/** partial linked list class for answering q1 */
public class LinkedList
{
   /** the links connecting up the list */
   protected class Link
   {
      public int data; // the thing in the link
      public Link next; // the next link of the list
      // initialize a link
      public Link (int forData, Link forNext)
      { data = forData; next = forNext; }
   } // end Link

   /** points to a dummy link before the first link of the list */
   protected Link contents = new Link(0,null);

   /** does quiz */
   public void deleteNegativeNumbers()
   {
      // traverse deleting negatives
      for(Link before = contents; null!=before.next;)
      // check if we should delete
      if(before.next.data<0) // delete it
      {
         before.next = before.next.next; // remove link
      }
      else before = before.next
   } // end deleteNegativeNumbers

} // end LinkedList
Efficiency of an algorithm (focus on speed):

Pick a size measure for input.
Easy for lists - number of items

Harder as the input data gets more complex

Avoid comparing apples to oranges

input size = n

Efficiency = f(n)
But there are lots of inputs with the same size.

3 ways to analyze:

Best case - pick the fastest run over all the inputs.
Silly
Lower limit

Worst case - the slowest run of all the inputs of size n.

Average case - average all the possible inputs of size n
Most useful

What are we measuring in speed.
We don't want a different answer for every different computer.

Pick something to count that won't change with computer speed.
Report 1: due 10/7

1. Why is it important to know the worst case speed for accessing, modifying, and deleting classes from the open class list in Banner?

Pasted from <http://matcmp.ncc.edu/~sherd/classdoc/cmp251/Plan.htm>

When you go into banner and see what classes are available - this is the open class list.

O-notation
Asymptotic behavior

Ignore constant start up and termination work.
Ignore constant multiples since they are machine speed dependent

\[ O(n) = O(2n) = O(2n+33402098402) = O(342.3402n+3942) < O(n^2) \]
\[
\log_a(x) = \frac{\log_b(x)}{\log_b(a)}
\]

Don't need to worry about log bases inside O

O(\log(n)) is the next best thing to constant time

O(n \log(n)) is almost as good as proportional.

Is O(n \log(n)^3) < O(n^2)
\[
\lim_{n \to \infty} \frac{\ln(n)}{n} = \lim_{n \to \infty} \frac{\ln(n)}{n} = \frac{1}{\infty} = 0
\]

\[
\lim_{n \to \infty} \frac{n \ln(n)^2}{n^2} = \frac{\ln(n)^2}{\infty} = 0
\]

\[
\lim_{n \to \infty} \frac{2 \ln(n)}{n} = \frac{2 \frac{1}{n}}{\infty} = 0
\]

\[
500000 + 2 \ln(n) \quad \text{vs} \quad \frac{\ln(n)}{30000}
\]

\[
\ln(n)^n \quad \text{vs} \quad \ln(n)^n
\]

\[
\ln(n)^n = e^{\ln(n)^n} = e^{\frac{\ln(\ln(n))}{\ln(n)}}
\]

\[
e^{\ln(\ln(n))} = \ln(n)
\]

\[
e^{\ln(\ln(n))} = \ln(n)
\]

\[
e^{\ln(\ln(n))} = \ln(n)
\]

\[
\lim_{n \to \infty} \ln(n) - \ln(n)^2
\]

\[
e \lim_{n \to \infty} \ln(n) - \ln(n)^2
\]

Which is bigger as \(n \to \infty\):

\[
\ln(n) \quad \text{or} \quad \ln(n)^2
\]

What is \(\lim_{n \to \infty} \frac{\ln(n)^n}{\ln(n)^2}\)?

\[
\lim_{n \to \infty} \frac{\ln(n)^n}{\ln(n)^2}
\]

\[
\frac{\ln(n)^n}{\ln(n)^2}
\]
\[
\lim_{n \to \infty} \frac{2 \ln(n)/n}{n \ln(n)} + \frac{n}{\ln(n)} \leq \frac{2}{2 \ln(n)} \\
\lim_{n \to \infty} \ln(n) + \frac{1}{\ln(n)} + \frac{1}{\ln(n)} - \frac{n}{\ln(n)^2} = \infty \\
\lim_{n \to \infty} \ln(n) \ln(n) - \ln(n)^2 = \infty \\
\lim_{n \to \infty} \frac{\ln(n)^n}{\ln(n)} = \infty \\
O(\ln(n)^n) > O(n \ln(n)) \\
\text{which is larger} \\
O(2^{\log_2 n}) \text{ or } O(3^{\log_3 n}) \\
O(n) \text{ vs } O(2^{\log_2 n}) \\
\text{Searching an unsorted linked list of size } n > 0 \\
\text{Best case } \frac{1}{n} \\
\text{Worst case } \frac{1}{n} \\
\text{Average case} \\
\text{either it is in the list}
Average = \frac{n (1 + \frac{p}{n} - 1 - p)}{2}

Average = \frac{n}{2} (2 + \frac{p}{n} - p)

= n + \frac{p}{2} - \frac{np}{2}

= n + (1-n) \frac{p}{2}
Exam on 10/14  
Project 2 due 10/21

\[
\lim_{n \to \infty} \frac{2(\log n)^3 + 3n \log \log n + 100}{n} \\
= \lim_{n \to \infty} \frac{2(\log n)^3}{n^2} + \lim_{n \to \infty} \frac{3n \log \log n}{n^2} + \lim_{n \to \infty} \frac{100}{n} \\
= \frac{2}{\infty} \cdot \frac{3}{\infty} \cdot \frac{100}{\infty} \\
= 0 \\
\]

\[
12 \lim_{n \to \infty} \frac{h}{n} = 0 \\
\text{hence } c < O(n) \\
\]

\[
\text{Avg} = h + (-99) \cdot \frac{3}{2} \\
= (1-p)n + p \left( \frac{n-1}{2} \right) \\
\]

70% of a list of size 100

\[
\rho = 0.7 \\
\frac{100 + (-99) \cdot 0.7}{2} \\
= 65.35 \\
\]

On average we do 65.35 comparisons

The first 5 of the 100 occur 20% of the time.
The remaining 95 occur 50% not in list

\[
0.2 \cdot \frac{1+5}{2} + 0.5 \cdot \frac{6+100}{2} + 0.3 \cdot 100 = 57.1 \\
\text{8.25 comparisons faster} \\
\]
Two formulas for sorted lists:

\[(1-p)(n/2+n/(n+1)) + p(n+1)/2\]
\[(1-p)((n+2)/2-1/(n+1)) + p(n+1)/2\]

From sorting you should be

\[(1-p)n + p(n+1)/2 - (1-p)(n/2+n/(n+1)) + p(n+1)/2)\] faster

\[(1-p)(n-(n/2+n/(n+1))) = (1-p)(n/2-n/(n+1)) = (1-p)(n/2-1+1/(n+1))\]

Figure 1 Visual Representation of Expected Number of Searches

\[E = 1 + (1-p)E\]
\[-(1-p)E - (1-p)E\]
\[pE = 1\]
\[E = 1/p\]

List of size n
\[1/p - (1-p)^n/p\]
Review

1. 
   a. 
   \[ 600 = n \cdot 0.5 = p \]
   \[ 600 + (1 - 600) \cdot 0.5/2 = 450.25 \]
   \[ 0.5 \cdot 600 + 0.5 \cdot (601/2) = 450.25 \]
   b. 
   \[ 0.5 \cdot ((600+2)/2 - 1/(600+1)) + 0.5 \cdot (601/2) = 300.7492 \]
   c. 
   \[ 450.25 - 300.7492 = 149.5008 \]
   d. 
   \[ 0.2 \cdot (1+30)/2 + 0.3 \cdot (31+600)/2 + 0.5 \cdot 600 = 397.75 \]
   e. 
   \[ 450.25 - 397.75 = 52.5 \]

2. 
\[ \frac{n^2 \ln(n) - 150n^2}{n^2 \ln(n) - 5n^2 + n^3} \]
\[ \lim_{n \to \infty} \frac{7 \ln(n)}{2n \ln(n)^2 + 3n \ln(n) + 7n} \]
\[ \lim_{n \to \infty} \frac{42 \ln(n)}{2n^2 \ln(n)^2 + 7n^2 \ln(n) + 42} \]
\[ \frac{2n^2}{2n^2 \ln(n)^3 + 3n^2 \ln(n)^2 + 7n \ln(n)} = 0 \]

\[ \lim_{n \to \infty} \frac{n^2}{n^2 \ln(n)^3 + 3n^2 \ln(n)^2 + 7n \ln(n)} = 0 \]

1. \( \frac{1}{p} - (1-p)^{n/p} \)
2. \( \frac{1}{0.03} \cdot (1 - 0.03)^{1000}/0.03 = 33.3333 \)
3. \( \frac{1}{0.03} \cdot (1 - 0.03)^{4}/0.03 = 3.8236 \)
4. Alpha (beta (gamma (delta ())))
   () = gurgle
   Delta () = gurgle : delta
   Gamma(delta()) = gurgle : delta : gamma
   Beta (gamma (delta ()))) = gurgle: delta : gamma : beta
   Gurgle: delta : gamma : beta : alpha
Never use linked lists again.
Actually inserting in the middle is still better.
But stretch arrays have no real limit in size.
What makes a stretch array stretch is when it runs out of room
It doubles in size.
Keep a class variable called size = number of elements in the array.
When it equals array.length we allocate an array twice as big
Copy the elements to the new array
And replace the old one with the new one.
In fact we don't start with an array of size 1 but to make the math easier
We'll assume that.

Assume size is 2^n

We insert 1.

How many copies occurred historically

2^(n+1) - 1
Which is approximately twice the number of elements in the list

Between 2 and 3 copies per element.
1 copy to put element in the array (2^n)
+2^(n+1) - 1 extras
= 3*2^n - 1 worst case
Best case when list is of size 2^n - 1

Still need to shift for inserting in the middle.

Hash tables
Chained hash tables

Make search the list look like array indexing.
Finding a[x] is constant amount of work.

Hashing translates the item we are searching for into an
integer called the hash code.

We use the hash code to index an array.

Hashing properties.
• Hash(x) translates x into an array index.
• Hash(x) should always translate x into the same index
  (for a specified array size).
• If \( x \neq y \) then \( \text{Hash}(x) \neq \text{Hash}(y) \) most of the time.
• \( \text{Hash}(x) \) should not take long to compute.
• \( x \) is what is searched for. Different hash tables for different searches.

Hashing started for variables in compilers.

My favorite for string hashing is add all the character codes \% size.

Still a problem with anagrams. (Nothing is perfect).

Can't use a straight array. Collisions are when two items hash to the same index.
Each array element has a lot of items hash to it. Leads to collisions.

Chained hash don't have to store hash codes.

Best case $O(1)$

Worst case $O(n)$

Average case for a perfect hash

All hash codes are equally likely independent. Depends only on $n$. 


\[ S - \text{size of table} \]

\[ \sum_{k=0}^{n} p(k)(p(k+1)+(1-p)k) \]

p(k) probability that a list is of size k.

\[ \frac{1}{S} = \frac{\text{Prob that item licenses}}{\text{Size of list}} \]

\[ (1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k \]

\[ (1+x)^2 = 1^2 + 2x + x^2 \]

\[ (1+x)^3 = 1 + 3x + 3x^2 + x^3 \]

\[ \binom{n}{k} \binom{2}{0} = 1 \]

\[ \frac{2^2}{2!} = 2 \]

\[ \binom{2}{1} = 2 \]

\[ \binom{2}{2} = 1 \]

is the number of ways to select an size k.
\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} \]

\[ \sum_{k=0}^{n} \binom{n}{k} \frac{(s-1)^{n-k}}{s^n} \left( p \left( \frac{k+1}{2} \right) + (1-p)k \right) \]

\[ \sum_{k=0}^{n} \binom{n}{k} \frac{(s-1)^{n-k}}{s^n} \left( k + (1-k)p \right) \]

\[ \sum_{k=0}^{n} \binom{n}{k} \frac{(s-1)^{n-k}}{s^n} k + \left( 1 - \sum_{k=0}^{n} \binom{n}{k} \frac{(s-1)^{n-k}}{s^n} \right) \frac{p}{2} \]

If we simplify:

\[ \frac{1}{s} = \sum_{k=0}^{n} \binom{n}{k} \frac{(s-1)^{n-k}}{s^n} k \]

\[ \left( \frac{s-1}{s} \right)^{n-1} \sum_{k=0}^{n} \binom{n}{k} \frac{k}{(s-1)^k} \]

\[ \sum_{i=0}^{n} \frac{\binom{n}{i}}{(s-1)^i} \] assuming positive

\[ \left( \frac{s-1}{s} \right)^{n-1} \frac{s^n}{(s-1)^n} \]

\[ \frac{s^n-1}{(s-1)^n} \]
List $n = 100 \ p = .7$
Unsorted $= 100 \cdot 99 \cdot .35 = 65.35$
Sorted $= .7 \cdot \frac{(100+1)/2}{100} + .3 \cdot \frac{100/2 + 100/101}{100/2} = 50.647$
Hash table $s=10$
$100/10 + (1 - 100/10) \cdot .35 = 6.85$
s=100
$100/100 + (1 - 100/100) \cdot .35 = 1$
$100/1000 + (1 - 100/1000) \cdot .35 = 0.415$
<table>
<thead>
<tr>
<th>Item</th>
<th>Work</th>
<th>Largest</th>
<th>Total</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2/5</td>
</tr>
<tr>
<td>t</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1/2 (3/6)</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5/7</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>7/8</td>
</tr>
</tbody>
</table>
We need $p$

$p(index)\cdot \text{Length}(index)$ (length of list in each index)

Sum $p(index)\cdot (\text{length}(index)+(1-\text{length}(index))\cdot p/2)$

If above we assume $p(index) = 1/4$

$P = .8$

$(((3-2\cdot .4)+0+(5-4\cdot .4)+1)/4 = 1.65$

Assume $a=1$ $b=2$ ...

<table>
<thead>
<tr>
<th>Index</th>
<th>Possible Letters</th>
<th>Actual letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gnu 3/26</td>
<td>U 1</td>
</tr>
<tr>
<td>1</td>
<td>Ahov 4/26</td>
<td>O 1</td>
</tr>
<tr>
<td>2</td>
<td>Bipw 4/26</td>
<td>P 1</td>
</tr>
<tr>
<td>3</td>
<td>Cjqx 4/26</td>
<td>C 1</td>
</tr>
<tr>
<td>4</td>
<td>Dkry 4/26</td>
<td>R 1</td>
</tr>
<tr>
<td>5</td>
<td>Elsz 4/26</td>
<td>E 1</td>
</tr>
<tr>
<td>6</td>
<td>Fmt 3/26</td>
<td>Mt 2+(1-2)*.4 = 1.6</td>
</tr>
</tbody>
</table>

$3/26+4/26+4/26+4/26+4/26+4/26+1.6*3/26 = 1.0692$
<table>
<thead>
<tr>
<th>Index</th>
<th>Possible Letters</th>
<th>Actual letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gnu 3/26</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Ahov 4/26</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Bipw 4/26</td>
<td>I 1</td>
</tr>
<tr>
<td>3</td>
<td>Cjqx 4/26</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Dkry 4/26</td>
<td>KD 2+(1-2)*.4 = 1.6</td>
</tr>
<tr>
<td>5</td>
<td>Elsz 4/26</td>
<td>LE 2+(1-2)*.4 = 1.6</td>
</tr>
<tr>
<td>6</td>
<td>Fmt 3/26</td>
<td>M 1</td>
</tr>
</tbody>
</table>

\[
\frac{4}{26} + \frac{1.6}{4} + \frac{1.6}{4} + \frac{3}{26} = 0.7615
\]
<table>
<thead>
<tr>
<th>Index</th>
<th>Possibilities</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 12/50</td>
<td>4 8 12 16 20 24 28 32 36 40 44 48</td>
<td></td>
</tr>
<tr>
<td>1 13/50</td>
<td>1 5 9 13 17 21 25 29 33 37 41 45 49</td>
<td>13</td>
</tr>
<tr>
<td>2 13/50</td>
<td>2 6 10 14 18 22 26 30 34 38 42 46 50</td>
<td></td>
</tr>
<tr>
<td>3 12/50</td>
<td>3 7 11 15 19 23 27 31 35 39 43 47</td>
<td>27 31 11 47 35 19</td>
</tr>
</tbody>
</table>

\[0 \times \frac{12}{50} + 1 \times \frac{13}{50} + 0 \times \frac{13}{50} + 12 \times \frac{50}{50} \times (6 + (1 - 6) \times \frac{0.14}{2})
\]

\[\frac{13}{50} + \frac{12}{50} \times (6 + (1 - 6) \times \frac{0.14}{2}) = 0.3025\]

Alternatively

\[\frac{13}{50} \times 1 + \frac{(6 + 3 + 6 + 6 + 6 + 1 + 2 + 5 + 6 + 6 + 4)}{50} = 1.4\]

Letter frequencies:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>8.167% *2</td>
</tr>
<tr>
<td>b</td>
<td>1.492% *2</td>
</tr>
<tr>
<td>c</td>
<td>2.782% *1</td>
</tr>
<tr>
<td>d</td>
<td>4.253% *2</td>
</tr>
<tr>
<td>e</td>
<td>12.702% *2</td>
</tr>
<tr>
<td>f</td>
<td>2.228% *4</td>
</tr>
<tr>
<td>g</td>
<td>2.015% *2</td>
</tr>
<tr>
<td>h</td>
<td>6.094% *2</td>
</tr>
<tr>
<td>i</td>
<td>6.966% *4</td>
</tr>
<tr>
<td>j</td>
<td>0.153% *2</td>
</tr>
<tr>
<td>k</td>
<td>0.772% *2</td>
</tr>
<tr>
<td>l</td>
<td>4.025% *4</td>
</tr>
<tr>
<td>m</td>
<td>2.406% *1</td>
</tr>
<tr>
<td>n</td>
<td>6.749% *2</td>
</tr>
<tr>
<td>o</td>
<td>7.507% *2</td>
</tr>
<tr>
<td>p</td>
<td>1.929% *2</td>
</tr>
<tr>
<td>q</td>
<td>0.095% *2</td>
</tr>
<tr>
<td>r</td>
<td>5.987% *4</td>
</tr>
<tr>
<td>s</td>
<td>6.327% *2</td>
</tr>
<tr>
<td>t</td>
<td>9.056% *1</td>
</tr>
<tr>
<td>u</td>
<td>2.758% *3</td>
</tr>
<tr>
<td>Letter</td>
<td>Possibilities</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td>0</td>
<td>cfilorwux</td>
</tr>
<tr>
<td>1</td>
<td>adgjmsvp</td>
</tr>
<tr>
<td>2</td>
<td>behknqtzw</td>
</tr>
<tr>
<td>a</td>
<td>8.17%</td>
</tr>
<tr>
<td>b</td>
<td>1.49%</td>
</tr>
<tr>
<td>c</td>
<td>2.78%</td>
</tr>
<tr>
<td>d</td>
<td>4.25%</td>
</tr>
<tr>
<td>e</td>
<td>12.70%</td>
</tr>
<tr>
<td>f</td>
<td>2.23%</td>
</tr>
<tr>
<td>g</td>
<td>2.02%</td>
</tr>
<tr>
<td>h</td>
<td>6.09%</td>
</tr>
<tr>
<td>i</td>
<td>6.97%</td>
</tr>
<tr>
<td></td>
<td>0.15%</td>
</tr>
<tr>
<td></td>
<td>0.77%</td>
</tr>
<tr>
<td></td>
<td>4.03%</td>
</tr>
<tr>
<td></td>
<td>2.41%</td>
</tr>
<tr>
<td></td>
<td>6.75%</td>
</tr>
<tr>
<td></td>
<td>7.51%</td>
</tr>
<tr>
<td></td>
<td>1.93%</td>
</tr>
<tr>
<td></td>
<td>0.10%</td>
</tr>
<tr>
<td></td>
<td>5.99%</td>
</tr>
<tr>
<td></td>
<td>6.33%</td>
</tr>
<tr>
<td></td>
<td>9.06%</td>
</tr>
<tr>
<td></td>
<td>2.76%</td>
</tr>
<tr>
<td></td>
<td>0.98%</td>
</tr>
<tr>
<td></td>
<td>2.36%</td>
</tr>
<tr>
<td></td>
<td>0.15%</td>
</tr>
<tr>
<td></td>
<td>1.97%</td>
</tr>
<tr>
<td></td>
<td>0.07%</td>
</tr>
</tbody>
</table>
average: 2.27224
<table>
<thead>
<tr>
<th>Index</th>
<th>Possibilities</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4,8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,5,9</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2,6,10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3,7</td>
<td>3,7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Search time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>9/10 = 0.9</td>
</tr>
</tbody>
</table>
Merge sort
w,e,l,c,o,m,i,n,g

Ew(1) cl(1) mo(1) in(1) g(0)

Celw(3) imno(3) g(0)
Ceilmnow(7) g(0)
Cegilmnow(3)

20 comparisons

Q implementation
w,e,l,c,o,m,i,n,g

Ew(1) cl(1) mo(1) in(1)

Egw(2) clmo(2)

Eginw(4)

Cegilmnow(8)

20 comparisons

Merging (when equal use left one)
Woollingly

Ow(1) lo(1) il(1) gn(1) ly(1)

Loow(3) giln(3)

Lloowy(5)

Gillnoowy(6)
22 comparisons.

Heap property recursively root is the largest
Every level is totally filled in except the bottom which is filled in left to right

Example heap
100
Insert 200
First put it into the leftmost space in bottom level
(if full then left side next level down).

If bigger than parent switch with parent repeat until root or parent is >=

Michaely a-lowest z-highest

m
i

m
i
c

m
i
c
h

m
i
c
ha

m
i
e
ha
c

m
i
l
ha
c
e

y
m
l
ia
c
e
h

DQ
Top of tree is the result.
Take the right most element of the bottom and replace top with it.
Look at two children if the larger is larger than top switch with larger child repeat

HW
Take
c,o,m,p,u,t,e,r
Enqueue them all and dequeue them all.
Take
c,o,m,p,u,t,e,r
a has lowest priority, z has the highest.
Enqueue all the letters into a heap and dequeue 3 times. Show the heap after each enqueue and dequeue.

Review 11/23
Exam 11/25
Report 3 11/24
Project 4 12/2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>oc</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>ocm</td>
<td></td>
</tr>
<tr>
<td>c m</td>
<td>ocm</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>pomc</td>
<td></td>
</tr>
<tr>
<td>o m</td>
<td>pomc</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>upmco</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>upmco</td>
<td></td>
</tr>
<tr>
<td>p m</td>
<td>upmco</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>c</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>o</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>c</td>
<td>o</td>
<td>m</td>
</tr>
<tr>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ur</td>
<td>t</td>
<td>p</td>
</tr>
<tr>
<td>o</td>
<td>m</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dequeues

| c   | r   | t   |
| r   | t   | p   |
| o   | m   | e   |
|     |     |     |
| e   | r   | m   |
| p   | m   |
| o   | e   |
|     |     |     |
| c   | p   | m   |
|     | p   | o   |
|     | m   |

Index 0 is the root.
Left child of index is at 2*index+1
Right child of index is at $2 \times \text{index} + 2$

Parent of index is at $(\text{index}-1)/2$ (integer division)

Heapsort: (queue is before :)
:computer
c:omputer
oc:mputer
ocm:puter
pomc:uter
upmco:ter
uptcom:er
uptcome:r
urtpomec:
trmpoce:u
rpmeoc:tu
pomec:rtu
oemc:prt
mec:oprt
ec:mopr

\[ \text{c:emopr} \]
| a:rguing     | 0 |
| ra:guing     | 1 |
| rag:uing     | 1 |
| urga:ing     | 2 |
| urgai:ng     | 1 |
| urnaig:g     | 2 |
| u(rn)(ai)(gg): | 1 |
| r(in)(ag)g:u | 4 |
| n(ig)(ag):ru | 2 |
| igga:nru     | 3 |
| gga:inru     | 2 |
| ga:ginru     | 1 |
| a:ginru      | 0 |

20 comparisons

Quick Sort
Worst case $O(n^2)$
One of the worst cases is
1,2,3,4,5,6,...

c,o,m,p,u,t,e,r
Review CMP251 Exam2.doc – C

November 09

1. **40 Pts**: Take all the letters of "data structures" and show each enqueue and dequeue of heapsort. The letters d,a,v,i sort to
d
  da
  vad
  vida
  iad:v
  da:iv
  a:div

d:atastructures
da:tastructures
tad:astructures
tada:structures
tsd:aa:structures
t(st)(a)a:d:ru:ctures
t(st)(a)(dr):uctures
u(tt)(sa)(dr)a:ctures
u(tt)(sa)(dr)(ac):tures
u(tt)(st)(dr)(ac):ures
u(ut)(st)(dr)(ac)(at):res
u(ut)(st)(rr)(ac)(at)d:es
u(ut)(st)(rr)(ac)(at)(de):s
u(ut)(st)(rs)(ac)(at)(de)r:
u(tt)(st)(rs)(ac)(ar)(de):u
t(ts)(st)(re)(ac)(ar)d:uu
t(ts)(sr)(re)(ac)(ad):tuu
t(ss)(dr)(re)(ac)a:ttuu
s(sr)(dr)(ae)(ac):tttuu
s(rr)(dc)(ae)a:stttuu
r(re)(dc)(aa):sstttuu
r(de)(ac)a:rsstttuu
e(da)(ac):rssstttuu
d(ca)a:errsstttuu
ca:dderrasstttuu
aa:cderrasstttuu
a:acderrasstttuu

2. **40 Pts**: Take every letter of "data structures" and put it into a hash table of size 8 using the encoding (a - 1, b - 2, c - 3, ... z - 26). Find the
theoretical average number of comparisons to find a character. (case
independent) (assume all the letters are equally probable).

<table>
<thead>
<tr>
<th>Index</th>
<th>Letters</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hpx</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>aiqy</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>bjrz</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>cks</td>
<td>sc</td>
</tr>
<tr>
<td>4</td>
<td>dlt</td>
<td>dt</td>
</tr>
<tr>
<td>5</td>
<td>emu</td>
<td>ue</td>
</tr>
<tr>
<td>6</td>
<td>fnv</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>gow</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>Search</th>
<th>Letter</th>
<th>Search</th>
<th>Letter</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>i</td>
<td>1</td>
<td>q</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>j</td>
<td>1</td>
<td>r</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>k</td>
<td>2</td>
<td>s</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>l</td>
<td>2</td>
<td>t</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>m</td>
<td>2</td>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>n</td>
<td>0</td>
<td>vwx</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>o</td>
<td>0</td>
<td>y</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0</td>
<td>p</td>
<td>0</td>
<td>z</td>
<td>1</td>
</tr>
</tbody>
</table>

*Average # of searches: 23/26*

3. **20 pts:** Merge sort all the letters of “data structures”. How many
comparisons are necessary to do the sort.

datastructures
ad(1) at(1) st(1) ru(1) ct(1) ru(1) es(1)
aadt(3) rstu(3) crtu(3) es(0)
aadrsttu(6) cerstu(4)
aacderrsstttuu(13)

39 comparisons.
Draw a priority queue array after inserting these priorities:
27, 13, 31, 11, 47, 35, 19

String word;
word.toLowerCase();
word.println();

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>27</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>13</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>13</td>
<td>27</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>31</td>
<td>27</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>47</td>
<td>31</td>
<td>35</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>47</td>
<td>31</td>
<td>35</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>

Public class SearchTree
{
    Boolean empty = true;
    Comparable top;
    SearchTree smallers;  // everything in this tree is smaller than the top
    SearchTree largers;   // everything in this tree is larger than the top
    /** inserts a new item into the search tree */
    Public void insert(Comparable toInsert)
    {
        if (empty)
            { top = toInsert; empty = false;
              smallers = new SearchTree(); largers = new SearchTree;
            }
        Else if(toInsert.compareTo(top) > 0) largers.insert(toInsert);
        Else smallers.insert(toInsert);
    } // end insert
    Public boolean smallest() throws tantrum
    {
        if (empty) throw tantrum;
        If(smallers.empty) return top; else return smallers.smallest();
    } // end smallest
} // end SearchTree

1, 5, 2, 4, 3
\[ S(n) = \sum_{j=0}^{n-1} \left( \sum_{r=0}^{j} (S(n-1-r)+1) + \sum_{r=K+1}^{n-1} (S(r)+1) \right) \]

\[ = 1 + \frac{1}{n^2} \left( \sum_{k=0}^{n-1} \sum_{j=n-k}^{n-1} s(j) \right) + \frac{1}{n^2} \sum_{r=K+1}^{n-1} s(r) \]

\[ = 1 + \frac{1}{n^2} \left( \sum_{d=0}^{n-1} \sum_{i=d+1}^{n-1} s(i) \right) + \frac{1}{n^2} \sum_{r=K+1}^{n-1} s(r) \]

\[ = 1 + \frac{2}{n^2} \left( \frac{1}{d+1} \sum_{j=d+1}^{n-1} s(j) \right) + \frac{3}{n^2} \sum_{r=K+1}^{n-1} s(r) \]

\[ S(n) = 1 + \frac{2}{n^2} \sum_{r=K+1}^{n-1} s(r) \]

\[ S(n) = \frac{(n-1)^2}{n^2} \left( S(n-1) - S(n-1) (n-1) \right) \]

\[ = 1 + \frac{2}{n^2} \sum_{r=K+1}^{n-1} s(r) \]

\[ = \frac{2n-1}{n^2} + \frac{2(n-1)}{n^2} S(n-1) = S(n) - \frac{(n-1)^2}{n^2} S(n-1) \]

\[ = 2n-1 + \frac{n^2-1}{n^2} S(n-1) = S(n) \]

\[ = \frac{2n-1}{n^2} + \left( 1 - \frac{1}{n^2} \right) \frac{2n-2}{(n-1)^2} + \left( 1 - \frac{1}{n^2} \right) (1-\frac{1}{n^2}) \frac{2n-3}{(n-2)^2} + \ldots \]
\[ h^2 \left( \prod_{n=1}^{\infty} \frac{1}{1 - \frac{h^2}{(n-1)^2}} \right) = \prod_{n=1}^{\infty} \frac{\Gamma(n-1/2)}{\Gamma(n-1/2)} \]

\[ = \sum_{k=0}^{n-1} \frac{2(n-k)-1}{(n-k)^2} \left( \frac{1}{\left( \frac{1}{n-k} \right)^2} = x(n) \right) \]

\[ = \frac{2}{3} + \frac{3}{4} = \frac{5}{4} \]

\[ = \frac{5}{4} + \frac{2}{4} \cdot \frac{3}{4} + \frac{3}{4} \cdot \frac{3}{4} = \frac{17}{9} \]

\[ = \frac{17}{9} + \frac{15}{16} + \frac{3}{8} \cdot \frac{5}{16} + \frac{15}{16} \cdot \frac{3}{4} \]

\[ = \frac{21}{4} + \frac{25}{48} + \frac{20}{48} + \frac{30}{48} = \frac{105}{48} = \frac{53}{24} \]
\[
T(0) = 0
\]

\[
T(n) = \sum_{k=0}^{n} \left( \sum_{r=0}^{k} T(n-r) \right) + \sum_{r=k}^{n-1} T(r)
\]

\[
\eta(n+1) = \sum_{j=0}^{n-1} T(n+1-j)
\]

\[
T(n) = 1 + \sum_{j=n-k}^{n-1} T(j) + \sum_{r=k}^{n-1} T(r)
\]

\[
= 1 + \sum_{j=0}^{n} T(j) + \sum_{r=0}^{n-1} T(r)
\]
\begin{align*}
&\text{Let } n = n - k \\
&\text{for } k = 1, \ldots, n.
\end{align*}

\begin{align*}
T(n) &= 1 + \frac{2}{n(n+1)} \sum_{k=0}^{n-1} \sum_{r=k}^{n-1} T(r) \\
&\quad - \frac{n-1}{n+1} \left( 1 + \frac{2}{n(n-1)} \sum_{k=0}^{n-2} \sum_{r=k}^{n-2} T(r) \right) \\
&= \frac{2}{n+1} nT(n-1) + \frac{2}{n(n+1)} \left( T(n-1) + \sum_{k=0}^{n-1} T(n-1) \right) \\
&\quad - \frac{n-2}{n+1} \sum_{k=0}^{n-2} T(n-1) \\
T(n) &= \frac{n}{n+1} T(n-1) + \frac{2}{n+1} T(n-1) + \frac{2}{n+1} \\
&\quad + \frac{n-2}{n+1} \sum_{k=0}^{n-2} T(n-1) \\
T(n) &= \frac{2}{n+1} + T(n-1) \\
T(n) &\approx 2 \ln(n)
\end{align*}
Average Depths of tree nodes

\( \text{Ad}(n) = \text{Average depths of nodes in a tree of size } n \)

\( \text{Ad}(1) = 1 \)

\( \text{Ad}(2) = 1.5 \)

\( \text{Ad}(3) = \frac{17}{9} \)

\[
\text{Ad}(n) = \frac{1}{n} \left( \frac{1}{n} + \frac{n-1}{n} \left( 1 + \sum_{k=1}^{n-1} \frac{k}{n} \text{Ad}(k) + \frac{n-1-k}{n-1} \text{Ad}(n-1-k) \right) \right)
\]

\[
\text{Ad}(1) = 1, \quad \text{Ad}(2) = 1 + \frac{2}{3} \left( 1 + \frac{1}{3} \sum_{k=1}^{1} k \text{Ad}(k) \right) = 1 + \frac{2}{3} \left(1 + \frac{1}{3} \right) = 1 + \frac{2}{3} \left(1 + \frac{2}{3} \right)
\]

\[
\text{Ad}(3) = 1 + \frac{2}{3} \left( 1 + \frac{1}{3} \sum_{k=1}^{2} k \text{Ad}(k) \right) = 1 + \frac{2}{3} \left(1 + \frac{1}{3} \sum_{k=1}^{2} k \text{Ad}(k) \right) = 1 + \frac{2}{3} \left(1 + \frac{1}{3} \sum_{k=1}^{2} k \text{Ad}(k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]

\[
\text{Ad}(n) = 1 - \frac{(n-1)^2}{n^2} + \frac{2}{n^2} \left( \sum_{k=1}^{n-1} k \text{Ad}(k) + \sum_{k=1}^{n-1} k \text{Ad}(n-1-k) \right)
\]
\[ A_d(1) = \frac{2}{1} \cdot \frac{1}{2} = 1 \]
\[ A_d(2) = \frac{3}{2} \cdot \left( \frac{1}{2} + \frac{3}{6} \right) = \frac{3}{2} \]
\[ \frac{A_d(3)}{2} = \frac{4}{3} \left( \frac{1}{2} + \frac{3}{6} + \frac{5}{12} \right) = \frac{4}{3} \left( \frac{12}{12} + \frac{17}{12} \right) = \frac{4}{3} \left( \frac{29}{12} \right) = \frac{29}{9} \]
\[ = \frac{n+1}{n} \cdot \sum_{k=1}^{n} \frac{1}{k} - \frac{\sum_{k=1}^{n} \frac{1}{k}}{\sum_{k=1}^{n} \frac{1}{k}} \]
\[ = \frac{n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - \frac{\sum_{k=1}^{n} \frac{1}{k}}{\sum_{k=1}^{n} \frac{1}{k}} \right) \]
\[ = \frac{n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - 1 - \frac{1}{n+1} \right) \]
\[ = \frac{n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - 1 + \frac{1}{n+1} \right) \]
\[ = \frac{n+1}{n} \cdot \frac{\sum_{k=1}^{n} \frac{1}{k} - 1}{n+1} \]
\[ A_d(1) = -1 + 2 \cdot 2 \cdot \frac{1}{2} = 1 \]
\[ A_d(2) = -1 + \frac{3}{2} \cdot 2 \left( \frac{1}{2} + \frac{1}{4} \right) = -1 + \frac{3}{2} \cdot \frac{3}{4} = \frac{3}{2} \]
\[ A_d(3) = -1 + \frac{4}{3} \cdot 2 \left( \frac{1}{2} + \frac{1}{4} + \frac{1}{6} \right) = -1 + \frac{4}{3} \cdot \frac{17}{12} = \frac{17}{3} \]
\[ = \frac{-n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - 1 \right) = \frac{-n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - 1 + \frac{1}{n+1} \right) \]
\[ = \frac{-n+1}{n} \left( \sum_{k=1}^{n} \frac{1}{k} - 1 + \frac{1}{n+1} \right) \]
\[ \approx \frac{2}{n(n-1)} \cdot \frac{1}{\sin n(n-2)} - 2 \]
Quiz  put these letters into a binary search tree
c,o,m,e,n,t
Assume all the letters are equally likely, what is the average number of comparisons to search for a letter?

```
a 1
b 1
c 1
d 4
e 4
f 4
```
\[ \frac{81}{26} = 3.1154 \]
All balanced trees balance themselves with rotations.
4 types of rotations (but really 2 with 2 mirror images).

Every rotation needs to maintain search tree property
(everything smaller goes into the smallers; everything larger is
in the largers).

Variation in balanced trees is when it is unbalanced.
Also lowest unbalance is addressed first.

Rebalancing at the lowest level fixes all higher levels (AVL).

Left rotation:
Class HybridTree
{
    boolean empty;
    class Node
    {
        Comparable root;
        HybridTree smallers;
        HybridTree largers;
    };
    Node root;
    Public void leftRotation()
    {
        Node toNotLose = root;
        root = largers.root;
        toNotLose.largers.root = root.smallers.root;
        root.smallers.root = toNotLose;
    }
}

AVL tree property
Tree depth is the depth of the deepest node in the tree (# of nodes on the path from root to node).

Depth is 4
AVL property is the depth of the smallers and depth of the largers can only differ by 1.
AVL trees are search trees with the AVL property.

AVL insertion algorithm
Assumes tree starts with AVL property
If empty insert new item.
Otherwise AVL insert the new item into appropriate subtree.
At lowest level where AVL property is violated rebalance with rotations.
Rebalancing lowers depth and fixes any other inbalances that occur.
Right left imbalance (go right and then left to the deepest part of the tree). Rotate into right right imbalance and then left rotate.
AVL Tree

Regular BST

```plaintext
\[
\begin{align*}
\text{a-b-1-n} & \quad \text{e-i-3-o-9} \\
5 & \quad \text{s-3} \\
\text{r-2} & \quad \text{u-2} \\
4 & \quad \text{y-4} \\
+5 & \quad +6
\end{align*}
\]
```
\[
\begin{align*}
1 & \cdot 1 = n \cdot 4 \\
5 \cdot 3 & + 2 \cdot 7 = 3 \\
8 \cdot 3 & + 2 \cdot 5 \cdot 4 + 3 \cdot 3 + 1 \\
+ 3 & + 2 + 6 \cdot 3 \\
\frac{79}{26} & \approx 3
\end{align*}
\]