CSCI 323 Lecture 3 Note Monday, 02/07/2022

Announcements:

- The Lecture Recordings will be available on the following YouTube Playlists Link: https://youtube.com/playlist?list=PLZaTmV9UMKlgYpo2cAiMaEWxqyvbiXDFd

- Went over the $\log(n!)$ is $\Theta(n\log(n))$ example from previous class. See Lecture Note 2

- Arrange the following functions in increasing order: $(1.5)^n$, n^{100}, $(\log n)^3$, $\sqrt{n}\log n$, 10^n, $(n!)^2$, and $n^{99}+n^{98}$.
  logarithm: $(\log n)^3$, $\sqrt{n}\log n$. $(\log n)^3 < \sqrt{n}\log n$, since $\log n$ is smaller than n to any positive power, then, $\log n < n^{\frac{1}{6}}$, $(\log n)^3 < \left(n^{\frac{1}{6}}\right)^3 = n^{\frac{1}{2}} < \sqrt{n}\log n$.
  polynomial: n^{100}, $n^{99}+n^{98}$. $n^{100} > n^{99}+n^{98}$
  exponential: $(1.5)^n$, 10^n. $(1.5)^n < 10^n$
  factorial: $(n!)^2$.
  Thus, the order will be $(\log n)^3 < \sqrt{n}\log n < n^{99}+n^{98} < n^{100} < (1.5)^n < 10^n < (n!)^2$.

Reading Assignment

Algorithm Design: 2.1, 2.2*, 2.4

- [2.1] Let f and g be two functions that $\lim_{n\to\infty}\frac{f(n)}{g(n)}$ exists and is equal to some number $c > 0$. Then $f(n) = \Theta(g(n))$.
- [2.2] (a) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.
  (b) If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$.
- [2.3] If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$.
- [2.4] Suppose that f and g are two functions such that for some other function h, we have $f = O(h)$ and $g = O(h)$. Then $f + g = O(h)$.
- [2.5] Let k be a fixed constant, and let $f_1, f_2, \ldots, f_k$ and h be functions such that $f_i = O(h)$ for all i. Then $f_1 + f_2 + \ldots + f_k = O(h)$.
- [2.6] Suppose that f and g are two functions (taking nonnegative values) such that $g = O(f)$. Then $f + g = \Theta(f)$. In other words, f is an asymptotically tight bound for the combined function $f + g$.
- [2.7] Let f be a polynomial of degree d, in which the coefficient a_d is positive. Then $f = O(n^d)$.
- [2.8] For every $b > 1$ and every $x > 0$, we have $\log_b n = O(n^x)$.
- [2.9] For every $r > 1$ and every $d > 0$, we have $n^d = O(r^n)$.

Common Running Times

- $O(1)$ Constant time algorithm
  - Find minimum or maximum value in a sorted list.
  - Some Dynamic Problem, where queue comes one by one.
- $\log n$ logarithmic time algorithm
  - Binary Search – search in a sorted list
- $\sqrt{n}$
- n Linear time algorithm
  - Searching in an unsorted list
    - find a target value, minimum or maximum value
  - Merge two sorted lists
- $n\log n$ near-Linear time algorithm
  - Merage Sort

- $n^{1.5}$ near-Linear time algorithm
- n^2 Quadratic time algorithm
  - Insertion Sort – insert the array to correct position from left to right.
  - Bubble Sort – keep swapping the adjacent value from left to right. Each round will move the largest value to the right.
  - Selection Sort – keep selecting the minimum value and swapped it to the correct position.
  - Quick Sort – Select a pivot, swap values so the pivot is on correct spot, and break the problem by pivot. Repeat the process.
- $n^3, n^4, \ldots, n^c$ (c is some constant) Polynomial time algorithm
  - [n^3 runtime] We are given sets $S_1, S_2, \ldots, S_n$, each of which is a subset of $\{1, 2, \ldots, n\}$, and we would like to know whether some pair of these sets is disjoint, in other words, has no elements in common.
    - For pair of sets S_i and S_j
      Determine whether S_i and S_j have an element in common
      End for
    - For each set S_i
      For each other set S_j
      For each element p of S_i
      Determine whether p also belongs to S_j
      End for
      If no element of S_i belongs to S_j then
      Report that S_i and S_j are disjoint
      End if
      End for
      End for
  - [n^k runtime] Finding independent sets of size k in a graph. Independent set is a set of vertices in a graph with no two vertices that are adjacent.
    - For each subset S of k nodes
      Check whether S constitutes an independent set
      If S is an independent set then
      Stop and declare success
      Endif
      Endfor
      If no k-node independent set was found then
      Declare failure
      Endif
    - There are $\binom{n}{k}$ k-element subsets in an n-element set.
      $\binom{n}{k} = \frac{n!}{k!(n-k!)!} = \frac{n(n-1)(n-2)\ldots(n-k+1)}{k!} \le \frac{n^k}{k!} = \frac{1}{k!} \cdot n^k$, $\frac{1}{k!}$ is a constant, $\binom{n}{k} = O\left(n^k\right)$.
      To check whether the set is independent, you need to check every pair of vertices to see are they adjacent, that's $O(k^2)$. Thus, the runtimes is $O(k^2 n^k)$, k^2 is a constant, we can have $O\left(n^k\right)$.

- $n^{\log n}$ superPolynomial/subExponential time algorithm
- $2^{\sqrt{n}}$ superPolynomial/subExponential time algorithm
  - $n^{\log n}$ vs $2^{\sqrt{n}}$, which grows faster?
    - Let's make them the same base.

$$n^{\log n} = 2^{\log_2\left(n^{\log n}\right)} = 2^{\log n \cdot \log_2 n} \rightarrow \log n \cdot \log_2 n = c \cdot (\log n)^2 \le c'\sqrt{n}$$
$$\rightarrow 2^{\log n \cdot \log_2 n} \le 2^{c'\sqrt{n}} = 2^{c'} \cdot 2^{\sqrt{n}} \rightarrow n^{\log n} = O\left(2^{\sqrt{n}}\right)$$

- $(1.5)^n$ Exponential time algorithm
- 2^n Exponential time algorithm
    - [$n^2 2^n$ runtime] Given a graph and want to find an independent set of maximum size.
        - For each subset S of nodes

          Check whether S constitutes an independent set

          If S is a larger independent set than the largest seen so far then

          Record the size of S as the current maximum

          Endif

          Endfor
        - There are 2^n subsets in an n-element set and $O(n^2)$ to check whether set is independent, since the set can get as large as n nodes. Multiplying these two together, we get a running time of $O(n^2 2^n)$.
- $n!$
    - Traveling Salesman Problem: given a set of n cities, with distances between all pairs, what is the shortest tour that visits all cities? We assume that the salesman starts and ends at the first city, so the crux of the problem is the implicit search over all orders of the remaining $n-1$ cities, leading to a search space of size $(n-1)!$.
- We say an algorithm is efficient if it has a polynomial running time.

Exercise/In class quiz [2% Each]
- Prove that $1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + \cdots + 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ is $\Theta(n!)$.

- Solve for x and y in the following system of equations:

$$\begin{cases} 2^{\left(\frac{\log_2 y^3}{3}\right)} + 4^{(\log_2 \sqrt{x})} = 8 \\ 8^{(\log_4 (x^{2/3}))} \cdot 9^{(\log_3 \sqrt{y})} = 15 \end{cases}$$

What to expect or prepare for the next class:
- Stable Matching Problem
    - Try to read and understand the problem statement for stable marriage problem and try with an example if you can.

Reading Assignment

Algorithm Design: 2.1, 2.2, 2.4

Algorithm Design: 1.1 (Stable marriage problem statement)

Suggested Problems
- Algorithm Design – Chapter 2 – 1,2,3,4,5,8
- Discrete Mathematics and its Application
    - 3.1 - 9, 13, 19, 23, 27, 33, 41
    - 3.2 - 5, 9, 18, 21, 22, 24, 27, 33, 41, 42, 48