Week 11 - Friday

COMP 2230

Last time

- Algorithm efficiency

Questions?

Assignment 5

Back to Insertion Sort Analysis

Insertion sort in code

```
public static void sort( int[] array, int n) {
  for (int i = 1; i < n; ++i) {
     int next = array[i];
     int j = i - 1;
     while (j != 0 && array[j] > next) {
          array[j+1] = array[j];
          --j;
     }
     array[j] = next;
  }
}
```

Average case analysis of insertion sort

- What's the average case analysis of insertion sort?
- Much harder than the previous two!
- Let's look at it recursively
- Let E_k be the average number of comparisons needed to sort k elements
- E_k can be computed as the sum of the average number of comparisons needed to sort $k - 1$ elements plus the average number of comparisons (x) needed to insert the k^{th} element in the right place
  - $E_k = E_{k-1} + x$

Finding x

- We can employ the idea of **expected value** from probability
- There are k possible locations for the element to go
- We assume that any of these k locations is equally likely
- For each turn of the loop, there are 2 comparisons to do
- There could be 1, 2, 3, … up to k turns of the loop
- Thus, weighting each possible number of iterations evenly gives us

$$x = \sum_{j=1}^{k} \frac{1}{k} 2j = \frac{2}{k}\left(\frac{k(k+1)}{2}\right) = k + 1$$

Finishing the analysis

- Having found x, our recurrence relation is:
- $E_k = E_{k-1} + k + 1$
- Sorting one element takes no time, so $E_1 = 0$
- Solve this recurrence relation!
- Well, if you really banged away at it, you might find:
  - $E_n = \frac{1}{2}(n^2 + 3n - 4)$
- By the polynomial rules, this is $\Theta(n^2)$ and so the average case running time is the same as the worst case

Exponential and Logarithmic Functions

Exponential functions

- They grow **fast**
- Graph 2^x for $-3 \leq x \leq 3$
- When considering b^x, it's critically important whether $b > 1$ (in which case b^x grows very fast in the positive direction) or $0 < b < 1$ (in which case b^x goes to zero very fast)
- Graph b^x when $b > 1$
- Graph b^x when $0 < b < 1$
- What happens when $b = 1$?
- What happens when $b \leq 0$?

Logarithmic function

- The **logarithmic function with base** b, written $\log_b$ is the inverse of the exponential function
- Thus,
  - $b^y = x \leftrightarrow \log_b x = y$ for $b > 0$ and $b \neq 1$
- Log is a "de-exponentiating" function
- Log grows very **slowly**
- We're interested in $\log_b$ when $b > 1$, in which case $\log_b$ is an increasing function
  - If $x_1 < x_2$, then $\log_b x_1 < \log_b x_2$, for $b > 1$ and positive x_1 and x_2

Applying log

- How many binary digits are needed to represent a number n?
- We can write $n = 2^k + c_{k-1}2^{k-1} + \ldots c_2 2^2 + c_1 2 + c_0$ where c_i is either 0 or 1
- Thus, we need no more than $k + 1$ digits to represent n
- We know that $n < 2^{k+1}$
- Since $2^k \leq n < 2^{k+1}$, $k \leq \log_2 n < k + 1$
- The total number of digits we need $k + 1 \leq \log_2 n + 1$

Recurrence relations

- Consider the following recurrence relation
  - $a_1 = 0$
  - $a_k = a_{\lfloor \frac{k}{2} \rfloor} + 1$ for integers $k \geq 2$
- What do you think its explicit formula is?
- It turns out that $a_n = \lfloor \log n \rfloor$
- We can prove this with strong induction

Exponential and logarithmic orders

- For all real numbers b and r with $b > 1$ and $r > 0$
  - $\log_b x \leq x^r$, for sufficiently large x
  - $x^r \leq b^x$, for sufficiently large x
- These statements are equivalent to saying for all real numbers b and r with $b > 1$ and $r > 0$
  - $\log_b x$ is $O(x^r)$
  - x^r is $O(b^x)$

Important things

- We don't have time to show these things fully
- x^k is $O(x^k \log_b x)$
- $x^k \log_b x$ is $O(x^{k+1})$
- The most common case you will see of this is:
  - x is $O(x \log x)$
  - $x \log x$ is $O(x^2)$
  - In other words, $x \log x$ is between linear and quadratic
- $\log_b x$ is $\Theta(\log_c x)$ for all $b > 1$ and $c > 1$
  - In other words, logs are equivalent, no matter what base, in asymptotic notation
- $\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$ is $\Theta(\log_2 n)$

Review

Multiplication rule

- If an operation has k steps such that
  - Step 1 can be performed in n_1 ways
  - Step 2 can be performed in n_2 ways

    …
  - Step k can be performed in n_k ways
- Then, the entire operation can be performed in $\prod_{i=1}^{k} n_i = n_1 n_2 \ldots n_k$ ways
- This rule only applies when each step always takes the same number of ways (unlike the previous possibility tree example)

Addition and inclusion/exclusion rules

- If a finite set A equals the union of k distinct mutually disjoint subsets A_1, A_2, … A_k, then:

$$N(A) = N(A_1) + N(A_2) + \ldots + N(A_k)$$

- If A, B, C are any finite sets, then

$$N(A \cup B) = N(A) + N(B) - N(A \cap B)$$

- And

$$N(A \cup B \cup C) = N(A) + N(B) + N(C) - N(A \cap B) - N(A \cap C) - N(B \cap C) + N(A \cap B \cap C)$$

Handy dandy guide to counting

- This is a quick reminder of all the different ways you can count things:

	Order Matters	Order Doesn't Matter
Repetition Allowed	n^k	$\binom{k+n-1}{k}$
Repetition Not Allowed	$P(n,k)$	$\binom{n}{k}$

How many ways are there to…

- Order the letters in the word "replicant"
- Order any three letters in the word "replicant"?
- Order the letters in the word "neverness"
  - If all letters are considered distinct?
  - If identical letters cannot be told apart?
- Select an elite assassin team of 8 people from a squadron of 100?
- Select an elite assassin team of 8 people from a squadron of 100 and make one person the leader?

How many ways are there to…

- Name a child if you're willing use any sequence of characters from the Latin alphabet from 1 to 10 characters long?
- Decide the five courses you'd like to take in a semester, if the kinds are:
  - Computer Science courses
  - INST courses
  - Electives

Binomial theorem

- $a + b$ is called a **binomial**
- Using combinations (or Pascal's Triangle) gives an easy way to compute $(a + b)^n$

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

Probability axioms

- Let A and B be events in the sample space S
  - $0 \leq P(A) \leq 1$
  - $P(\emptyset) = 0$ and $P(S) = 1$
  - If $A \cap B = \emptyset$, then $P(A \cup B) = P(A) + P(B)$
  - It is clear then that $P(A^c) = 1 - P(A)$
  - More generally, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- All these axioms can be derived from set theory and the definition of probability

Expected value

- **Expected value** is one of the most important concepts in probability, especially if you want to gamble
- The expected value is simply the sum of all events, weighted by their probabilities
- If you have n outcomes with real number values $a_1, a_2, a_3, \ldots\ a_n$, each of which has probability $p_1, p_2, p_3, \ldots\ p_n$, then the expected value is:

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

Conditional probability

- Given that some event A has happened, the probability that some event B will happen is called conditional probability
- This probability is:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Graphs and Trees

Graphs

- A **graph** G is made up of two finite sets
  - **Vertices**: $V(G)$
  - **Edges**: $E(G)$
- Each edge is connected to either one or two vertices called its **endpoints**
- An edge with a single endpoint is called a **loop**
- Two edges with the same sets of endpoints are called **parallel**
- Edges are said to **connect** their endpoints
- Two vertices that share an edge are said to be **adjacent**
- A graph with no edges is called **empty**

Special graphs

- A **simple graph** does not have any loops or parallel edges
- Let n be a positive integer
- A **complete graph** on n vertices, written K_n, is a simple graph with n vertices such that every pair of vertices is connected by an edge
- A **bipartite graph** is a simple graph whose vertices can be partitioned into sets X and Y such that there are no edges from any vertex in X to another vertex in X and there are no edges from any vertex in Y to another vertex in Y
- A **complete bipartite graph on** (m, n) **vertices**, written $K_{m,n}$ is a simple graph with a set of m vertices and a disjoint set of n vertices such that:
  - There is an edge from each of the m vertices to each of the n vertices
  - There are no edges among the set of m vertices
  - There are no edges among the set of n vertices
- A **subgraph** is a graph whose vertices and edges are a subset of another graph

Degree

- The **degree of a vertex** is the number of edges that are incident on the vertex
- The **total degree of a graph** G is the sum of the degrees of all of its vertices
- What's the relationship between the degree of a graph and the number of edges it has?
- What's the degree of a complete graph with n vertices?
- Note that the number of vertices with odd degree must be even

Example

- What is the maximum number of edges a simple disconnected graph with n vertices can have?
- Prove it.

Definitions

- A **walk from v to w** is a finite alternating sequence of adjacent vertices and edges of G, starting at vertex v and ending at vertex w
  - A walk must begin and end at a vertex
- A **path from v to w** is a walk that does not contain a repeated edge
- A **simple path from v to w** is a path that does not contain a repeated vertex
- A **closed walk** is a walk that starts and ends at the same vertex
- A **circuit** is a closed walk that does not contain a repeated edge
- A **simple circuit** is a circuit that does not have a repeated vertex other than the first and last

Circuits

- If a graph is connected, non-empty, and every node in the graph has even degree, the graph has an **Euler circuit**
- Algorithm to find one:
  1. Pick an arbitrary starting vertex
  2. Move to an adjacent vertex and remove the edge you cross from the graph
     - Whenever you choose such a vertex, pick an edge that will not disconnected the graph
  3. If there are still uncrossed edges, go back to Step 2
- A **Hamiltonian circuit** must visit every vertex of a graph exactly once (except for the first and the last)
- If a graph G has a Hamiltonian circuit, then G has a subgraph H with the following properties:
  - H contains every vertex of G
  - H is connected
  - H has the same number of edges as vertices
  - Every vertex of H has degree 2

Matrix representations of graphs

- To represent a graph as an **adjacency matrix**, make an $n \times n$ matrix a, where n is the number of vertices
- Let the nonnegative integer at a_{ij} give the number of edges from vertex i to vertex j
- A simple graph will always have either 1 or 0 for every location
- We can find the number of walks of length k that connect two vertices in a graph by raising the adjacency matrix of the graph to the k^{th} power

Isomorphism

- We call two graphs isomorphic if we can relabel one to be the other
- Graph isomorphisms define equivalence classes
  - **Reflexive:** Any graph is clearly isomorphic to itself
  - **Symmetric:** If x is isomorphic to y, then y must be isomorphic to x
  - **Transitive:** If x is isomorphic to y and y is isomorphic to z, it must be the case that x is isomorphic to z

Isomorphism invariants

- A property is called an **isomorphism invariant** if its truth or falsehood does not change when considering a different (but isomorphic) graph
- These **cannot** be used to prove isomorphism, but they can be used to disprove isomorphism
- 10 common isomorphism invariants:
  1. Has n vertices
  2. Has m edges
  3. Has a vertex of degree k
  4. Has m vertices of degree k
  5. Has a circuit of length k
  6. Has a simple circuit of length k
  7. Has m simple circuits of length k
  8. Is connected
  9. Has an Euler circuit
  10. Has a Hamiltonian circuit

Trees

- A **tree** is a graph that is **circuit-free** and **connected**
- Any tree with more than one vertex has at least two vertcies of degree 1
- If a vertex in a tree has degree 1 it is called a **terminal vertex** (or **leaf**)
- All vertices of degree greater than 1 in a tree are called **internal vertices** (or **branch vertices**)
- For any positive integer n, a tree with n vertices must have $n - 1$ edges

Rooted trees

- In a **rooted tree**, one vertex is distinguished and called the **root**
- The **level** of a vertex is the number of edges along the unique path between it and the root
- The **height** of a rooted tree is the maximum level of any vertex of the tree
- The **children** of any vertex v in a rooted tree are all those nodes that are adjacent to v and one level further away from the root than v
- Two distinct vertices that are children of the same parent are called **siblings**
- If w is a child of v, then v is the parent of w
- If v is on the unique path from w to the root, then v is an ancestor of w and w is a descendant of v

Binary trees

- A **binary tree** is a rooted tree in which every parent has at most two children
- Each child is designated either the **left child** or the **right child**
- In a **full binary tree**, each parent has exactly two children
- Given a parent v in a binary tree, the **left subtree** of v is the binary tree whose root is the left child of v
- Ditto for **right subtree**

Spanning Trees

Spanning trees

- A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree
- Some properties:
  - Every connected graph has a spanning tree
    - Why?
  - Any two spanning trees for a graph have the same number of edges
    - Why?

Weighted graphs

- In computer science, we often talk about **weighted graphs** when tackling practical applications
- A weighted graph is a graph for which each edge has a real number **weight**
- The sum of the weights of all the edges is the **total weight** of the graph
- Notation: If e is an edge in graph G, then $w(e)$ is the weight of e and $w(G)$ is the total weight of G
- A **minimum spanning tree** (**MST**) is a spanning tree of lowest possible total weight

Finding a minimum spanning tree

- Kruskal's algorithm gives an easy to follow technique for finding an MST on a weighted, connected graph
- Informally, go through the edges, adding the smallest one, unless it forms a circuit
- **Algorithm:**
  - Input: Graph G with n vertices
  - Create a subgraph T with all the vertices of G (but no edges)
  - Let E be the set of all edges in G
  - Set $m = 0$
  - While $m < n - 1$
    - Find an edge e in E of least weight
    - Delete e from E
    - If adding e to T doesn't make a circuit
      - Add e to T
      - Set $m = m + 1$
  - Output: T

Prim's algorithm

- Prim's algorithm gives another way to find an MST
- Informally, start at a vertex and add the next closest node not already in the MST
- **Algorithm:**
  - Input: Graph G with n vertices
  - Let subgraph T contain a single vertex v from G
  - Let V be the set of all vertices in G except for v
  - For i from 1 to $n - 1$
    - Find an edge e in G such that:
      - e connects T to one of the vertices in V
      - e has the lowest weight of all such edges
    - Let w be the endpoint of e in V
    - Add e and w to T
    - Delete w from V
  - Output: T

Growth of Functions

Growth of functions

- Power functions:
  - $p_a(x) = x^a$ where $a, x \geq 0$
- Increasing and decreasing functions
- Let f and g be real-valued functions defined on the same set of nonnegative real numbers
  - f **is of order at least** g, written $f(x)$ is $\Omega(g(x))$, iff there is a positive $A \in \mathbb{R}$ and a nonnegative $a \in \mathbb{R}$ such that
    - $A|g(x)| \leq |f(x)|$ for all $x > a$
  - f **is of order at most** g, written $f(x)$ is $O(g(x))$, iff there is a positive $B \in \mathbb{R}$ and a nonnegative $b \in \mathbb{R}$ such that
    - $|f(x)| \leq B|g(x)|$ for all $x > b$
  - f **is of order** g, written $f(x)$ is $\Theta(g(x))$, iff there are positive $A, B \in \mathbb{R}$ and a nonnegative $k \in \mathbb{R}$ such that
    - $A|g(x)| \leq |f(x)| \leq B|g(x)|$ for all $x > k$

Polynomials

- Let $f(x)$ be a polynomial with degree n
  - $f(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} \ldots + a_1 x + a_0$
- By extension from the previous results, if a_n is a positive real, then
  - $f(x)$ is $O(x^s)$ for all integers $s \geq n$
  - $f(x)$ is $\Omega(x^r)$ for all integers $r \leq n$
  - $f(x)$ is $\Theta(x^n)$

Running time for algorithms

- First, assume that the number of operations performed by A on input size n is dependent only on n, not the values of the data
  - If $f(n)$ is $\Theta(g(n))$, we say that A **is** $\Theta(g(n))$ or that A **is of order** $g(n)$
- If the number of operations depends not only on n but also on the values of the data
  - Let $b(n)$ be the **minimum** number of operations where $b(n)$ is $\Theta(g(n))$, then we say that **in the best case**, A **is** $\Theta(g(n))$ or that A **has a best case order of** $g(n)$
  - Let $w(n)$ be the **maximum** number of operations where $w(n)$ is $\Theta(g(n))$, then we say that **in the worst case**, A **is** $\Theta(g(n))$ or that **A has a worst case order of** $g(n)$

Computing running time

- With a single `for` (or other) loop, we simply count the number of operations that must be performed
- When loops do not depend on each other, we can simply multiply their iterations (and asymptotic bounds)
- When loops depend on each other, we should analyze the loops like a series
- Repeated divisions by a constant k **often** lead to a $\log_k n$ term

Upcoming

Next time…

- Exam 3

Reminders

- Finish Assignment 5
- Study for Exam 3 on Monday