Chapter 2

Mathematics

Standard notion and common functions

Mathematical background for Analysis of Algorithms

- The analysis of algorithms requires the need to draw upon a body of mathematical operations. Some of these operations are as simple as high school algebra while others are abit complex
- Along with learning how to manipulate asymptotic notations and solving recurrences, there is the need to learn other concepts and methods in order to analyze algorithms
- Examples of these concepts and methods include:-

- Methods for evaluating bounding summations
- Sets, relations, functions, graphs and trees
- Counting: permutations and combination
- Probability and statistics

Summations

- When an algorithms contains an iterative control construct for instance **while** and **for** loop its running time can be expressed as the sum of the times spent on each execution of the body of the loop.
- **Summation and product formulas**
- Given a sequence of numbers a1, a2,...,an, the finite sum of those numbers can be written as
-

$$\sum_{k=1}^{n} a_k$$

- (if n = 0, the summation is defined to be 0).

Summations formulas and properties

- Arithmetic Series

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{1}{2} n(n+1) = \Theta\left(n^2\right)$$

- Geometric or exponential Series

$$\sum_{k=1}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

  - For real $x \neq 1$

Summations formulas and properties

- **Sums of Squares and Cubes**

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$

Bounding summations Methods

- Most of the summations cannot be computed precisely when determining the run time of algorithms with iterative controls. In such cases there is need to find asymptotic upper and lower bound for the summation.
- In the ideal case, a Θ() bound is used.
- There are many techniques for bounding summations

➢Mathematical Induction

- This is the most basic way to evaluate a series
- For example: prove the bound for the following equation
- $\sum_{k=1}^{n} k^2 = \frac{n(2n+1)(n+1)}{6}$

➢Bounding the terms

- This is a method of bounding summations done by using the largest (smallest) value of terms to bound others
- a1 + a2 + . . . an ≤ amax + amax + . . . + amax = n · amax
- a1 + a2 + . . . an ≥ amin + amin + . . . + amin = n · amin

- Example

➢Splitting summations

- Better bounds can be obtained by partitioning the range of index and then bounding each of the results series

Summation conclusion

- Given a sum to calculate the bound for it. You do the following
  - Express it in terms of basic summations for which you know the bound
  - Guess the bound and prove it by induction
  - Obtain upper and lower bounds by bounding terms and/or splitting

Sets

- A set is a collection of distinguishable objects; called members or elements
- A set cannot contain the same element more than once; a variation of this is called multiset
- Given a set say S and element x then we can write:
-

$x \in S$

- (read as "x is in S")
-
- OR
- $x \notin S$
- (read as "x is not in S")

Frequently used set

- θ

  the empty set
- Z

  set of integers {…, -2,-1,0,1,2,…}
- R

  set of real numbers
- N

  set of natural numbers {0,1,2,…}

- **Recurrence Relations**
- Algorithms can be analyzed through a direct mapping from a recursive representation of a program to a recursive representation of a function describing its properties.
- Recurrence is used to efficiently calculate the quantity in a question, it is done by first computing the small values in order to get a feeling of how the values are growing.

Probability and statistics

- Probability is an essential tool for design and analysis of probabilistic and randomized algorithms.
- Probability is defined in terms of a sample space say S
- For instance when determining the expected runtime of a comparison based sort algorithm, the following steps would be followed to solve the problem

- Define the distribution over all possible inputs to the algorithm
- For example N! permutations of ordering N distinct numbers are equally likely $\frac{1}{N!}$
- Determine the run time for each possible input
- Compute the expected runtime by summing up over all possible inputs the probability of the input times the runtime for the input
- $E[Runtime] = sumi = 1 \ldots, N! \{P(i) * Runtime(i)\} = sumi = 1, .., N! \{Runtime(i)\}/N!$

- If the above steps are followed while analysing quicksort algorithm ignoring the pivot the expected runtime of quicksort is $\boldsymbol{O(N\ log\ N)}$ while the **worst-case** runtime is $\boldsymbol{O(N^2)}$

Monotonicity:

- Monotonically increasing function f(n);

  If $m \leq n$ implies that $f(m) \leq f(n)$

- Monotonically decreasing function f(n);

  If $m \geq n$ implies that $f(m) \geq f(n)$

- Strictly increasing function f(n);

  If $m < n$ implies that $f(m) < f(n)$

- Strictly decreasing function f(n);

  If $m > n$ implies that $f(m) > f(n)$

Floors and Ceilings

- The floor and ceiling function give the nearest integer up and down
- Example: what is the floor and ceiling of 5.6
- The floor of 5.6 is 5
- The ceiling off 5.6 is 6
- For integer the floor and ceiling does not change
- For example : the floor and ceiling of 3 is 3
- **Symbols**
- The symbols for the floor and ceiling are like the square brackets with the top or bottom part missing

- Floor function-$\lfloor x \rfloor$
- Ceiling function- $\lceil x \rceil$
- The word form for the symbol is
- **Floor(x)**
- **Ceil(x)**

- **Definition**
- The floor of x ($\lfloor x \rfloor$) is the greatest integer less than or equal to x

$x=2.6; \lfloor x \rfloor=2$

$x= -1.3; \lfloor x \rfloor=-2$

- The Ceiling of x(⌈x⌉) is the least integer greater or equal to x

  x=2.6; ⌈x⌉=3

  x= -1.3; ⌈x⌉=-1

Polynomials

- Given a nonnegative integer d, a polynomial in n of degree d is the a function p(n) of the form:

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

  - Where a_0, a_1, …, a_d are coefficients of the polynomial and $a_d \neq 0$

Exponentials

- For all real a>0, m and n, the following hold:
  - $a^0 = 1$
  - $a^1 = a$
  - $a^{-1} = 1/a$
  - $(a^m)^n = a^{mn}$
  - $(a^m)^n = (a^n)^m$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

Logarithms

- The following notations are adopted
  - Lgn = $\log_2 n$ *(binary logarithm)*
  - Lnn = $\log_e n$ *(natural logarithm)*
  - Lg$^k n$ = $(\log n)^k$ *(Exponentiation)*
  - Lg Lgn = $lg(lgn)$ *(composition)*

*

Factorials and Functional Iteration

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

- Functional Iteration
  - $f^{(i)}(n)$ – function $f(n)$ iteratively applied i times to an initial value of n

*

Fibonacci Numbers

- Defined as follows:
  - $F_0 = 0$
  - $F_1 = 1$
  - $F_i = F_{i-1} + F_{i-2}$ for i >=2
- Each Fibonacci number is the sum of the two previous ones:
  - 0,1,1,2,3,5,8,13,21,34,55,…

*