Computer Information Systems 23 - Homework 9

Cody Vig

Problem 1

(20 points) Consider the list: `5, 12, 25, 32, 38, 46, 58, 62, 85, 90, 97, 105, 110`.

1. How many comparisons are made with **sequential search** to determine if the number 120 is in the list?
2. How many comparisons are made with **binary search** to determine if the number 120 is in the list?
3. Repeat 1 and 2 for the number 25.

Solution

Note that the length of the list is $n = 13$ and it is sorted.

Problem 1

Sequential search searches through the list linearly from left to right until it finds the element in question or it exhausts the list. Since 120 is not in the list, there will be $n = 13$ comparisons until then program terminates and returns `-1`.

Problem 2

The sequence of comparisons look like this:

```
# N = 13, m = 6
int sortedList [] = {
  5, 12, 25, 32, 38, 46, 58, 62, 85, 90, 97, 105, 110
}
```

Is `list[m] == 58` smaller than `item == 120`? Yes, proceed with:

```
# N = 7, m = 2
int sortedList [] = {62, 85, 90, 97, 105, 110}
```

Is `list[m] == 90` smaller than `item == 120`? Yes, proceed with:

```
# N = 3, m = 1
int sortedList [] = {97, 105, 110}
```

Is `list[m] == 105` smaller than `item == 120`? Yes, proceed with:

```
# N = 1, m = 0
int sortedList [] = {110}
```

Is `list[m] == 110` smaller than `item == 120`? Yes, so it is not in the list.

Here, there were only four ($\approx \log_2(13)$) comparisons.

Problem 3.1

`25` is in this list at index `2`, so the sequential search algorithm with find it in three comparisons.

Problem 3.2

The sequence of comparisons look like this:

```
# N = 13, m = 6
int sortedList [] = {
  5, 12, 25, 32, 38, 46, 58, 62, 85, 90, 97, 105, 110
}
```

Is `list[m] == 58` bigger than `item == 25`? Yes, proceed with:

```
# N = 6, m = 2
int sortedList [] = {5, 12, 25, 32, 38, 46}
```

Is `list[m] == 25` equal to `item == 25`? Yes, so we are done.

Here, there were only two comparisons.

Problem 2

(20 points) Each of the following expressions represents the number of operations for certain algorithms. What is the order of each of these expressions? That is, put the following expressions in terms of Big-O notation.

1. $n^3 - 8n$
2. $5n^4 + 2n^2 + 8$
3. $(n^2 + n - 1)(8n + 5)$
4. $8n(6n + 1)$

Solution

Definition. Suppose $f, g : \mathbb{N} \to \mathbb{R}$. We say $f \in \mathcal{O}(g)$ if $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right|$ is finite.

Theorem 1. Suppose $f : \mathbb{N} \to \mathbb{R}$. If $\lim_{n\to\infty} f(n)$ exists, then $\limsup_{n\to\infty} f(n) = \liminf_{n\to\infty} f(n) = \lim_{n\to\infty} f(n)$

Theorem 2. Let $f, g : \mathbb{N} \to \mathbb{R}$ be $f(n) = \sum_{k=0}^{d} a_k n^k$ and $g(n) = n^d$, where $\{a_0, a_1, \ldots, a_d \neq 0\} \subset \mathbb{R}$ are arbitrary real numbers and $d \in \mathbb{N}$ is the degree of the polynomials. Then $f \in \mathcal{O}(g)$.

Proof. We verify $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = a_d$ by definition. Let $\varepsilon > 0$ and $M = \max\limits_{k\in\{0,\ldots,d-1\}} \{|a_k|\}$, and let $N \in \mathbb{N}$ any number greater than $\frac{dM}{\varepsilon}$. Then for all $n > N$ we have

$$\left|\frac{f(n)}{g(n)} - a_d\right| = \left|\sum_{k=0}^{d-1} \frac{a_k}{n^{d-k}}\right| \leq \sum_{k=0}^{d-1} \left|\frac{a_k}{n^{d-k}}\right| \leq \frac{\sum_{k=0}^{d-1} |a_k|}{n} = \frac{dM}{n} \leq \frac{dM}{N} \leq \varepsilon.$$

Hence by definition, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a_d < \infty$ and so $f \in \mathcal{O}(g)$ by Theorem 1. $\square$

Part a) $n^3 - 8n \in \mathcal{O}(n^3)$, by Theorem 2.

Part b) $5n^4 + 2n^2 + 8 \in \mathcal{O}(n^4)$, by Theorem 2.

Part c) $(n^2 + n - 1)(8n + 5) = 8n^3 + 13n^2 - 3n - 5 \in \mathcal{O}(n^3)$, by Theorem 2.

Part d) $8n(6n + 1) = 48n^2 + 8n \in \mathcal{O}(n^2)$, by Theorem 2.

Problem 3

(10 points) Suppose that L is a sorted list of 1024 elements. Using the binary search algorithm, what is the maximum number of comparisons made by binary search to determine if an item is in L? Explain how this is related to the time complexity of the binary search algorithm.

Solution

Lemma. Let $k > 1$ be an integer and let L_m be any sorted list of m elements. Then checking if $n \in L_{2^k}$ requires at most $k + 1$ comparisons.

Proof. Without loss of generality, let $L_m = \{1, 2, \ldots, m\}$ for any $m \in \mathbb{N}$. The number of comparsions required to check if $n \in L_m$ is maximized when $n \notin L_m$, so suppose $n > m$ in all cases that follow.

We prove the lemma by induction on k. The base case is when $k = 2$. Then $L_{2^k} = L_4 = \{1, 2, 3, 4\}$. Here the midpoint index $m = (3 - 0)//2 = 1$ and

$$L_4[1] = 2 < n.$$

Hence we bisect and take $L = L_4[2 ::] = 3, 4$. Here the midpoint is $(1 - 0)//2 = 0$ and

$$L[0] = 3 < n.$$

Again we bisect and take $L = 4$. Here the midpoint is $(0-0)//2 = 0$ and

$$L[0] = 4 \neq n$$

and the program terminates. This required $3 = 2+1 = k+1$ function calls. This completes the base case.

Now we assume by way of induction that it takes at most $k+1$ comparisons to search for an element in $L_{2^k} = \{1, 2, \ldots, 2^k\}$ and consider the maximum number of comparisons required to search for an element in $L_{2^{k+1}} = \{1, 2, \ldots, 2^k, 2^k + 1, \ldots, 2^{k+1}\}$.

Again we assume without loss of generality that $n > 2^{k+1}$ and run a binary search on the statement $n \in L_{2^{k+1}}$. Here the midpoint index is $m = (k^{k+1} - 1 - 0)//2 = 2^k - 1$, and

$$L_{2^{k+1}}[2^k - 1] = 2^k < n.$$

Bisecting gives $L = L_{2^{k+1}}[2^k - 1 ::] = \{2^k + 1, 2^k + 2, \ldots, 2^{k+1}\}$. But by the inductive hypothesis, L requires $k+1$ comparisons, and so bisection on $L_{2^{k+1}}$ requires at most $1 + (k+1) = (k+1) + 1$ comparisons. This completes the proof. $\square$

Since $1024 = 2^{10}$, by the lemma it takes at most $10 + 1 = 11$ comparisons to determine if an element is in L using a binary search. Further, since $k = \log_2(2^k)$, the lemma implies that the time complexity of binary search on a list of length $n \in \mathbb{N}$ is $\mathcal{O}(\log n)$.

Problem 5

(10 points) With a sorted list of 1024 elements, what is the maximum number of recursive calls made to the binary search algorthm to find an element? Explain how this is related to the time complexity of the binary search algorithm.

Solution

The number of recursive calls made to the binary search algorithm is one less the number of comparisons made in the iterative implementation (since we do not make a recursive call if the length of the list is 1). By Problem 3, the number of recursive calls is $(\log_2(n) + 1) - 1 = \log_2(n)$. In this case, $\log_2(1024) = 10$, so there are 10 recursive calls. Similar to Problem 3, this implies that the time complexity of the algorithm is $\mathcal{O}(\log n)$.