3.2 big-O and sorting

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: is_sorted

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

How many comparisons are done in total?

$$\begin{aligned} (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1 &= \text{total} \\ 1 + 2 + 3 + \cdots + (n-3) + (n-2) + (n-1) &= \text{total} \\ \hline n + n + n \cdots\cdots + n &= 2 \cdot \text{total} \end{aligned}$$

$$(n-1)\,n = 2 \cdot \text{total}$$

$$\text{total} = \frac{n^2}{2} - \frac{n}{2}$$

Version 2:

i loop goes from i=0 to $\underbrace{\text{size}-1}_{n-1}$
  comparison

How many comparisons are done in total? $n-1$
  input size is length of array, called n

version 3

i loop goes from i=0 to size-1
  10k comparisons

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

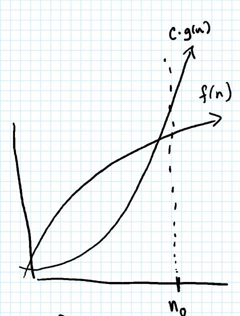
version 1: $\frac{n^2}{2} - \frac{n}{2}$

version 2: $n-1$

version 3: $10000n - 10000$

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 $f(n) = \frac{n^2}{2} - \frac{n}{2}$ $g(n) = n^2$

Want to show this is $O(n^2)$ for $n \geq 1 = n_0$, is

$c = \underline{4}$ $\frac{n^2}{2} - \frac{n}{2} \overset{?}{\leq} 4 \cdot n^2$

$n_0 = \underline{1}$ $\frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2} \leq 4 \cdot n^2$ since $n \geq 1$

Version 2: $n-1$ comparisons

<u>outline</u>
- 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)

<u>Review classes of algorithms</u>
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? $O(n \log n)$

<u>Definition of big-O</u>
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 $n \geq n_0$.

<u>Proofs</u>

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.
```

} this is the DIVIDING

merge function:

$c = \underline{4}$ $\qquad \frac{n^2}{2} - \frac{n}{2} \overset{?}{\leq} 4 \cdot n^2$

$n_0 = \underline{1}$ $\qquad \frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2} \leq 4 \cdot n^2$ since $n \geq 1$

Version 2 : $n-1$ comparisons

want to show this is $O(n)$ $\qquad$ for $n \geq 1$:

$c = \underline{3}$ $\qquad n-1 \overset{?}{\leq} 3 \cdot n$

$n_0 = \underline{1}$ $\qquad n-1 \leq n \leq 3 \cdot n$ since $n \geq 1$

alternate:
$c=1$ $\quad n-1 \leq n \leq 1 \cdot n$
$n_0 = 1$

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

want to show this is $O(n)$ $\qquad$ for $n \geq 10$:

$c = \underline{20{,}000}$ $\qquad 10{,}000n - 10{,}000 \overset{?}{\leq} 20{,}000n$

$n_0 = \underline{10}$ $\qquad 10{,}000n - 10{,}000 \leq 10{,}000n \leq 20{,}000n$

version 1 $\quad O(n^2)$
version 2 $\quad O(n) \leftarrow$ we picked $c=3$
version 3 $\quad O(n) \leftarrow c = 20{,}000$

common classes of algorithms (from fastest to slowest ...)

- constant $O(1)$
  ex: return the final element in an array
- logarithmic $O(\log_2 n)$
  ex: binary search
- linear $O(n)$
  ex. is_sorted version 2
- $O(n \log n)$
  ex. mergesort, quicksort
- quadratic $O(n^2)$
  ex. is_sorted version 1, bubblesort, selectionsort
- $O(n^3)$
- 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: $c = \underline{9}$
$n_0 = \underline{1}$

Let $n \geq n_0 = 1$.

$f(n) = n^2 + 6n + 2 \overset{?}{\leq} 9 \cdot n^2$

$n^2 + 6n + 2 \underset{\text{b/c } 6n \leq 6n^2}{\leq} n^2 + 6n^2 + 2 \underset{2 \leq 2n^2}{\leq} n^2 + 6n^2 + 2n^2 = 9n^2$ ✓

(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
$O(n^4)$? yes
$O(2^n)$? yes

$\}$ correct, but less helpful than a tighter O

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

$c=1$, $n_0=1$ $\qquad n^2 \leq 8 \cdot n^2 + n$

```
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)
```

} this is the DIVIDING

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

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

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

1. $f(n)$ is $O(g(n))$ ← true $c=1$, $n_0=1$ $n^2 \le 8 \cdot n^2 + n$
2. $g(n)$ is $O(f(n))$ ← true $c=100$, $n_0=1$ $8n^2 + n \le 100n^2$
3. both 1 and 2 ☺
4. neither 1 nor 2 ← false

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)   ← stop looking at this index
        swap(array, i, indexOfMax)

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

How much work (# of swaps + comparisons) is done?

	iteration 1:	iteration 2:	iter 3:		iteration n-1:
swaps:	1	1	1	. . . .	1
comparisons:	$(n-1)$ +	$(n-2)$ +	$(n-3)$ +	. . . +	$(n-(n-1)=1)$

total # comparisons = $\frac{(n)(n-1)}{2}$ so Selection Sort is $O(n^2)$

example run of selection sort:

① $[7, 6, 9, 1, 3, 5, 4]$
indices: 0 1 2 3 4 5 6

$n = 7$, $i = 6$
indexOfMax = 2 b/c that's where largest el is

② $[7, 6, 4, 1, 3, 5, 9]$ ← this part is sorted
indices: 0 1 2 3 4 5 6

$i = 5$
indexOfMax: 0

③ $[5, 6, 4, 1, 3, 7, 9]$
indices: 0 1 2 3 4 5 6

$i = 4$
indexOfMax = 1