Numerical Computation

Ege Emir Ozkan

Chapter 1

Sorting

1.1 Selection Sort

Take this from Oguzhan

1.2 Merge Sort

1.2.1 Merging Two Sorted Arrays

Take this from Oguzhan

1.2.2 Sorting using Mergesort

Figure 1.1: Mergesort Algorithm in C

```
void mergesort(int n, int a[]) {
  int *aux = new int[n];
  for (int i = 0; i < n; ++i) {
    aux[i] = a[i]
  }
  mergesort_r(a, aux, 0, n-1);
  delete [] aux;
}

void mergesort_r(int a[], int b[], int l, int r) {
  if (r <= l) {
    return;
  }
  int m = (l + r)/2;
  mergesort_r(b, a, l, m);
  mergesort_r(b, a, m + 1, r);
  merge(a + l, m - l + 1, b + l, r - m, b + m, 1)
}
```

Chapter 2

Asymptotic Analaysis

Instead of saying $f(n) \in \Theta(g(n))$, we say $f(n) = \Theta(g(n))$

2.1 Asymptotic Notation

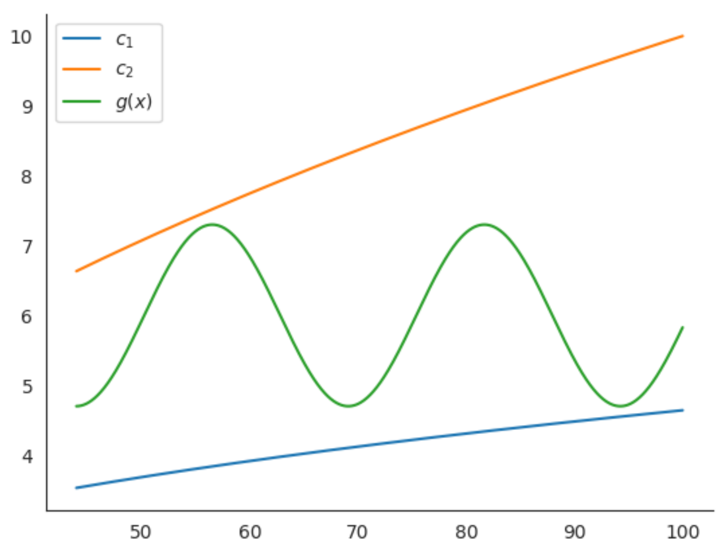


Figure 2.1: Complexity of functions

Functions defined on $IN = \{0, 1, 2, 3, ...\}, f(n) = an^2+bn+c \to \Theta(n^2) asn \to \infty$ **given** $g(n), \Theta(g(n))$ **is the set** $\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0$ **such that** $0 \leq c_1 g(u) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}$

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

$$c_1 n^2 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$c_1 \leq \frac{n-6}{2n} \leq c_2$$

For example, $6n^3 \neq \Theta(n^2)$ basically, all this boils down to:

$$p(n) = \sum_{i=0}^{d} a_i n^i = \Theta(n^d) \tag{2.1}$$

2.2 O notation (big-Oh)

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)), \Theta(g(n)) \subseteq O(g(n)) \tag{2.2}$$

$g(n)$ is an asymptotic upper bound for $f(n)$, If an algorithm such as insertion sort has different best and worst cases, such as for insertion sort $\Theta(n)$ best case and $\Theta(n^2)$ for worst case, we say it has an $O(n)$, such that for an algorithm f, if $f = \Theta(k(x))$ for worst case and $f = \Theta(g(x))$ for best case, we say that $f = O(k(x))$

2.2.1 Growth of an algorithm

2.2.2 Analysis of Recursive Algorithms

Reccurance equation describes the runtime of rectursive algorithm in terms of smaller algorithms.

$$T(n) = \begin{cases} \Theta(1) & n \leq C \\ aT\left(\frac{n}{b}\right) + D(u) + C(u) & \text{otherwise} \end{cases}$$

Where D is divide time and C is combine time.

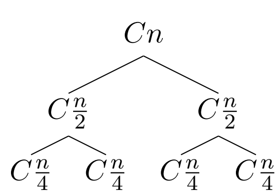
Analaysis of Mergesort

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(1) + \Theta(n) & n > 1 \end{cases} \tag{2.3}$$

And therefore, the computational complexity of merge sort is

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \tag{2.4}$$

The Recursion Tree Method



From here, it is easy to see each level has a total complexity of Cn, and the recursion tree is of depth $\log_2(n)$ therefore the whole program has a total complexity of $\Theta(n \log_2(n))$, this can also be proven from a method where one would solve the recursion equation **??**.

Basic Reccurances

For a loop eliminating a single item, calculations end up to $\Theta(n^2)$, as for a recursive program that halves the input in constant time.