Student's Manual for Programming Methodology

with C++

Paul Kim

powered by LaTeX 2ε

Contents

Part I

Algorithms

Chapter 1

Time Complexity Analysis

1. The running time of an algorithm is relevent to the amount of input. Therefore the running time is a function of the amount of input: $T(n)$
2. Definitions of Time Complexity: with a positive constant c,
   1) Big-O: $T(n) \geq c \times f_O(n) \Rightarrow T(n) = O(f_O(n))$ Best-case scenarios can be described via Big-O functions.
   2) Big-Omega: $T(n) \leq c \times f_\Omega(n) \Rightarrow T(n) = \Omega(f_\Omega(n))$ Worst-case scenarios can be described via Big-Omega functions.
   3) Big-Theta: $T(n) \geq c \times f(n)$ and $T(n) \leq c' \times f(n) \Leftrightarrow T(n) = O(f(n)) = \Omega(f(n)) \Rightarrow T(n) = \Theta(f(n))$ Best- and Worst-case scenarios are the same in Big-Theta functions.
   4) Small-O: $T(n) = O(f_o(n)) \neq \Theta(f_o(n)) \Rightarrow T(n) = o(f_o(n))$

   Constants are ignored, and only the highest degree of the polynomial's monomials are relevant to Time Complexity Analysis.
3. Running Time Calculations
   1) Summations for Loops: One loop sequence of running time $f(i)$ is equivalent to:

   $$T(n) = \sum_{i=1}^{n} f(i)$$

   Two loop sequences of running time $f(i,\ j)$ is equivalent to:

   $$T(n) = \sum_{j=1}^{n} \sum_{i=1}^{n} f(i,\ j)$$

   2) Selective Controls: Worst-case scenario, $T(n) = \max(T_1(n),\ T_2(n), \cdots)$. Best-case = minimum.
   3) Recursion: $T(n) = f(T(n'))$ (점화식)

Chapter 2

Objective 1: Finding the Maximum Subarray Sum

The objective of this challenge is to find the maximum value of the sum of elements in a subarray of a given array. If all integers are negative, said maximum value is the sum of a subarray equivalent to the 'empty set', which is zero.

2.1 Cubic Brute Force Algorithm

```
6  int max_sum1(int* arr, int arrsize) {
7      int maxSum = 0;
8      for (int i = 0; i < arrsize; i++) {
9          for (int j = i; j< arrsize; j++) {
10             int thisSum = 0;
11             for (int k = i; k <= j; k++) thisSum += arr[k];
12             if (maxSum < thisSum) maxSum = thisSum;
13         }
14     } return maxSum;
15 }
16
```

2.2 Quadratic Brute Force Algorithm

2.3 Divide and Conquer

2.4 Kadane's Algorithm: A Linear, Incremental Solution

Part II

C++