)$ be functions.
We say that $f(n)$ is $O(g(n))$
if there exist a constant $c > 0$
and a constant $n_0 \geq 1$ such that
$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

To prove that
$f(n)$ is $O(g(n))$
we need to find
values for c and n_0
that satisfy the definition.

(A) Show that $f(n) = n^2 + 6n + 2$ is $O(n^2)$.

Assume: $c = 9$

$n_0 = 3$

Want to show: $n^2 + 6n + 2 \leq 9 \cdot n^2$ for $n \geq n_0 = 3$.

$n^2 + 6n + 2 \leq n^2 + 6n^2 + 2$ // $6n \leq 6n^2$ for $n \geq 3$

$\leq n^2 + 6n^2 + 2n^2$ // $2 \leq 2n^2$ for $n \geq 3$

$= 9n^2$ ✓

(B) Show that $f(n) = 4n^5 - 3n^4 + 8n^3 - 7n^2 + 12$
is $O(n^5)$. (Safe to assume n always > 0.)

(C) The function $f(n) = 20n^3$ is ...

$O(n^2)$? no

$O(n^3)$? yes $c = 20$, $n_0 = 2$

$O(n^4)$? yes $c = 20$, $n_0 = 1$

$O(n^{100})$? yes

In general, we want our O bounds to be tight (close to the function f)
in order to be useful.

(D) Let $f(n) = n^2$ and $g(n) = 8n^2 + n$.
What best describes these functions?

```
copy the first half of the array into a new array B
copy the second half of the array into a new array C
mergeSort(B,size/2)
mergeSort(C,size/2)

merge(B,C, array) // helper function merge takes two arrays and puts them back in sorted order in the original array.
```

this is the DIVIDING

merge function:

live blackboard crowdsourcing with giant playing cards!

```
merge(A, sizeA, B, sizeB, C):
    i=0, j=0, k=0 // indices for A, B, and C, respectively
    while i<sizeA and j<sizeB:
        if A[i] ≤B[j]:
            C[k++] = A[i++] // this uses the values k and i, then increments each of them
        else
            C[k++] = B[j++]
    while i<sizeA: // to handle the leftover items if they are in A
        C[k++] = A[i++]
    while j<sizeB: // ditto
        C[k++] = B[j++]
```

live class demo of mergesort w/students (as stack frames)

how much ~~work~~ levels to get to base case? $O(\log_2 n)$

how much ~~work~~ levels to merge 2 lists of size $n/2$? $O(n)$

$O(n \log_2 n)$ work overall

1. $f(n)$ is $O(g(n))$ ← true
2. $g(n)$ is $O(f(n))$ ← true $8n^2 + n \le \overset{9}{\overset{\shortparallel}{C}} \cdot n^2$ for $n \ge n_0 = 1$
3. both 1 and 2 ← best answer!
4. neither 1 nor 2

SORTING

problem: take an unsorted array of elements and rearrange them to be in ascending (increasing) order

Selection Sort: pseudocode

```
selectSort(array, size)
    for i = size - 1 down to 1
        indexOfMax = findMax(array, i)   ← end
        swap(array, i, indexOfMax)

findMax(array, end)
    indexOfMax = 0
    for i = 1 to end
        if array[i] > array[indexOfMax]
            indexOfMax = i
    return indexOfMax
```

example run of selection sort:

0 [7, 6, 9, 1, 3, 5, 4]