Application: Analysis of Algorithm Efficiency I

The Sequential Search Algorithm

In a sequential search, x is compared to the first item in the array, then to the second, then to the third, and so on. The search is stopped if a match is found at any stage. On the other hand, if the entire array is processed without finding a match, then x is not in the array.

Example 1: Best- and Worst Case Orders for Sequential Search

Find best-and worst-case words for the sequential search algorithmfrom among the set of power functions.

Solution

Suppose the sequential search algorithm is applied to an input array $a[1], a[2], \ldots, a[n]$ to find an item x. In the best case, the algorithm requires only one comparison between x and the items in $a[1], a[2], \ldots, a[n]$. This occurs when x is the first item in the array. Thus in the best case, the sequential search algorithm is $\Theta(1)$. (Note that $\Theta(1) = \Theta(n^0)$.) In the worst case, the algorithm requires n comparisons. This occurs when $x = a[n]$ or when x does not appear in the array at all. Thus in the worst case, the sequential search algorithm is $\Theta(n)$.

The Insertion Sort Algorithm

Insertion sort is an algorithm for arranging the items in an array into ascending order. Initially, the second item is compared to the first. If the second item is less than the first, their values are interchanged, and as a result the first two array items are in ascending order. The

$$\overbrace{a[1], a[2], a[3]. \ldots, a[k-1]}^{\text{sorted subarray}}, a[k], a[k+1], \ldots, a[n]$$

Step k: Insert the value of $a[k]$ into its proper position relative to $a[1], a[2], \ldots, a[k-1]$. At the end of this step $a[1], a[2], \ldots, a[k]$ is sorted.

Time Efficiency of an Algorithm

One factor algorithms can be calculated on is the size of the set of data that is input to the algorithm. Consequently, the execution time of an algorithm is generally expressed as a function of its input size.

Another factor that may affect the run time of an algorithm is the nature of the input data. A program that searches sequentially through a list of length n to find a data item requires only one step if the item is first on the list, but uses n steps if the item is last on the list. Thus algorithms are frequently analyzed in terms of their "best-case", "worst-case", and "average-case" performances for an input of size n.

The analysis of an algorithm for time efficiency begins by trying to count the number of elementary operations that must be performed when the algorithm is executed with an input of size n (in the best-case, worst-case, or average-case). What is classified as an "elementary operation" may vary depending on the nature of the problem the algorithm being compared are designed. For instance, to compare two algorithms for evaluating a polynomial, the crucial issue is the number of additions and multiplications that are needed, whereas to compare two algorithms for searching a list to find a particular element, the important distinction is the number of comparisions that are required. These are **elementary operations:** addition, subtraction, multiplication, division, and comparisons that are indicated explicitly in an if-statement using one of the relational symbols $<, \leq, >, \geq, =$, or $\neq$.

Definition

Let A be an algorithm.

1. Suppose the number of elementary operations performed when A is executed for an input of size n depends on n alone and not on the nature of the input data; say it equals $f(n)$. If $f(n)$ is $\Theta(g(n))$, we say that A is $\Theta(g(n))$ or A is of order $g(n)$.

2. Suppose the number of elementary operations performed when A is executed for an input of size n depends on the nature of the input data as well as on n.

(a) Let $b(n)$ be the <u>minimum number</u> of elementary operations required to execute A for all possible input sets of size n. If $b(n)$ is $\Theta(g(n))$. We say that in the best case, A is $\Theta(g(n))$ or A has a best-case of $g(n)$.

(b) Let $w(n)$ be the <u>maximum number</u> of elementary operations required to execute A for all possible input sets of size n. If $w(n)$ is $\Theta(g(n))$, we say that in the worst case, A is $\Theta(g(n))$ or A has a worst-case order of $g(n)$.

Example 2: Computing an Order of an Algorithm Segment

Assume n is a positive integer and consider the following algorithm segment

```
p := 0, x := 2
for i := 2 to n
   p := (p + i) · x
next i
```

a. Compute the actual number of additions and multiplications that must be performed when this algorithm segment is executed.

b. Use the theorem on polynomial orders to find an order for this algorithm segment.

Solution

a. There are one multiplication and one addition for each iteration of the loop, so there are as many multiplications and additions as there are iterations of the loop. Now the number of iterations of the `for-next` loop equalsthe top index of the loop minus the bottom index plus 1; that is, $n - 2 + 1 = n - 1$. Hence there are $2(n - 1) = 2n - 2$ multiplications and additions.

b. By the theorem on polynomial orders.

$$2n - 2 \text{ is } \Theta(n)$$

and so this algorithm segment is $\Theta(n)$

Example 3: An Order for an Algorithm with a Nested Loop

Assume n is a positive integer and consider the following algorithm segment:

```
s := 0
for i := 1 to n
   for j := 1 to i
      s := s + j · (i - j +1)
   next j
next i
```

a. Compute the actual number of additions and multiplications that must be performed when this algorithm segment is executed.

b. Use the theorem on polynomial orders to find an order for this algorithm segment.

Solution

a. There are two additions, one multiplication, and one subtraction for each iteration of the inner loop, so the total number of additions, multiplications, and subtractions is four time the number of iterations of the inner loop. Now the inner loop is iterated

$$\begin{aligned} \text{one time when } i &= 0 \\ \text{two times when } i &= 2 \\ \text{three times when } i &= 3 \\ &\vdots \\ n \text{ times when } i &= n \end{aligned}$$

Hence the total number of iterations of the inner loop is

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

and so the number of additions, subtractions, and multiplications is

$$4 \cdot \frac{n(n+1)}{2} = 2n(n+1)$$

b. By the theorem on polynomial orders, $2n(n+1) = 2n^2 + 2n$ is $\Theta(n^2)$, and so this algorithm segment is $\Theta(n^2)$.

Example 4: When the Number of Iterations Depends on the Floor Function

Assume n is a positive integer and consider the following algorithm segment:

```
for i := ⌊n/2⌋ to n
   a := n - i
next i
```

a. Compute the actual number of additions and multiplications that must be performed when this algorithm segment is executed.

b. Use the theorem on polynomial orders to find an order for this algorithm segment.

Solution

a. There is one subtraction for each iteration of the loop, and the loop is iterated $n - \lfloor \frac{n}{2} \rfloor + 1$. If n is even, then $\lfloor \frac{n}{2} \rfloor = \frac{n}{2}$, and so the number of subtractions is

$$n - \left\lfloor \frac{n}{2} \right\rfloor + 1 = n - \frac{n}{2} + 1 = \frac{n+2}{2}$$

If n is odd, then $\lfloor \frac{n}{2} \rfloor = \frac{n-1}{2}$, and so the number of subtractions is

$$n - \left\lfloor \frac{n}{2} \right\rfloor + 1 = n - \frac{n-1}{2} + 1 = \frac{2n - (n-1) + 2}{2} = \frac{n+3}{2}$$

b. By the theorem on polynomial orders,

$$\frac{n+2}{2} \text{ is } \Theta(n) \quad \text{and} \quad \frac{n+3}{2} \text{ is } \Theta(n)$$

also. Hence, regardless of whether n is even or odd, this algorithm segment is $\Theta(n)$

Algorithm: Insertion Sort The aim of this algorithm is to take an array $a[1], a[2], a[3] \ldots, a[n]$, where $n \geq 1$, and reorder it. The output array is also denoted $a[1], a[2], a[3] \ldots, a[n]$. It has the same values as the inut array, but they are in ascending order. In the kth step, $a[1], a[2], a[3] \ldots, a[k-1]$ is in ascending order, and $a[k]$ is inserted into the correct position with respect to it.

Input: n is a positive integer, $a[1], a[2], a[3] \ldots, a[n]$

Algorithm Body

```
for k := 2 to n
   x := a[k]
   j := k - 1
   while (j ≠ 0)
        if x < a[j] then
          a[j + 1] := a[j]
          a[j] := x
          j := j - 1
        else j := 0
        end if
   end while
next k
```

Output: $a[1], a[2], a[3] \ldots, a[n]$ in ascending order.

Example 7: Finding a Worst-Case Order for Insertion Sort

a. What is the maximum number of comparisons that are performed when insertion sort is applied to the array $a[1], a[2], a[3] \ldots, a[n]$?

b. Use the theorem on polynomial orders to find a worst-case order for insertion sort.

Solution

a. In each iteration of the **while** loop, two explicit comparisons are made: one to test whether $j \neq 0$ and the other to test whether $a[j] > x$. During the time that $a[k]$ is put into position relative to $a[1], a[2], a[3] \ldots, a[k-1]$, the maximum number of attempted iterations of the **while** loop is k. This happens when $a[k]$ is less than every $a[1], a[2], a[3] \ldots, a[k-1]$?; on the kth attempted iteration, the condition of the **while** loop is satisfied because $j = 0$. Thus the maximum number of comparisons for a given value of k is $2k$. Because k goes from 2 to n, it follows that the maximum total number of comparisons occurs when the items in the array are in reverse order, and it equals

$$\begin{aligned}(2 \cdot 2) + (2 \cdot 3) + \cdots + (2 \cdot n) &= 2(2 + 3 + \cdots + n) \\ &= 2\big[(1 + 2 + 3 + \cdots + n) - 1\big] \\ &= 2\Big(\frac{n(n+1)}{2} - 1\Big) \\ &= n(n+1) - 2 \\ &= n^2 + n - 2\end{aligned}$$

b. By the theorem on polynomial orders, $n^2 + n - 2$ is $\Theta(n^2)$, and so the insertion sort algorithm has worst-case order $\Theta(n^2)$.

Example 8: Finding an Average-Case Order for Insertion Sort

a. What is the average number of comparisons that are performed when insertion sort is applied to the array $a[1], a[2], a[3] \ldots, a[n]$?

b. Use the theorem on polynomial orders to find an average-case order for insertion sort.

Solution

a. Let E_n be the average, or expected, number of comparisons used to sort $a[1], a[2], a[3], \ldots, a[n]$ with insertion sort. Note that for each integer $k = 2, 3, \ldots, n$.

[the expected number of comparisons used to sort $a[1], a[2], a[3] \ldots, a[n]$] =

the expected number of comparisons used to sort $a[1], a[2], a[3] \ldots, a[n]$+

the expected number of comparisons used to place $a[k]$ into positions relative to $a[1], a[2], a[3] \ldots, a[k-1]$

Thus

$E_k = E_{k-1}+$

expected number of comparisons used to place $a[k]$ into positions relative to $a[1], a[2], a[3], \ldots, a[k-1]$

Also, $E_1 = 0$ because there is just one item in the array, $n = 1$ and no iterations of the outer loop are performed.

Now at the time $a[k]$ is placed to $a[1], a[2], a[3] \ldots, a[k-1]$, a reasonable assumption is that it is equally likely to belong in any one of the first k positions. Thus the probability of its belonging in any particular position is $1/k$. If it actually belongs in position j, then $2(k - j + 1)$ comparisons will be used in moving it, because there will be $k - j + 1$ attempted iterations of the **while** loop and there are 2 comparisons per attempted iteration.

According to the definition of expected value, the expected number of comparisons used to place $a[k]$ relative to $a[1], a[2], a[3], \ldots, a[k-1]$ is therefore

$$\begin{aligned}\sum_{j=1}^{k} \frac{1}{k} \cdot 2(k-j+1) &= \frac{2}{k}\big[k + (k-1) + \cdots + 3 + 2 + 1\big] \\ &= \frac{2}{k}\Big(\frac{k(k+1)}{2}\Big) \\ &= k+1\end{aligned}$$

Hence

$$E_k = E_{k-1} + k + 1 \qquad \text{for all integers } k \geq 2, \text{ and}$$
$$E_1 = 0$$

$$E_n = \frac{n^2 + 3n - 4}{2} \qquad \text{for each integer } n \geq 1$$

b. By the theorem on polynomial orders, $n = \frac{n^2+3n-4}{2} = \frac{1}{2}n^2 + \frac{3}{2}n - 2$ is $\Theta(n^2)$, and so the average-case order of insertion sort is also $\Theta(n^2)$