Exercise 7

Finger Exercises due Aug 5, 2020 20:30 -03

Exercise 7

4/7 points (graded)
ESTIMATED TIME TO COMPLETE: 14 minutes

This problem will walk through some applications of complexity analysis. Suppose you're asked to implement an application. One of the things it has to do is to report whether or not an item, `x`, is in a list `L`. `L`'s contents do not change over time. Below are two possible ways to implement this functionality. Assume that `mergeSort` is implemented as per the lecture.

`L` is a list with `n` items.

- **Application A:**

  Every time it's asked to, it performs a linear search through list `L` to find whether it contains `x`.

- **Application B:**

  Sort list `L` once before doing anything else (using `mergeSort`). Whenever it's asked to find `x` in `L`, it performs a binary search on `L`.

1. If the application is asked to find `x` in `L` exactly one time, what is the worst case time complexity for Application A?

- $O(1)$
- $O(\log n)$
- [✓ $O(n)$]
- $O(n \log n)$
- $O(n^2)$

✓

Explanation:
Application A just performs one linear search through `n` elements. This is $O(n)$ complexity.

2. If the application is asked to find `x` in `L` exactly one time, what is the worst case time complexity for Application B?

- $O(1)$
- $O(\log n)$
- $O(n)$
- [✓ $O(n \log n)$]
- $O(n^2)$

Explanation:
The time complexity for Application B is how long it takes to do `mergeSort` once, plus how long it takes to do a binary search. That works out to $O(n \log n + \log n)$, which is just $O(n \log n)$.

3. If the application is asked to find `x` in `L` k times, what is the worst case time complexity for Application A?

- ○ $O(1)$
- ○ $O(k + \log n)$
- ○ $O(k + n)$
- [✓ ● $O(kn)$]
- ○ $O(n + k \log n)$

✓

Explanation:
We have to do *k* linear searches so the time complexity is $O(kn)$.

4. If the application is asked to find `x` in `L` *k* times, what is the worst case time complexity for Application B?

- ○ $O(kn)$
- ○ $O(n \log n)$
- ○ $O(n + k \log n)$
- [✓ ● $O(n \log n + k \log n)$]
- ○ $O(kn \log n + \log n)$

✓

Explanation:
Doing the search *k* times means that the time complexity is how long it takes to do `mergeSort` once, plus how long it takes to do a binary search *k* times. That works out to $O(n \log n + k \log n)$. Since we don't know what the value of *k* is, we cannot simplify further.

5. What value(s) of *k* would make Application A be faster (i.e., asymptotically grow slower than) Application B?

- [✓] $k = 1$
- [ ] $k = n$
- [ ] $k = \log n$
- [ ] $k = n^2$
- [ ] $k = 2^n$

Explanation:
When $k = 1$, Application A's complexity is $O(kn) = O(n)$, and Application B's complexity is $O(n \log n + k \log n) = O(n \log n + \log n)$.
Setting *k* at any value greater than 1 makes $O(kn)$ asymptotically grow faster than $O(n \log n + k \log n)$.

6. What value(s) of *k* would make Application A grow at the same rate as Application B?

- [ ] $k = 1$
- [ ] $k = n$
- [✓] $k = \log n$
- [ ] $k = n^2$
- [ ] $k = 2^n$

✓

Explanation:
When $k = \log n$, Application A's complexity is $O(kn) = O(n \log n)$, and Application B's complexity is
$O(n \log n + k \log n) = O(n \log n + \log n \log n)$. $\log n \log n$ grows slower than $n \log n$, so in this case Application B's time complexity is $O(n \log n)$. So. when $k = \log n$, the order of time complexity of Applications A and B are eq

7. Which application should you choose if you know that there are going to be n^3 requests to find `x` in `L`?

○ Application A

[✓ Application B]

Explanation:
When $k = n^3$, Application A has time complexity $O\left(n^4\right)$ whilst Application B's time complexity is $O\left(n^3 logn\right)$. Generally we can extrapolate that for very large *k*, Application B will be the preferred choice.

Enviar

Exercise 7

Ocultar discussão

Topic: Lecture 12 / Exercise 7

Add a Post

Show all posts por atividade recente

There are no posts in this topic yet.

✖