UWI

Design and Analysis of Algorithms

Tutorial 1 - Sample Solutions

Time Efficiency

In this course, emphasis is placed on analysing algorithms by their time efficiency. Knowing an algorithm's time efficiency can encourage us to make them more efficient by reducing the number of iterations to perform the algorithm's task.

1. What is time efficiency?

   Time efficiency analyses the number of repetitions of the basic operation as a function $T(n)$ of input size n.

2. What is a basic operation?

   An operation that contributes most towards the running time of an algorithm.

3. Given the following formula $T(n) \approx c_{op}C(n)$
   (a) What does n mean?
   The input size
   (b) What does c_{op} mean?
   The execution time for the basic operation (only important for empirical time analysis)
   (c) What does $T(n)$ mean?
   The running time of an algorithm
   (d) What does $C(n)$ mean?
   The number of times a basic operation is executed

4. Given that $C_b(n)$, $C_w(n)$ and $C_a(n)$ stand for the best, worst and average case time complexities.
   (a) What leads to the best case?
   The type of input that would make the count $C(n)$ will be the *smallest* among all possible inputs of size n.
   (b) What leads to the worst case?
   The type of input that would make the count $C(n)$ will be the *largest* among all possible inputs of size n.
   (c) What is meant by the average case?
   The count $C(n)$ that is expected from a run of an algorithm with truly random input.

5. Asymptotic orders of growth serve as ways of comparing the efficiency groups of algorithms.
   (a) What are the 3 asymptotic notations and what do they mean?
   Given that $t(n)$ and $g(n)$ are functions: (See pages 20-26 in topic 2's presentation for formal definitions)
      1. $O(g(n))$: set of functions $t(n)$ that grow no faster than $g(n)$.
      2. $\Omega(g(n))$: set of functions $t(n)$ that grow at least as fast as $g(n)$.
      3. $\Theta(g(n))$: set of functions $t(n)$ that grow at same rate as $g(n)$.
   (b) What are the 8 basic asymptotic efficiency classes? Indicate the most and least efficient ones.

1	constant	Best
$logn$	logarithmic	Excellent
n	linear	Good
$nlogn$	n-log-n	Okay
n^2	quadratic	Meh
n^3	cubic	Nah
2^n	exponential	Bad
$n!$	factorial	Worst

6. Analyse the following algorithms and identify their basic operations. Discuss their best and worst case time efficiencies.

   (a) Algorithm 1: Sequential Search

   for $i \leftarrow 0$ to n–1 **do**
   if $A[i] = K$ **then**
   return i
   end if
   end for
   return –1

   Best Case = $C_b(n) \in \Theta(1)$
   Worst Case = $C_w(n) \in \Theta(n)$

   (b) Algorithm 2: Max Element

   $maxval \leftarrow A[0]$
   for $i \leftarrow 1$ to n–1 **do**
   if $A[i] > maxval$ **then**
   $maxval \leftarrow A[i]$
   end if
   end for
   return $maxval$

   Best Case = Worst Case = Average Case = $C(n) \in \Theta(n)$

Mathematical Proofs

In this course, similar emphasis is also placed on mathematical proofs. This is because they serve as a foundation for some subsequent algorithmic proofs.

1. Prove that $\lfloor log_2(n) \rfloor + 1 = \lceil log_2(n+1) \rceil$, where n is a positive integer.

   Proof:

   From the left hand side of the given equation, we know that n is bounded both above and below by two powers of 2. These two powers of two must also be consecutive due to the nature of logarithms. This means that $2^k \leq n < 2^{k+1}$, where k is a non-negative integer that represents a power of two and is uniquely defined by the value of n.

   Now if we apply log_2 to $2^k \leq n < 2^{k+1}$, we get the following:

$$log_2 2^k \leq log_2 n < log_2 2^{k+1} \tag{1}$$
$$klog_2 2 \leq log_2 n < (k+1)log_2 2 \tag{2}$$
$$k(1) \leq log_2 n < (k+1)(1) \tag{3}$$
$$k \leq log_2 n < k+1 \tag{4}$$

   As a result, this means that $k \leq \lfloor log_2 n \rfloor$. Since $\lfloor log_2 n \rfloor \leq log_2 n < k+1$, we can derive the inequality $k \leq \lfloor log_2 n \rfloor < k+1$. This then implies that $k = \lfloor log_2 n \rfloor$.

Now using an argument similar to the one used to show that $k = \lfloor log_2 n \rfloor$, we can show that $\lceil log_2(n+1) \rceil = k+1$. If we apply log_2 to $2^k < n+1 \leq 2^{k+1}$, we get the following:

$$log_2 2^k < log_2(n+1) \leq log_2 2^{k+1} \tag{5}$$
$$k log_2 2 < log_2(n+1) \leq (k+1) log_2 2 \tag{6}$$
$$k(1) < log_2(n+1) \leq (k+1)(1) \tag{7}$$
$$k < log_2(n+1) \leq k+1 \tag{8}$$

As a result, this means that $\lceil log_2(n+1) \rceil \leq k+1$. Since $k < log_2(n+1) \leq \lceil log_2(n+1) \rceil$, we derive the inequality $k < \lceil log_2(n+1) \rceil \leq k+1$. This implies that $k+1 = \lceil log_2(n+1) \rceil$.

Lastly, since $k = \lfloor log_2 n \rfloor$ and $k+1 = \lceil log_2(n+1) \rceil$, then $\lfloor log_2 n \rfloor + 1 = \lceil log_2(n+1) \rceil$, as was required to be proven.