Assignment 1.2

Bennet Montgomery Student No. 20074049

2019-01-27

1 Big-O, Big-Ω, Big-Θ Analysis and Proofs

Q1

Since the highest ordered term in the polynomial has exponent 2, we can assume that $T(n) = O(n^2)$ is a reasonable guess. To prove our estimate, we must find some c for which $T(n) \leq cn^2$:

$$\frac{3}{2}n^2 + \frac{5}{2}n - 3 \leq cn^2$$
$$\frac{3}{2} + \frac{5}{2n} - \frac{3}{n^2} \leq c$$

The second two terms of the left of the above inequality approach 0 as n approaches ∞. Because of this, we can see that at some n_0, cn^2 passes $T(n)$ and increases at a faster rate for any value $c, c > \frac{3}{2}$, implying $T(n) = O(n)$.

Q2

If our answer for 1 is correct, than $T(n)$ must be tightly bound by cn^2, that is to say $T(n) = \Theta(n)$. Since we have already proved that $T(n) = O(n^2)$, we need only prove that $T(n) = \Omega(n^2)$ as well in order to show that $T(n)$ is $\Theta(n^2)$:

$$cn^2 \leq \frac{3}{2}n^2 + \frac{5}{2}n - 3$$
$$c \leq \frac{3}{2} + \frac{5}{2n} - \frac{3}{n^2}$$

As we noted above, the terms $\frac{5}{2n}$ and $-\frac{3}{n^2}$ approach 0 as n approaches ∞. This means that at some point n_0, $T(n)$ passes and accelerates at a higher rate than cn^2 for any c such that $c < \frac{3}{2}$. Since we have shown that $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$, we can conclude that $T(n) = \Theta(n^2)$, implying n^2 is the tightest upper bound.

Q3

In order for $T(n)$ to be $O(n \log n)$, there must be some value c where:

$$T(n) \leq cn \log n$$
$$3n \log n + 4 \log n + 2 \leq cn \log n$$
$$3 + \frac{4}{n} + \frac{2}{n \log n} \leq c$$

Since the terms $\frac{4}{n}$ and $\frac{2}{n \log n}$ approach 0 as n approaches ∞, c is any value such that $c > 3$. Since we have shown $\exists c, T(n) \leq cnlogn$, we can confidently say $T(n) = O(n \log n)$.

Q4

If $max(f(n) + g(n) = \Theta(f(n) + g(n))$, that implies that there exists two values c_1 and c_2 such that (assuming $f(n)$ is the maximum function):

$$c_1(f(n) + g(n)) \leq f(n) \leq c_2(f(n) + g(n))$$
$$c_1 \leq \frac{f(n)}{(f(n) + g(n))} \leq c_2$$

Since $f(n)$ and $g(n)$ are non-negative $\frac{f(n)}{f(n)+g(n)}$ is non-negative, therefore there are 2 real values c_1 and c_2 such that $c_1(f(n) + g(n)) \leq f(n) \leq c_2(f(n) + g(n)), c_1 > 0, c_2 > 0$. this proof can be easily extended to the case where $g(n)$ is the maximum function.

Q5

The answer is clearly TRUE. If $2^{n+1} = O(2^n)$, then there exists some real value c such that:

$$2^{n+1} \leq c2^n$$
$$2 \cdot 2^n \leq c2^n$$
$$2 \leq c$$

Since c is real and is any value such that $c \geq 2$, we can confidently say $2^{n+1} = O(2^n)$.

Q6

The answer is solidly FALSE. Assuming $2^{2n} = O(2^n)$, there must be some value c such that:

$$2^{2n} \leq c2^n$$
$$2^{n+n} \leq c2^n$$
$$2^n \cdot 2^n \leq c2^n$$
$$2^n \leq c$$

The inequality above is not possible for any value of c. Since c is a constant and n is a variable approaching ∞, there is no value of c that is not eventually less than 2^n.

2 Runtime Analysis of Programs

Abstract

This study was conducted to find the relationship between list to search size and target set size and the comparative time efficiencies of the binary and linear search algorithms. The study was conducted by attempting to find a value k, k = target set size for which binary search beat linear search in speed on lists to search of varying size. In all list to search sizes, binary search eventually beat linear search in time efficiency, supporting the theoretical time efficiencies of each algorithm found via algorithm analysis.

Introduction

The purpose of this study was to determine the effect that the number of elements in a list and the number of target elements to search for have on the comparative speeds of the linear search and binary search algorithms. Linear search is a searching algorithm where each element of the data structure to search are iterated through exhaustively until the target element is found or all elements of the data structure have been searched. The time complexity of linear search is $O(n)$.[1] Binary search is a sorting algorithm of time complexity $O(logn)$ which can only be used on a presorted list that checks the middle element of the list to search. If the middle element is less than the target, the algorithm performs the binary search on the second half of the list. If the middle element is greater than the target, the algorithm performs the binary search on the first half of the list.[2] Since binary search has a time complexity of $O(logn)$, we expect binary search to eventually surpass linear search in efficiency for some value k where k is the number of targets to search for regardless of the size of the list to search.

Methods

A python program was created (see assignment1-2.py) which contained functions for binary search and linear search. The binary search consisted of a merge sort of the list to search, followed by the traditional binary search algorithm of an ordered list. A list was then passed to the linear search and binary search methods of size 1000, and the speeds of the two algorithms were compared. Speed comparison was done by generating a target list of size k, where half the elements of the target list were in the list to search and the other half were not; k was iterated until the binary search algorithm was recorded as faster than the linear search algorithm for the two lists. This test was performed 1000 times, and the average k threshold was recorded. The above steps were repeated for lists to search of size 2000, 5000, and 10000.

Results

In all cases, binary search passed linear search in efficiency for some k:

Size of List to Search vs k Threshold Values

Size Of List To Search (n)	Search Target Threshold for Binary Search Beating Linear Search (k)
1000	189.418
2000	203.213
5000	218.380
10000	232.014

Discussion

As expected, the binary search surpassed the linear search in time efficiency for all list sizes as target size increased. This was expected as binary search is $\Theta(\log n)$, and linear search is $\Theta(n)$, meaning we expect there to be some x_0 where binary search permanently surpasses linear search in efficiency $\forall x, x > x_0$, the experimental value threshold k being x_0 for each list. There is a clear trend of k increasing as list size increases. A potential explanation for this is that linear search may be generally more time efficient than binary search when the ratio between list to search size and number of search targets is very high.

References

1. No author. Linear search. Geeks for geeks [Internet]. No date [cited 2019-01-27]. https://www.geeksforgeeks.org/linear-search/

2. No author. Binary search. Geeks for geeks [Internet]. No date [cited 2019-01-27]. https://www.geeksforgeeks.org/binary-search/