Logistics

- The final exam **time and location**
  - **Thursday May 19th from 9AM-12PM.**
  - **Location**
    - McCosh Hall 46: **Friday Precepts** P06, P07, P07A.
    - McCosh Hall 50: **Thursday Precepts** P01, P02, P02A, P03, P03A, P04, P05 and P99(flipped).

- **Exam Format**
  - Closed book, closed note.
  - You may bring one 8.5-by-11 sheet (both sides) with notes in your own handwriting to the exam.
  - No electronic devices (e.g., calculators, laptops, and cell phones).
Material covered

• The exam will stress material covered since the midterm, including the following components.
  – Lectures 13–23.
  – Exercises 12–22.
  – Programming assignments 6–8
    • Wordnet, seam-carving, burrows-wheeler
Topics covered

Depth-first search
Kruskal's algorithm
Key-indexed counting
Knuth-Morris-Pratt substring search
RE to NFA
Run-length coding

Topological sort
Bellman-Ford algorithm
MSD radix sort
Rabin-Karp substring search
Ternary search tries
LZW compression

Breadth-first search
Dijkstra's algorithm
LSD radix sort
Boyer-Moore substring search
R-way tries
Huffman coding

Prim's algorithm
Ford-Fulkerson algorithm
3-way radix quicksort
Reductions
Burrows-Wheeler
Algorithms

• focus on understanding fundamentals, not memorizing details (eg: code)
• Write down as many algorithms as you can recall
• For each algorithm
  – understand how it works on typical, worst case, best case input
  – How is it different from other algorithms for the same class of problems?
  – When is it effective?
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Application Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP, Boyer-Moore, Rabin-Karp</td>
<td>String search</td>
</tr>
<tr>
<td>Insertion, selection, mergesort, quicksort, 3-way quicksort</td>
<td>General Sorting</td>
</tr>
<tr>
<td>LSD, MSD, 3-way MSD</td>
<td>String/Radix Sorting</td>
</tr>
<tr>
<td>Linear search, binary search</td>
<td>General Search</td>
</tr>
<tr>
<td>Kosaraju-Sharir</td>
<td>Strong components in a directed graph</td>
</tr>
<tr>
<td>Topological Sort</td>
<td>Vertex ordering</td>
</tr>
<tr>
<td>DFS, BFS</td>
<td>Graph search, path detection, cycle detection</td>
</tr>
<tr>
<td>Dijkstra’s, Bellman-Ford</td>
<td>Single-source shortest path</td>
</tr>
<tr>
<td>Ford-Fulkerson</td>
<td>MaxFlow-MinCut, Matching Algorithms</td>
</tr>
<tr>
<td>Huffman, run-length encoding, LZW, Burrows-Wheeler</td>
<td>Data compression</td>
</tr>
<tr>
<td>Prims (eager, lazy), Kruskal’s</td>
<td>Minimum Spanning Tree (MST)</td>
</tr>
</tbody>
</table>
Data Structures

• Write down as many data structures as you can

• For each data structure
  – How is it implemented?
  – Alternate implementations?
  – What type of problems are good for this data structure?
  – What are the memory requirements?
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>Random access, contiguous memory, static</td>
</tr>
<tr>
<td>Resizable arrays</td>
<td>Random access, contiguous memory, dynamic with constant amortized cost per</td>
</tr>
<tr>
<td></td>
<td>insert/delete</td>
</tr>
<tr>
<td>Linked Lists, Doubly Linked Lists</td>
<td>Sequential access, flexible memory allocation/deallocation</td>
</tr>
<tr>
<td>Priority Queues</td>
<td>Binary heap implementation, delMax, insert in worst case O(log N) time</td>
</tr>
<tr>
<td>Binary Search Trees</td>
<td>At most two children per node, worst case linear access, ordered</td>
</tr>
<tr>
<td>Red-Black Trees</td>
<td>Balanced BST, guaranteed log N operations</td>
</tr>
<tr>
<td>Kd-Trees</td>
<td>Efficient organization of multi-dimensional data</td>
</tr>
<tr>
<td>Directed Graphs</td>
<td>Set of vertices, set of directed edges</td>
</tr>
<tr>
<td>Undirected Graphs</td>
<td>Set of vertices, set of undirected edges</td>
</tr>
<tr>
<td>Tries</td>
<td>Supports efficient prefix lookup, store strings with common prefixes efficiently</td>
</tr>
<tr>
<td>Union-Find</td>
<td>Support two operations, union and find, implementations of quick union, quick</td>
</tr>
<tr>
<td></td>
<td>find, weighted UF</td>
</tr>
<tr>
<td>Symbol tables</td>
<td>A key-value mapping, multiple implementations with red-black trees(ordered ST), arrays</td>
</tr>
<tr>
<td>Hash table</td>
<td>Constant time insert/search with uniform hashing, linear time worst case</td>
</tr>
<tr>
<td>Problem Cluster</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Identify the sort</td>
<td>(Sorting invariants)</td>
</tr>
<tr>
<td>Maxflow-mincut</td>
<td>(application of the algorithm)</td>
</tr>
<tr>
<td>Tilde Notation</td>
<td>(counting comparisons, exchanges)</td>
</tr>
<tr>
<td>Order of growth Analysis</td>
<td>(recurrences)</td>
</tr>
<tr>
<td>Design Problems</td>
<td>(data structure, algorithm, performance requirements)</td>
</tr>
<tr>
<td>Memory Calculation</td>
<td>(size of data)</td>
</tr>
<tr>
<td>Graph Algorithms Trace</td>
<td>(DFS, BFS, Kruskals, Prim’s, Dijkstra’s, SCA)</td>
</tr>
<tr>
<td>String Sort/Search</td>
<td>(LSD, MSD, 3-way quicksort, Key-indexed counting, KMP, DFA construction, Boyer-Moore, Rabin-Karp)</td>
</tr>
<tr>
<td>RegEx/NFA</td>
<td>(construction, tracing)</td>
</tr>
<tr>
<td>Tries and TST</td>
<td>(construction, operations, memory)</td>
</tr>
<tr>
<td>Compression</td>
<td>(Huffman, LZW compress/expand, Burrows-Wheeler)</td>
</tr>
<tr>
<td>Reductions</td>
<td>(3sum → 3sum variant, longest path → longest cycle)</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>(Matching, True/False, Possible/impossible)</td>
</tr>
</tbody>
</table>
Assume $f_1(N)$ is of $O(N)$

$$f_2(N,R) = f_1(0) + f_1(1) + \ldots + f_1(R-1) = 0 + 1 + \ldots + R-1 \sim O(R^2)$$

$$f_5(N,R) = N \left( 1 + 2 + 4 + \ldots + R \right) \sim N \left( 2R \right) \sim O(NR)$$
Assume f1 is of O(N)

\[
f_4(N) = 2f_4(N/2) + 3f_1(N)
\]
\[
= 2f_4(N/2) + 3N \quad \text{(like mergesort)}
\]
\[
\sim O(N \log N)
\]

\[
f_3(N) = N \cdot f_3(N-1)
\]
\[
\sim O(N!)
\]
Counting Memory

```java
public class Bag<Item> implements Iterable<Item> {
    private Node<Item> first; // beginning of bag
    private int N; // number of elements in bag

    // helper linked list class
    private static class Node<Item> {
        private Item item;
        private Node<Item> next;
    }
}
```

Node = $16 + 8 + 8 = 32$ (no inner class overhead)

Bag of N Nodes = $32N + 16 + 8 + 4 + 4$ (padding)

~ $32N$
## Graphs

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s–t path</td>
<td>Is there a path between s and t?</td>
</tr>
<tr>
<td>shortest s–t path</td>
<td>What is the shortest path between s and t?</td>
</tr>
<tr>
<td>cycle</td>
<td>Is there a cycle in the graph?</td>
</tr>
<tr>
<td>Euler cycle</td>
<td>Is there a cycle that uses each edge exactly once?</td>
</tr>
<tr>
<td>Hamilton cycle</td>
<td>Is there a cycle that uses each vertex exactly once?</td>
</tr>
<tr>
<td>connectivity</td>
<td>Is there a path between every pair of vertices?</td>
</tr>
<tr>
<td>biconnectivity</td>
<td>Is there a vertex whose removal disconnects the graph?</td>
</tr>
<tr>
<td>planarity</td>
<td>Can the graph be drawn in the plane with no crossing edges?</td>
</tr>
<tr>
<td>graph isomorphism</td>
<td>Are two graphs isomorphic?</td>
</tr>
</tbody>
</table>
Graph Order Traversals

- **Preorder**: order in which dfs() is called.
  
  \[0 \ 1 \ 2 \ 3\]

- **Postorder**: order in which dfs() returns.
  
  \[2 \ 1 \ 3 \ 0\]

- **Reverse postorder**: reverse order in which dfs() returns.
  
  \[0 \ 3 \ 1 \ 2\]
DFS vs BFS

• DFS enables
  – Reachability.
  – Path finding.
  – Topological sort.
  – Directed cycle detection.

• BFS enables
  – Single source shortest path
Mystery Code

```java
for (Edge e : G.adj(v))
{
    int w = e.to();
    if (dist[w] > dist[v] + e.weight())
    {
        dist[w] = dist[v] + e.weight();
        pred[w] = e;
        pq.insert(dist[w], w);
    }
}
```

This partial code belongs to one of the graph algorithms. Which one(s)?

This is piece of a shortest path algorithm
Finding SCC’s

v and w are connected if there is a path between v and w.

v and w are strongly connected if there is both a directed path from v to w and a directed path from w to v.

Finding the SCC
1. Find the reverse Post order of $G^R$
   4 2 5 1 0 6 3
2. Traverse the original graph G in the order found in 1

The outcome is each component is its own strong component. {0}, {1}, {2}, {3}, {4}, {5}, {6}
Hashing

• When implementing a ST with hashing, what operations are not allowed in the ST?

  Ordered operations are not allowed (like max, min, range)

• What is a collision in a hashtable?

  Given two keys K1 and K2, \( H(K1) = H(K2) \)

• How can we minimize collisions?

  Use a uniform hashing function

• With uniform hashing assumption, can we assure that hashtable operations always perform at \( O(1) \)

  No. There can be bad input, where all keys anyway map to the same place.
Hashing

• True/False
  – A linear probing hash table always finds a place
    
    Yes, if the number of keys $M$ is less than or equal to the table size $N$
  – A separate chaining hash table always finds a place
    
    Yes
  – The load factor of a hash table is always $\leq 1$
    
    The load factor is $M/N$ and if we are using a linear probing table this is true. Otherwise
  – A linear probing hash table must be rehashed if load factor is over 0.7
    
    Yes, otherwise the performance can degrade
  – A rehashed entry will be at the same location as the original
    
    Not necessarily as the table size will change and hence we will have a different % value for each hash code
Which is the min weight edge crossing the cut \{ 2, 3, 5, 6 \}? 

In this case draw the edges that crosses the cut \{2,3,5,6\} (as shown in red) 
And find the one with the lowest edge cost (0, 2, 0.26)
Kruskal’s algorithm

1. build a PQ of edges.
2. Delete min and add the edge and vertices to MST as long as no cycles are created.
3. See the edges that are selected.

We are done now, since we have added V-1 edges to MST.

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>build pq</td>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>delete-min</td>
<td>E</td>
<td>$\log V$</td>
</tr>
<tr>
<td>union</td>
<td>V</td>
<td>$\log V$</td>
</tr>
<tr>
<td>connected</td>
<td>E</td>
<td>$\log V$</td>
</tr>
</tbody>
</table>
Prim’s (Lazy)

This is the code for Prim's Lazy algorithm:

```java
pq = new MinPQ<Edge>();
mst = new Queue<Edge>();
marked = new boolean[G.V()];
visit(G, 0);

while (!pq.isEmpty() && mst.size() < G.V() - 1) {
    Edge e = pq.delMin();
    int v = e.either(), w = e.other(v);
    if (marked[v] && marked[w]) continue;
    mst.enqueue(e);
    if (!marked[v]) visit(G, v);
    if (!marked[w]) visit(G, w);
}

private void visit(WeightedGraph G, int v) {
    marked[v] = true;
    for (Edge e : G.adj(v))
        if (!marked[e.other(v)])
            pq.insert(e);
}
```

1. It maintains a PQ of edges (space E)
2. It starts to build MST from node 0
3. Find the min edge weight that connects 0 to other nodes. Add that to MST
4. Continue this until we have found V-1 edges

Notes:
1. The complexity of this algorithm is also $O(E \log E)$
2. It uses extra space proportional to E

6. Another version of this is Prim’s eager algorithm. In that case, you maintain a PQ of vertices instead of edges. $O(E \log V)$ order of growth
If edges X, Y and Z are in the MST
1. Find the other edges that are in MST

   Choose the min edges such that they do not form a cycle. They are 10, 20, 30, 40, 50 and 100

2. find upper bounds for edge costs of X, Y and Z?

   1. X <= 110, otherwise we can replace X with edge A-F
   2. Y <= 60, otherwise we can replace Y with edge D-I
   3. Z <= 80, otherwise we can replace Z with edge B-H
Shortest Paths

Generic algorithm (to compute a SPT from s)

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat until optimality conditions are satisfied:
   - Relax any edge.

Specific Algorithms

- Dijkstra's algorithm (nonnegative weights).
- Topological sort algorithm (no directed cycles).
- Bellman–Ford algorithm (no negative cycles).
Max Flow

Fundamental questions.

- How to compute a mincut? Easy.
- How to find an augmenting path? BFS works well.
- If FF terminates, does it always compute a maxflow? Yes.
- Does FF always terminate? If so, after how many augmentations?

yes, provided edge capacities are integers (or augmenting paths are chosen carefully) requires clever analysis
1. What is the possible max flow of the network?
2. Mark an augmenting path and increase flow
3. Find all augmenting paths and increase flow
4. What is the actual max flow?
5. What is a min-cut?
6. Min-cut can only be calculated when a certain condition is true. What is it? How do we find out?

Find a max flow and min cut for the flow diagram as a homework problem
1. What is the regular expression?
   
   \((A^* \mid (A \cdot B \cdot A)^*)\)

2. Suppose that you simulate the following sequence of characters on the NFA above: **A A A A A A A**. In which one or more states could the NFA be?

   - **Empty** = \{1, 2, 3, 11, 12, 4, 5, 10\}
   - **A** = \{1, 2, 3, 6, 7, 8, 10, 11, 12\}
   - **AA** = \{1, 2, 3, 7, 8, 10, 11, 12\}

3. Suppose that you want to construct an NFA for the regular expression \((A^* \mid (A \cdot B \cdot A)^+)^*\) where the operator + means one or more copies. What minimal change(s) would you make to the NFA above?

   - **Remove the epsilon edge from state 4 to state 10**
Compression Algorithms
Run-length encoding

• Describe the algorithm
  If we have long runs of say 0’s and 1’s they can be encoded by an integer. For example, if we have 200 0’s followed by 300 1’s, we can represent them by two integers 200 100. If the integers take on 16 bits each, we have encoded 500 original bits with 32 bits.

• Under what circumstances would you use this algorithm?
  For example, an image with many white spaces like a fax machine output.

• If 8-bit words are used to store counts, what is the length of the maximum run that can be stored?
  The max number we can represent with 8-bits is $2^8 - 1 = 255$.

• What can we do if the length of the run cannot be accommodated by n-bit word?
  Encode with 0’s. For example if the compressed file is: 255 0 54 0 etc.. What we have is a file that has 255 0’s followed by 54 1’s. In the decompression step we ignore the runs of length 0.

• What is the best case input for run-length encoding (8-bit code words)?
  When we can encode 255 bits with just 8 bits. A compression ratio of 8:255.
Compression Algorithms
Huffman encoding

- Describe the algorithm
  
  Given a file, find the frequencies of each character. Build a code (pre-fix free) such that characters with high frequencies have lower length codes.

- What is the preprocessing step of the algorithm?
  
  Using a PQ of counts, pick the smallest two counts first, combine them. Add that to the PQ. Continue this until there is one tree. The codes can be obtained from this Trie.

- What data structure(s) is/are used in the preprocessing step?
  
  Need a PQ and some way to maintain a Trie structure.

- If you compress a file with all characters the same (eg: 10000 A’s) what is the compression ratio?
  
  All characters can be represented with 1 or 0. So we can compress one character (8-bits) to 1 bit. So we have 1:8 ratio.

- Describe a situation where no compression is obtained
  
  Suppose there is a file with all 256 characters with the same frequency. Then we will create a nice looking huffman tree whose height is log (256) = 8. So each character would require 8 bits to compress. No advantage. In fact the file size can increase since we need to transmit the code table.
Compression Algorithms
LZW encoding

- Describe the algorithm
  The algorithm stores codes for strings of words it has seen before. As algorithm progresses, it finds longer and longer strings and encode them with some value

- What data structure is used to store code words?
  Code words are stored in a Trie

- Is it possible to run out of code words to store new words? How?
  Yes, if we use 16-bit code words, then we can have a max of $2^{16} - \{\text{alphabet size}\}$ of codes. But that is not a big problem, as encoding longer strings with code words, probably won’t be useful any way

- Should we send the code words with the compressed file?
  No unlike the huffman code (where we have to send the code table), in LZW, we can reconstruct the code table as we decompress

- How can we decompress a file?
  Most likely the first few numbers are alphabet codes like 41, 42. We can decompress them with A B etc and then add AB to the table as next code word. Continue this process.

- What is the tricky case and how do we overcome that?
  Tricky case is when a code appears when there is no matching string in the table. This is usually caused when a code is immediately used by the the compression algorithm. The way to deal with this is to know that each new code is previous code + some other character. So if the last encoded message was: abb, then the missing code should have been : abb + a
LZW

- 97: a, ab = 128
- 98: b, ba = 129
- 128: ab, abb = 130
- 129: ba, we don’t know what to add
- 131: ba + b, bab = 131 (previous code: ba + first character b)
- 132: bab + b, babb = 132 (sane as above)
- 130: abb

Decode the message a=97, b=98, and start next token from 128
KMP Algorithm

• Briefly describe the algorithm

   This is one of the very clever algorithms that was ever invented. The key idea is to remember the strings we have seen before, so we do not need to backup the text.

• What is the order of growth of building the DFA? Typical algorithm? Best algorithm?
   If the string is size M, then it would cost M R to build the DFA where R is the alphabet size. As an order of growth we would need O(M). The typical case is M^2, but in the best case this can be done in linear time.

• What is the order of growth of the algorithm for searching for a pattern of length m in a text of length n?
   M + N

• Can KMP be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

   Sure, reset the DFA and keep looking for the next pattern. The order of growth is M + N

• Is KMP the algorithm of choice for any substring search application?

   Not necessarily. But is the only one with guaranteed linear performance.
Exercise

• Build the DFA for: ABAAABB
Exercise

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>7</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>s</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

Complete the table and find the search string
Boyer Moore (BM)

• Briefly describe the algorithm

  Start comparing characters from right to left. You can do bigger jumps when they don’t match.

• What is the pre-processing step of Boyer-Moore?

  Create a table of jumps we need to make when it fails. Align the text with the last occurrence of the failed character in the pattern

• What is the order of growth of the algorithm for searching for a pattern of length m in a text of length n? Best case? Worst case?

  Best case, it can be sub-linear making jumps in the order of m. So we can do this in n/m time.
  The worst case still can be m n (consider text=aaaaaaabaa and pattern = baa

• Can BW be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

  Yes, Just keep looking as the substring after the matched pattern is a new string. Order of growth would be the same.
Rabin-Karp (RK)

- Briefly describe the algorithm

  Given a text \( T \) and pattern \( M \), find all substrings of \( T \) of the form \( T(i...i+M-1) \) and compute hash codes. Compute the hash code of \( M \) and see if there is any substring with the same hashcode and then compare the characters

- What is the pre-processing step of RK algorithm?

  Finding the hashes and this can be done efficiently

- What is the order of growth of the algorithm for searching for a pattern of length \( m \) in a text of length \( n \)? Best case? Worst case?

  To compute \( N \) hashes it cost \( N \) time. So it can be done in \( M + N \)

- Can RK be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

  Yes, this is really efficient since all occurrences of the string will have same hash codes
Challenge Questions

• Answer each question with possible, impossible, unknown

  – There exist an algorithm where duplicity of elements in a set can be determined in sub-linear time

    Not possible w/o examining at least all of the elements

  – The convex hull problem (i.e. finding a set of points that encloses a given set of n points) can be solved in linearithmic time

    Possible, since we know a way to do this

  – Inserting n comparable keys into a BST in time proportional to n

    Not possible. If it is, then we can sort in linear time and we know that is not true
Match Algorithms

A. Trie
B. Hashing
C. 3-way radix quicksort
D. Binary search tree
E. Kd tree
F. Depth-first search
G. Breadth-first search
H. Dijkstra’s algorithm
I. Topological sort
J. Bellman-Ford
K. Enumerate permutations

A__ T9 texting in a cell phone
D__ 1D range search
E__ 2D range search
B__ Document similarity
K__ Traveling salesperson problem
G__ Web crawler
H__ Google maps
I__ PERT/CPM (Program Evaluation and Review Technique / Critical Path Method).
Classifying Algorithms

(a) Which of the following can be performed in *linear time* in the *worst case*?
   Write *P* (possible), *I* (impossible), or *U* (unknown).

- **P** ___ Printing the keys in a binary search tree in ascending order.
- **P** ___ Finding a minimum spanning tree in a weighted graph.
- **P** ___ Finding all vertices reachable from a given source vertex in a graph.
- **P** ___ Checking whether a digraph has a directed cycle.
- **P** ___ Building the Knuth-Morris-Pratt DFA for a given string.
- **P** ___ Sorting an array of strings, accessing the data solely via calls to `charAt()`.
- **I** ___ Sorting an array of strings, accessing the data solely via calls to `compareTo()`.
- **I** ___ Finding the closest pair of points among a set of points in the plane, accessing the data solely via calls to `distanceTo()`.
1. What data structures are used to store wordnet?
   - DiGraphs, HashMaps

2. What data structures are used to store SCA?
   - DiGraphs,

3. Is there a reason that we used BFS not DFS?
   - DFS can fail to find shortest path as shown in precepts

4. What is the order of the best algorithm that can find the length of the common ancestor?
   - $V + E$

5. What is the order of the best algorithm that can find the common ancestor?
   - $V + E$

6. What is a rooted DAG and how do we determine that? Order or growth of your algorithm?
   - A Digraph with one node, whose outdegree is 0. Graph must also be acyclic

7. If the wordnet is NOT a rooted DAG, will answers to 3 and 4 will hold?
   - No since there may not be a SCA (nodes in two disjoint subsets)

8. Given a list of $n$ nouns, What is the order of growth of the outcast algorithm?
   - $n^2 (V + E)$
Seam-Carving

• What is the purpose of the seamcarving assignment?

To resize an image by removing seams that are low energy

• How does it relate to shortest path?

    The problem can be reduced to finding the shortest path in a DAG. An image with \( N \times N \) dimension can be represented as a graph with at most \( 3 \times N^2 \) edges. So we have an algorithm of order: \( 3 \times N^2 \)

• How to find Vertical and Horizontal seams?

    Find a column of pixels with the lowest energy total. Do the same for horizontal seams

• Why do a defensive copy of Picture?

    To return to client a new copy of the changed image

• What is the order of growth for the two methods, removeHorizontalSeam and removeVerticalSeam?

    \( W \times H \)
Burrows-Wheeler

- What libraries were used to read and write input/output to the program?
  - `binaryStdIn` and `BinaryStdOut`
- What method in the output library that was required to print the output correctly?
  - `HexDump`, `Close`, `Flush`
- What data structures were used to implement BW, CSA, MoveToFront?
  - **BW** – Circular Suffix Array
  - **CSA** uses Array and **MoveToFront** uses an Array
- What is the order of growth to form circular suffixes of a given string?
  - Can be done in linear time. One trick is to form a new string $S + S$ and then use start and end to keep track of the suffixes
- What is wrong with using `LSD.sort()` in the program?
  - Can be quadratic in performance
- Could we have used quicksort to sort suffixes? How? If so how can you avoid quadratic performance?
  - We could have used quicksort. However, the `compareTo` should be designed carefully to avoid suffixes comparing to itself leading to quadratic performance
Burrows-Wheeler ctd...

• What are the 3-steps to burrows-wheeler transform?

  BW, move to front, huffman

• How would you sort strings w/o forming them explicitly?

  You can use suffixes using the references to first and last characters of the suffix

• Is it necessary to do move to front? If not, why did we do it?

  Not necessary. But it helps with the huffman by creating only a few characters with many duplicates

• How did we do the inverse transform?

  We did the next array to figure out how to get string from the suffix sort
Good Luck with All Finals