Theory of Algorithms Homework 2

Evan Dreher

September 2022

1 Problem 1

Consider the following algorithm for *binary search*. Assume that x is the value to searched for and T is a non-empty, already sorted array of numbers containing that value.

Algorithm 1 Reverse-Search(x, T)

```
$low = 1$
$high = T.length$
while low < high do
    $mid = floor(\frac{low+high}{2})$
    if $x \leq T[mid]$ then
        $high = mid$
    else
        $low = mid + 1$
return $high$
```

Propose a loop invariant for the algorithm; prove the initial, maintenance, and termination conditions for this invariant.

- **Loop Invariant:** At the start of the loop, $T[low] \leq x \leq T[high]$.
- **Initialization:** At the start of the first loop, low = 1 and high = T.length which means T[low]...T[high] is the whole array. Since T is in sorted order, and $x \in T$, the LI is trivially true at the start of the first loop.
- **Maintenance:** Assume the LI holds at the start of the loop. The value mid will be the average of high minus low rounded down. If $x \leq T[mid]$ it follows then by our LI it follows that $T[low] \leq x \leq T[mid]$. So when the algorithm sets the value of high equal to mid it follows that the LI will hold for the next loop. If instead $x > T[mid]$ it follows that $T[mid + 1] \leq x \leq T[high]$ so when the algorithm sets that value of low equal to $mid + 1$ it follows that the LI will hold at the start of the next loop. Thus the LI holds in all cases.
- **Termination:** The loop terminates when $low = high$. By the LI we know that $T[low] \leq x \leq T[high]$. Thus $T[low] = x = T[high]$. So the algorithm can then return $high$ knowing it is the index of T that contains x.

2 Problem 2

- Look back at the binary search problem. Make an argument for the O, Omega, and Theta bounds of this algorithm, in your own words. You should reference the specifics of this code implementation as needed.

  The algorithm searches for x by narrowing the distance between $high$ and low (which it knows x is between). Each iteration halves the distance between those values, so the algorithm runs a number of loops equal to the number of times you must divide n by two to get 1. This is known as $lg(n)$ thus the algorithm is $O(lg(n))$. Furthermore, since the algorithm has no early breaking condition in the loop, we know it is also $\Omega(lg(n))$. Since $O = \Omega$, the algorithm runs in $\Theta(lg(n)$ time.

- Now suppose we replace the "if/else" clause with the following. Again, make an argument for the O, Omega, and Theta bounds of the algorithm.

```
if $x < T[mid]$ then
    $high = mid$
else if $x > T[mid]$ then
    $low = mid + 1$
else
    return $mid$
```

  The Algorithm does not get any slower so it is still $O(lg(n))$, but since there is now an early termination condition (when $x = T[mid]$) the algorithm could break as soon as the first loop making it $\Omega(1)$. Since $O \neq \Omega$ there is no Theta bound for the algorithm.

3 Problem 3

Lets return to binary search once more. This time, however, we're going to assume that we start with a value x and a pointer to the *head* of a doubly-linked list. We still know how many elements are in it, and they're still in sorted order, but we lost the ability of an array to have constant time lookups. Instead we have a method called `traverse(dir, link, k)` which takes three arguments - a direction (forward or backward), a link to start at, and a number of links k to move in that direction, and returns the link at the target location. The runtime for traverse is $\Theta(k)$. Our revised code is:

Algorithm 2 Binary-Search(x, head, length)

```
$low = 1$
$high = length$
$current = head$
while low < high do
    $mid = floor(\frac{low+high}{2})$
    $midLink = traverse(forward, current, mid)$
    if x $\leq$ midLink.value then
        $high = mid$
    else
        $low = mid + 1$
        $current = traverse(forward, midLink, 1)$
return $current$
```

Just like a regular binary search, the while loop will have to execute $lg(n)$ times. Everything inside the loop is constant time operations except for the call to `traverse` which is $\Theta(k)$. In each loop, k is half the distance between high and low. As we know that distance is being halved each iteration starting with $\frac{n}{2}$, then $\frac{n}{4}$, $\frac{n}{8}$ ... The total work done by the algorithm is the sum of these which can be represented as

$$\sum_{i=1}^{lg(n)} \frac{n}{2^i}$$

$$n \times \sum_{i=1}^{lg(n)} \frac{1}{2^i}$$

This summation in the second equation converges to 1 meaning the total work done by the algorithm is $n \times 1$ or just n. Since there is no condition for early termination we can say that the algorithm is $O(n)$, $\Omega(n)$, and $\Theta(n)$.

4 Problem 4

Prove that the following statements are true

- $n = O(n\ lg(n))$

  $n = O(n\ lg(n))$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $n \leq c \times (n\ lg(n))$ **for all** $n \geq n_0$**. Let** $f(n) = n$**,** $g(n) = n\ lg(n)$**,** $c = 1$**,**$n_0 = 2$

$$\begin{aligned} f(n) &= n \\ &= n \times 1 \\ &\leq n\ lg(n) && \text{since n } \geq 2 \\ &= c \times g(n) \end{aligned}$$

  Therefore, $n = O(n\ lg(n))$

- $n^2 + 3 = O(n\ lg(n^2))$

  $n^2 + 3 = O(n^2)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $n^2 + 3 \leq c \times (n^2)$ **for all** $n \geq n_0$**. Let** $f(n) = n^2 + 3$**,** $g(n) = n^2$**,** $c = 2$**,**$n_0 = 2$

$$\begin{aligned} f(n) &= n^2 + 3 \\ &\leq n^2 + 4 \\ &\leq n^2 + n^2 && \text{since } n \geq 2 \\ &= 2n^2 \\ &= c \times g(n) \end{aligned}$$

  Therefore, $n^2 + 3 = O(n\ lg(n^2))$

- $n^3 - n = \Theta(n^3)$

  $n^3 - n = \Theta(n^3)$ **if** $n^3 - n = O(n^3)$ **and** $n^3 - n = \Omega(n^3)$**.**

  $n^3 - n = O(n^3)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $n^3 - n \leq c \times (n^3)$ **for all** $n \geq n_0$**. Let** $f(n) = n^3 - n$**,** $g(n) = n^3$**,** $c = 1$**,**$n_0 = 0$

$$\begin{aligned} f(n) &= n^3 - n \\ &\leq n^3 && \text{since } n \geq 0 \\ &= c \times g(n) \end{aligned}$$

  Therefore, $n^3 - n = O(n^3)$

  $n^3 - n = \Omega(n^3)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $n^3 - n \geq c \times (n^3)$ **for all** $n \geq n_0$**. Let** $f(n) = n^3 - n$**,** $g(n) = n^3$**,** $c = \frac{3}{4}$**,**$n_0 = 2$

$$\begin{aligned} f(n) &= n^3 - n \\ &= n^3 - \frac{1}{4} \times 2 \times 2 \times n \\ &\geq n^3 - \frac{1}{4}n^3 && \text{since } n \geq 2 \\ &= \frac{3}{4}n^3 \\ &= c \times g(n) \end{aligned}$$

  Therefore, $n^3 - n = \Omega(n^3)$

  Therefore, $n^3 - n = \Theta(n^3)$

- $5n^2 + 3n + 2 = \Theta(n^2)$

  $5n^2 + 3n + 2 = \Theta(n^2)$ **if** $5n^2 + 3n + 2 = O(n^2)$ **and** $5n^2 + 3n + 2 = \Omega(n^2)$**.**

$5n^2+3n+2 = O(n^2)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $5n^2+3n+2 \leq c \times (n^2)$ **for all** $n \geq n_0$**. Let** $f(n) = 5n^2+3n+2$**,** $g(n) = n^2$**,** $c = 7$**,**$n_0 = 3$

$$\begin{aligned} f(n) &= 5n^2 + 3n + 2 \\ &\leq 5n^2 + n^2 + n^2 && \text{since } n \geq 2 \\ &= 7 \times n^2 \\ &= c \times g(n) \end{aligned}$$

Therefore, $5n^2+3n+2 = O(n^2)$

$5n^2+3n+2 = \Omega(n^2)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $5n^2+3n+2 \geq c \times (n^2)$ **for all** $n \geq n_0$**. Let** $f(n) = 5n^2+3n+2$**,** $g(n) = n^2$**,** $c = 5$**,**$n_0 = 0$

$$\begin{aligned} f(n) &= 5n^2 + 3n + 2 \\ &\geq 5n^2 \\ &= c \times g(n) \end{aligned}$$

Therefore, $n^3 - n = \Omega(n^3)$

Therefore, $n^3 - n = \Theta(n^3)$

5 Problem 5

Prove the following statements are true

- $n + a = O(n)$ when a is a non-negative constant.

  $n + a = O(n)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $n + a \leq c \times n$ **for all** $n \geq n_0$**. Let** $f(n) = n + a$**,** $g(n) = n$**,** $c = 2$**,**$n_0 = a$

  $$\begin{aligned} f(n) &= n + a \\ &\leq n + n && \text{since } n \geq a \\ &= 2 \times n \\ &= c \times g(n) \end{aligned}$$

  Therefore, $n + a = O(n)$

- $(n+a)^b = O(n^b)$

  $(n+a)^b = O(n^b)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $(n+a)^b \leq c \times n^b$ **for all** $n \geq n_0$**. Let** $f(n) = (n+a)^b$**,** $g(n) = n$**,** $c = 2^b$**,**$n_0 = a$

  $$\begin{aligned} f(n) &= (n+a)^b \\ &\leq (n+n)^b && \text{since } n \geq a \\ &= 2^b \times n^b \\ &= c \times g(n) \end{aligned}$$

  Therefore, $(n+a)^b = O(n^b)$

- $(n-a)^b = \Omega(n^b)$

  $(n-a)^b = \Omega(n^b)$ **if there exists some** $c > 0$ **and** $n_0 \geq 0$ **such that** $(n+a)^b \leq c \times n^b$ **for all** $n \geq n_0$**. Let** $f(n) = (n-a)^b$**,** $g(n) = n$**,** $c = \frac{1}{2}^b$**,**$n_0 = 2a$

$$\begin{aligned} f(n) &= (n-a)^b \\ &= (n - \frac{1}{2}2a)^b \\ &\geq (n - \frac{1}{2}n)^b \\ &= (\frac{1}{2}n)^b \\ &= \frac{1}{2}^b \times n^b \\ &= c \times g(n) \end{aligned}$$

  Therefore, $(n+a)^b = O(n^b)$