Analysis of insertion sort

Google Classroom Facebook Email Twitter

Like selection sort, insertion sort loops over the indices of the array. It just calls `insert` on the elements at indices $1, 2, 3, \ldots, n-1$. Just as each call to `indexOfMinimum` took an amount of time that depended on the size of the sorted subarray, so does each call to `insert`. Actually, the word "does" in the previous sentence should be "can," and we'll see why.

Let's take a situation where we call `insert` and the value being inserted into a subarray is less than every element in the subarray. For example, if we're inserting 0 into the subarray [2, 3, 5, 7, 11], then every element in the subarray has to slide over one position to the right. So, in general, if we're inserting into a subarray with k elements, all k might have to slide over by one position. Rather than counting exactly how many lines of code we need to test an element against a key and slide the element, let's agree that it's a constant number of lines; let's call that constant c. Therefore, it could take up to $c \cdot k$ lines to insert into a subarray of k elements.

Suppose that upon every call to `insert`, the value being inserted is less than every element in the subarray to its left. When we call `insert` the first time, $k = 1$. The second time, $k = 2$. The third time, $k = 3$. And so on, up through the last time, when $k = n - 1$. Therefore, the total time spent inserting into sorted subarrays is

$$c \cdot 1 + c \cdot 2 + c \cdot 3 + \cdots c \cdot (n-1) = c \cdot (1 + 2 + 3 +$$

.

That sum is an arithmetic series, except that it goes up to $n - 1$ rather than n. Using our formula for arithmetic series, we get that the total time spent inserting into sorted subarrays is

$$c \cdot (n - 1 + 1)((n-1)/2) = cn^2/2 - cn/2 \,.$$

Using big-Θ notation, we discard the low-order term $cn/2$ and the constant factors c and 1/2, getting the result that the running time of insertion sort, in this case, is $\Theta(n^2)$.

Can insertion sort take *less* than $\Theta(n^2)$ time? The answer is yes. Suppose we have the array [2, 3, 5, 7, 11], where the sorted subarray is the first four elements, and we're inserting the value 11. Upon the first test, we find that 11 is greater than 7, and so no elements in the subarray need to slide over to the right. Then this call of `insert` takes just constant time. Suppose that *every* call of `insert` takes constant time. Because there are $n - 1$ calls to `insert`, if each call takes time that is some constant c, then the total time for insertion sort is $c \cdot (n - 1)$, which is $\Theta(n)$, not $\Theta(n^2)$.

Can either of these situations occur? Can each call to `insert` cause every element in the subarray to slide one position to the right? Can each call to `insert` cause no elements to slide? The answer is yes to both questions. A call to `insert` causes every element to slide over if the key being inserted is less than every element to its left. So, if every element is less than every element to its left, the running time of insertion sort is $\Theta(n^2)$. What would it mean for every element to be less than the element to its left? The array would have to start out in *reverse* sorted order, such as [11, 7, 5, 3, 2]. So a reverse-sorted array is the worst case for insertion sort.

How about the opposite case? A call to `insert` causes no elements to slide over if the key being inserted is greater than or equal to every element to its left. So, if every element is greater than or equal to every element to its left, the running time of insertion sort is $\Theta(n)$. This situation occurs if the array starts out already sorted, and so an already-sorted array is the best case for insertion sort.

What else can we say about the running time of insertion sort? Suppose that the array starts out in a random order. Then, on average, we'd expect that each element is less than half the elements to its left. In this case, on average, a call to `insert` on a subarray of k elements would slide $k/2$ of them. The running time would be half of the worst-case running time. But in asymptotic notation, where constant coefficients don't matter, the running time in the average case would still be $\Theta(n^2)$, just like the worst case.

What if you knew that the array was "almost sorted": every element starts out at most some constant number of positions, say 17, from where it's supposed to be when sorted? Then each call to `insert` slides at most 17 elements, and the time for one call of `insert` on a subarray of k elements would be at most $17 \cdot c$. Over all $n - 1$ calls to `insert`, the running time would be $17 \cdot c \cdot (n - 1)$, which is $\Theta(n)$, just like the best case. So insertion sort is fast when given an almost-sorted array.

To sum up the running times for insertion sort:

- Worst case: $\Theta(n^2)$.
- Best case: $\Theta(n)$.
- Average case for a random array: $\Theta(n^2)$.
- "Almost sorted" case: $\Theta(n)$.

If you had to make a blanket statement that applies to all cases of insertion sort, you would have to say that it runs in $O(n^2)$ time. You cannot say that it runs in $\Theta(n^2)$ time in all cases, since the best case runs in $\Theta(n)$ time. And you cannot say that it runs in $\Theta(n)$ time in all cases, since the worst-case running time is $\Theta(n^2)$.

Ask a question...

Questions Tips & Thanks Top Recent

I am not able to understand this situation- "say 17, from where it's supposed to be when sorted? "
how running time can be 17·c·(n−1). for this situation.
Please make it clear.

8 votes • Comment • Flag
3 years ago by Gaurav Pareek

Basically, it is saying:
-Suppose the insert function, at most, performs 17 comparisons each time it is called (because the array is almost sorted)
-A comparison costs c and we perform 17 of them per insert, so the cost of an insert is 17 * c
-The insertion sort function calls the insert function on each of the n-1 elements (we get the 1st element for free), so:
cost of an insert * number of inserts = (17 * c) * (n-1)

Hope this makes sense

● 39 votes ▲▼ ● 8 comments ● Flag
3 years ago by Cameron

Show all 4 answers ● Answer this question

---

why wont my code checkout

```
var insert = function(array, rightIndex, value) {
for(var j = rightIndex; j > 0 && array[j-1] > value; j--) {
array[j] = array[j-1];
}
array[j] = value;
};

var insertionSort = function(array) {
for(var i = 0; i < array.length; i++){
insert(array, i, array[i]);
}
};

var array = [22, 11, 99, 88, 9, 7, 42];
insertionSort(array);
println("Array after sorting: " + array);
Program.assertEqual(array, [7, 9, 11, 22, 42, 88, 99]);
```

6 votes ▲▼ ● 2 comments ● Flag 3 years ago by csalvi42

You shouldn't modify functions that they have already completed for you, i.e. insert() , if you want to pass the challenges. If you change the other functions that have been provided for you, the grader won't be able to tell if your code works or not (It is depending on the other functions to behave in a certain way).

7 votes ▲▼ ● Comment ● Flag
3 years ago by Cameron

Show all 6 answers ● Answer this question

---

can the best case be written as big omega of n and worst case be written as big o of n^2 in insertion sort?

4 votes ▲▼ ● Comment ● Flag
about a year ago by ayush.goyal551

Yes, It can be.

3 votes ▲▼ • Comment • Flag
12 months ago by Akash Lakhera

Can we make a blanket statement that insertion sort runs it omega(n) time?

2 votes ▲▼ • Comment • Flag 2 years ago by Sam Chats

Yes, you could. In the best case (array is already sorted), insertion sort is omega(n). You can't possibly run faster than the lower bound of the best case, so you could say that insertion sort is omega(n) in ALL cases.

5 votes ▲▼ • Comment • Flag
2 years ago by Cameron

I don't understand how O is (n^2) instead of just (n); I think I got confused when we turned the arithmetic summ into this equation:
c·(n−1+1)((n−1)/2)=cn^2/2−cn/2

I inferred from the text that it takes for granted some mathematical theorem that I have no idea of. Is that what's happening? Thanks!

2 votes ▲▼ • Comment • Flag 2 years ago by Miriam BT

In general the sum of 1 + 2 + 3 + ... + x = (1 + x) * (x)/2

For more info on how it is derived see this to understand what an arithmetic sequence is:
https://www.khanacademy.org/math/precalculus/seq-induction/sequences-review/v/arithmetic-

sequences

and this to understand how the formula was derived: https://www.khanacademy.org/math/precalculus/seq-induction/seq-and-series/v/alternate-proof-to-induction-for-integer-sum

4 votes ▲▼ • 3 comments • Flag
2 years ago by Cameron

Show all 2 answers • Answer this question

Thank you for this awesome lecture. Would it be possible to include a section for "loop invariant"?

2 votes ▲▼ • Comment • Flag 3 years ago by me me

Loop invariants are really simple (but finding the right invariant can be hard):
They are statements that must be true before the loop and after the loop.

For insertion sort we have the loop invariant:
"After the kth iteration, elements a[0] to a[k] are sorted"

Before we start the first loop (we have perfomed 0 iterations) k=0 thus this is true:
"After iteration 0, elements a[0] to a[0] are sorted"
i.e. the first element is trivially sorted

In the first loop we insert a[1] in order on a[0].
After… (more)

4 votes ▲▼ • Comment • Flag
3 years ago by Cameron

The sumup is very helpful
Please tell me, how do you take each case running time in consideration before reaching the final conclusion?

1 vote ▲▼ • Comment • Flag

2 years ago by 1 con vịt siêu nhân

You can't be any faster than the best case, so we can say that the lower bound for running time for ALL cases is the lower bound of the best case. In this case, Ω(n).

You can't be slower than the worst case, so we can say that the upper bound for running time for ALL cases is the upper bound for the worst case. In this case, O(n^2)

Ω and O don't have the same asymptotic complexity when we consider ALL cases, so we can't make a statement about Θ for ALL cases.

Hope this makes sense

5 votes ▲▼ • Comment • Flag
2 years ago by Cameron

```
 var insert = function(array, rightIndex,
value) {
for(var i = rightIndex;
i > 0 && array[i-1] > value;
i--) {
array[i] = array[i-1];
}
array[i] = value;
};

var insertionSort = function(array) {
for (var st = 1; st < array.length; st++)
{
insert(array, st, array[st]);
}
};

var array = [22, 11, 99, 88, 9, 7, 42];
insertionSort(array);
```

```
println("Array after sorting: " + array);
Program.assertEqual(array, [7, 9, 11, 22,
42, 88, 99]);
```

Why this code doesn't work properly?

2 votes ▲▼ • Comment • Flag
3 years ago by Andrej Benedičič

It looks like you changed the `insert` function, but you should only be modifying the `insertionSort` function. The graders for these challenges assume that you aren't changing the completed functions that they provide you with.

3 votes ▲▼ • 1 comment • Flag
3 years ago by Cameron

No sure why following code does not work. I keep getting "A function is taking too long..." message. Any help?
var insert = function(array, rightIndex, value) {
for(var j = rightIndex;
j >= 0 && array[j] > value;
j--) {
array[j + 1] = array[j];
}
array[j + 1] = value;
};

var insertionSort = function(array) {
for (var i=1; i < array.length ; i++){
insert (array,i,array[i]);
}
};

var array = [22, 11, 99, 88, 9, 7, 42];
insertionSort(array);

1 vote ▲▼ • Comment • Flag 2 years ago by Jayanth

The insertionSort function has a mistake in the insert statement (Check the values of arguments that you are passing into it).

When your insertionSort function reaches the end of your array its call to insert is adding a copy of the last element of the array onto the end of itself. This causes the array to increase in size, which makes the condition in the for loop true. It repeats this forever until you get the error message.

Good Luck

3 votes ▲▼ • Comment • Flag
2 years ago by Cameron

I thought the asymptotic analysis was always with respect to worst-case runs. If so, then what is the significance of saying, "You cannot say that it runs in Θ(n^2) Θ(n^2) time in all cases, since the best case runs in Θ(n) time"?

1 vote ▲▼ • Comment • Flag 3 years ago by Ethan Godt

We often talk about the worst-case running time since it's a useful metric to look at. However you can also use asymptotic notation to describe the running-time of other cases, like the best-case or average-case.

2 votes ▲▼ • Comment • Flag
3 years ago by jdsutton

Show all 2 answers • Answer this question

Show more comments

‹ Challenge: Implement insertion sort

Our mission is to provide a free, world-class education to anyone, anywhere.

Khan Academy is a 501(c)(3) nonprofit organization. **Donate** or **volunteer** today!

About

News

Impact

Our team

Our interns

Our content specialists

Our leadership

Our supporters

Our contributors

Careers

Internships

Contact

Help center

Support community

Share your story

Press

Download our apps

iOS app

Android app

Subjects

Math by subject

Math by grade

Science & engineering

Computing

Arts & humanities

Economics & finance

Test prep

College, careers, & more

Language English Go