Algorithm Analysis

Madiba Hudson-Quansah

Contents

Chapter 1

Analysing Algorithms

Analysing an algorithm involves prediction the resources that the algorithm will require. The resources that are of interest are:

- Memory
- Computation Time
- Bandwidth
- Energy Consumption

In analysing an algorithm represented in pseudocode we can determine how long the algorithm takes to run by examining how many times each line of the code is executed and how long each line takes to execute. To do this we first derive a precise formula for the running time, then simplify the formula using a convenient notation that allows us to compare running times of different algorithms. For this example we will use the insertion-sort algorithm with the pseudocode defined below:

Algorithm 1 Insertion-Sort (A, n)

```
1: for index = 2 to n do
2:     prevIndex := index −1
3:     nextElement := A[index]                ▷ Insert A[index] into the sorted array A[1 : index − 1]
4:     while A[prevIndex] > nextElement and prevIndex ⩾ 0 do
5:         A[prevIndex +1] := A[prevIndex]
6:     end while
7:     A[prevIndex +1] = nextElement
8: end for
```

We first must acknowledge that the running time of an algorithm depends on the input which we will call A. Also we must acknowledge that the insertion-sort algorithm can take a different amount of time sorting two arrays of the same size depending on how already sorted the arrays are. We then describe the running time of an algorithm as a function of the size of the input n. To do so we must clearly define the terms "running time", "size of input", and be clear about what scenario we are considering, whether it be best-case, worst-case, or average-case.

The best description of the *size of input* depends on the problem being solved. In this case for a sorting algorithm the best notion of the size of input is the number of elements in the array to be sorted. The *running time* of an algorithm is the number of instructions and data accesses made by the algorithm, how this is accounted for differs from computer to computer but using the **RAM model** in which each simple operation takes a constant amount of time, we can generalize the running time of an algorithm as the number of comparisons and data movements made by the algorithm. Therefore in this framework each execution of the kth line of pseudocode takes c_k time units where c_k is a constant.

First for each $i = 2, 3, \ldots, n$, let t_i denote the number of times the `WHILE` loop test in line 5 is executed for that value of i. Below is a table of lines, the number of times each line is executed and cost of each line.

Line	Cost	Times
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{i=2}^{n} t_i$
5	c_5	$\sum_{i=1}^{n} (t_i - 1)$
6	c_6	$\sum_{i=1}^{n} (t_i - 1)$
7	c_7	$\sum_{i=1}^{n} (t_i - 1)$
8	c_8	$n-1$

Chapter 2

Exercises

Question 1

Describe an algorithm that takes as input a list of n integers and finds the number of negative integers in the list

Solution:

Algorithm 2 Count-Negative (A, n)

```
1: count := 0
2: for i := 1 to n do
3:     if A[i] < 0 then
4:         count = count + 1
5:     end if
6: end for
7: return count
```

Question 2

Describe an algorithm that takes as in put a list of n integers and produces as outputs the largest difference obtained by subtracting an integer in the list from the one following it.

Solution:

Algorithm 3 Largest-Difference (A, n)

```
1: largest := A_2 − A_1
2: for i := 2 to n − 1 do
3:     diff := A_{i+1} − A_i
4:     if  diff > largest then
5:         largest = diff
6:     end if
7: end for
8: return largest
```

Question 3

Solution:

Algorithm 4 Repeated-Ints (A, n)

```
1:  j := 0
2:  i := 2
3:  while i ⩽ n do
4:      if A_i = A_{i−1} then
5:          j := j + 1
6:          B_j := A_i                          ▷ B is the list of repeated values
7:          while i ⩽ n and A_i = C_i do
8:              i := i + 1
9:          end while
10:     end if
11:     i := i + 1
12: end while
13: return B
```