CSE 317: Design and Analysis of Algorithms

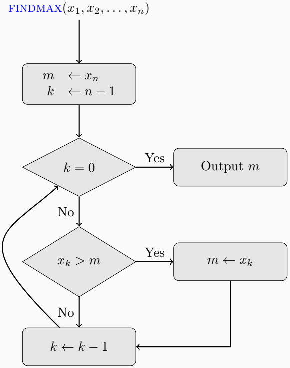
Shahid Hussain

Weeks 1 & 2: Fall 2024

Introduction

Finding the maximum from a sequence

- Given a sequence of numbers $x_1, \ldots, x_n$, we want to find the maximum
- Consider the following algorithm



Finding the maximum from a sequence

Algorithm: FINDMAX

Input: A sequence of numbers $x_1, \ldots, x_n$

Output: The maximum number m in the sequence

1. $m = x_k$
2. $k = n - 1$
3. **while** $k > 0$
4.   **if** $x_k > m$
5. $m = x_k$
6. $k = k - 1$
7. **return** m

Let $m = 4$ and $x_1 = 2.3$, $x_2 = 7.1$, $x_3 = 4.0$, and $x_4 = 5.9$. We can trace the algorithm (flow chart) with this data. Following is the trace

Finding the maximum from a sequence

- Let $m = 4$ and $x_1 = 2.3$, $x_2 = 7.1$, $x_3 = 4.0$, and $x_4 = 5.9$. We can trace the algorithm (flow chart) with this data. Following is the trace

$$m \leftarrow \cancel{5.9}, \underline{7.1}$$
$$k \leftarrow \cancel{3}, \cancel{2}, \cancel{1}, \underline{0}$$

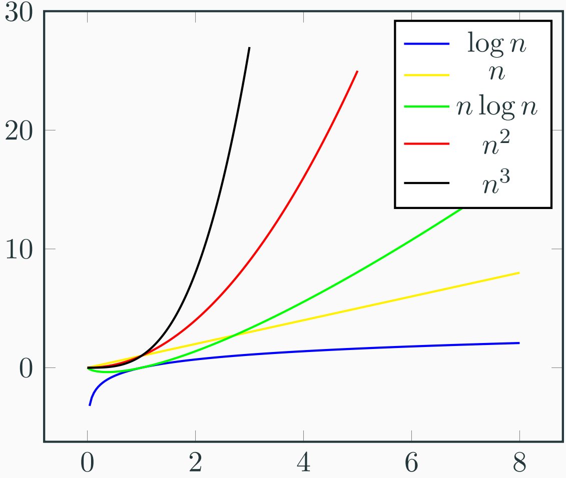
- Is this algorithm correct?
- We employ invariant
- For the above algorithm we can place the invariant $m = \max(x_{k+1}, \ldots, x_n)$ before the first decision (the if statement) and it will always remain true

Time Complexity

Time Complexity

- Time in an important resource while designing algorithms
- It is meaningless to say that an Algorithm A, when presented with input x, runs in time y seconds
- We, therefore, measure time in number of elementary operations such as arithmetic operations (addition and subtraction), comparisons, etc

Growth of Functions



Oh and Other Notations

The Big-Omicron Notation

- Let f and g be two functions, $f, g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is in big-omicron of $g(n)$, i.e., $f(n) = O(g(n))$, if and only if, there exists constants c and n_0 such that

$$f(n) \leq c \cdot g(n), \qquad \text{for all } n \geq n_0$$

- Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ exists, then

$$f(n) = O(g(n))$$

The Big-Omega Notation

- Let f and g be two functions, $f, g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is in big-omega of $g(n)$, i.e., $f(n) = \Omega(g(n))$, if and only if, there exists constants c and n_0 such that

$$f(n) \geq c \cdot g(n), \qquad \text{for all } n \geq n_0$$

- Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ exists, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \text{ implies } f(n) = \Omega(g(n))$$

- Informally, $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$

The Big-Theta Notation

- Let f and g be two functions, $f, g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is in big-theta of $g(n)$, i.e., $f(n) = \Theta(g(n))$, if and only if, there exists constants c_1, c_2, and n_0 such that

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \qquad \text{for all } n \geq n_0$$

- Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ exists, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \text{ implies } f(n) = \Theta(g(n))$$

- Importantly, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

The small-omicron Notation

- Let f and g be two functions, $f, g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is in small-omicron of $g(n)$, i.e., $f(n) = o(g(n))$, if and only if, there exists a constant n_0 such that for all possible values of $c > 0$

$$f(n) < c \cdot g(n), \qquad \text{for all } n \geq n_0$$

- Consequently, the $\lim_{n\to\infty} f(n)/g(n)$ exists and equal to zero i.e.,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \ \text{ implies } f(n) = o(g(n)).$$

The small-omega Notation

- Let f and g be two functions, $f, g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is in small-omega of $g(n)$, i.e., $f(n) = \omega(g(n))$, if and only if, there exists a constant n_0 such that for all possible values of $c > 0$

$$f(n) > c \cdot g(n), \qquad \text{for all } n \geq n_0$$

- Consequently, the $\lim_{n\to\infty} f(n)/g(n)$ is infinite i.e.,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \text{ implies } f(n) = \omega(g(n)).$$

Examples

1. Let $f(n) = 10n^3 + 20n$ then $f(n) = O(n^3)$ also $f(n) = \Omega(n^3)$ therefore $f(n) = \Theta(n^3)$
2. In general, let $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$, then $f(n) = \Theta(n^k)$
3. Let $f(n) = \log_b n^2$ then $f(n) = 2 \log_b n$ therefore $f(n) = \Theta(\log n)$
4. For any fixed constant k, $\log n^k = \Theta(\log n)$
5. Any constant function is $O(1)$, $\Omega(1)$, and $\Theta(1)$
6. $2^n = \Theta(2^{n+1})$, this is an example of *many* functions that satisfy $f(n) = \Theta(f(n+1))$

More Examples

1. Consider the series $\sum_{j=1}^{n} \log j$, clearly:

$$\sum_{j=1}^{n} \log j \leq \sum_{j=1}^{n} \log n = O(n \log n)$$

2. We also prove that $\sum_{j=1}^{n} \log j = \Omega(n \log n)$, therefore:

$$\sum_{j=1}^{n} \log j = \Theta(n \log n)$$

Problems: State True or False

1. $n^3 = O(n^2)$
2. $n \log n = O(n\sqrt{n})$
3. $n^2(1 + \sqrt{n}) = O(n^2 \log n)$
4. $\log n + \sqrt{n} = O(n)$
5. $\sqrt{n} \log n = O(n)$
6. $\log n = O(1/n)$
7. $n + \sqrt{n} = O(\sqrt{n} \log n)$

Problems: Proof Questions

1. $n^2 - 3n - 18 = \Omega(n)$
2. $2^n = O(n!)$
3. $n! = \Omega(2^n)$
4. Does $n^{\log n} = O((\log n)^n)$? Prove it
5. Prove or disprove: $2^{(1+O(1/n))^2} = 2 + O(1/n)$

Searching

Linear Search

- Given a sequence A of $n \geq 0$ pair-wise different numbers $a_1, \ldots, a_n$, and a number x
- We want to find the index i such that $a_i = x$

Algorithm: LINEARSEARCH

Input: A sequence of numbers $a_1, \ldots, a_n$ and a number x

Output: The index i such that $a_i = x$, 0 otherwise

```
for j = 1 to n
    if a_j = x return j
return 0
```

Linear Search

- The Algorithm LINEARSEARCH is correct, *why?*
- The time complexity of the algorithm is $O(n)$, *why?*
- The space complexity of the algorithm is $O(1)$, *why?*
- This algorithm gets its name because of its worst-case running time, which is *linear* in terms of the size of its input
- This algorithm does not depend on the order of the elements in the sequence

Binary Search

- Given a sequence A of $n \geq 0$ pair-wise different numbers $a_1, \ldots, a_n$, such that $a_1 < a_2 < \cdots < a_n$, and a number x
- We want to find the index i such that $a_i = x$
- Now we can exploit the fact that the sequence is *sorted*

Binary Search

Algorithm: BINARYSEARCH

Input: A sequence of numbers $a_1, \dots, a_n$ and a number x

Output: The index i such that $a_i = x$, 0 otherwise

Iterative Version

1. $l = 1$, $r = n$
2. **while** $l \leq r$
3. $\quad m = \lfloor (l + r)/2 \rfloor$
4. $\quad$ **if** $a_m = x$ **return** m
5. $\quad$ **if** $a_m < x$ $l = m + 1$
6. $\quad$ **else** $r = m - 1$
7. **return** 0

Recursive Version

1. $m = \lfloor (l + r)/2 \rfloor$
2. **if** $a_m = x$ **return** m
3. **else if** $a_m > x$ **return** BINARYSEARCH$(a_1, \dots, a_{m-1}, x)$
4. **else return** BINARYSEARCH$(a_{m+1}, \dots, a_n, x)$

Binar Search

- The Algorithm binarysearch is correct, *why?*
- The time complexity of the algorithm is $O(\log n)$, *why?*
- The space complexity of the algorithm is $O(1)$, *why?*

Sorting

Sorting

- Given a sequence A of $n \geq 0$ numbers $a_1, \ldots, a_n$
- We want to *sort* the sequence in *non-decreasing order* i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$
- There are several ways to sort a sequence, some are more efficient than others
- Here we look at some of the comparison based sorting algorithms

Selection Sort

Algorithm: SELECTIONSORT

Input: A sequence of numbers $A = \langle a_1, \ldots, a_n \rangle$

Output: A permutation of A such that $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

1. **let** $B = \langle\rangle$
2. **for** $i = 1$ **to** n
3. $\quad m = \text{FINDMIN}(a_i, \ldots, a_n)$
4. $\quad B = B \circ \langle a_m \rangle$
5. $\quad a_m = a_i$
6. **return** B

- The FINDMIN makes $n - i$ comparisons to find the minimum
- The time complexity of SELECTIONSORT is therefore $n + (n-1) + \cdots + 1 = \Theta(n^2)$

Insertion Sort

Algorithm: INSERTIONSORT

Input: A sequence of numbers $A = \langle a_1, \ldots, a_n \rangle$

Output: A permutation of A such that $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

1. **for** $i = 2$ **to** n
2. $\quad j = i$
3. $\quad$ **while** $j > 1$ **and** $a_j < a_{j-1}$
4. $\quad\quad a_j, a_{j-1} = a_{j-1}, a_j$
5. $\quad\quad j = j - 1$
6. **return** A

- The time complexity of INSERTIONSORT is $\Theta(n^2)$

Insertion Sort with Binary Search

Algorithm: BINARY-INSERTIONSORT

Input: A sequence of numbers $A = \langle a_1, \ldots, a_n \rangle$

Output: A permutation of A such that $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

for $i = 2$ **to** n
    $j = i$, $l = 1$, $r = j$
    while $l \leq r$
        $m = \lfloor (l + r)/2 \rfloor$
        if $a_j < a_m$ **then** $r = m - 1$
        else $l = m + 1$
    $a_j, a_{j-1} = a_{j-1}, a_j$
return A

- The time complexity of BINARY-INSERTIONSORT is $\Theta(n^2)$ (but slightly better than INSERTIONSORT)

Bubble Sort

Algorithm: BUBBLESORT

Input: A sequence of numbers $A = \langle a_1, \ldots, a_n \rangle$

Output: A permutation of A such that $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

1. **for** $i = 1$ **to** n
2.   **for** $j = 1$ **to** $n - i$
3.   **if** $a_j > a_{j+1}$ **then** $a_j, a_{j+1} = a_{j+1}, a_j$
4. **return** A

- The time complexity of BUBBLESORT is $\Theta(n^2)$ becasue

$$\text{number of comparisons} = \sum_{i=1}^{n} \sum_{j=1}^{n-i} 1 = \frac{n(n-1)}{2}$$