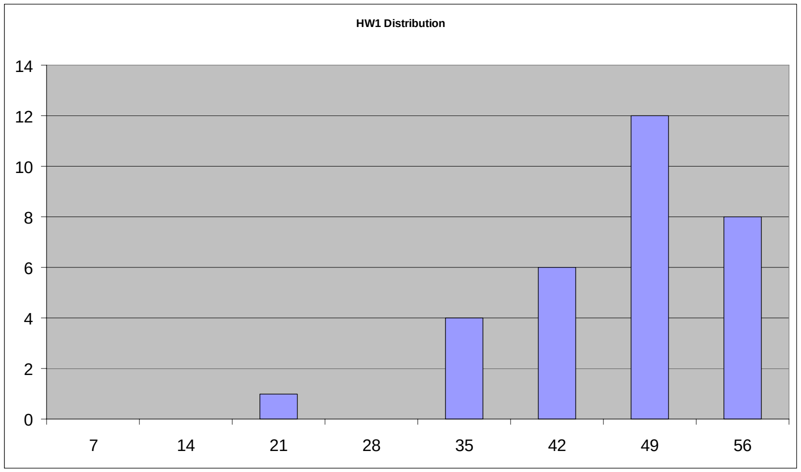
CSE 326: Data Structures

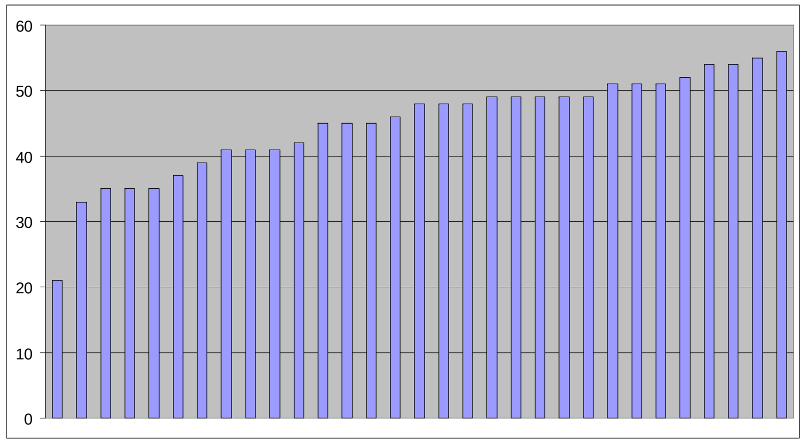
James Fogarty

Autumn 2007

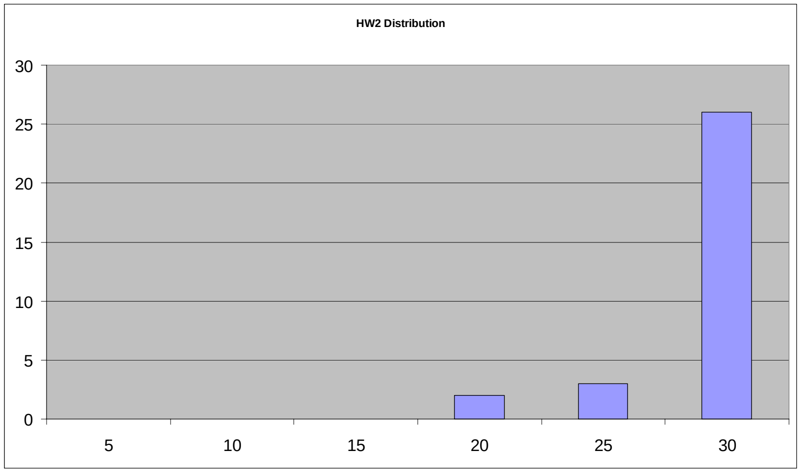
Lecture 15

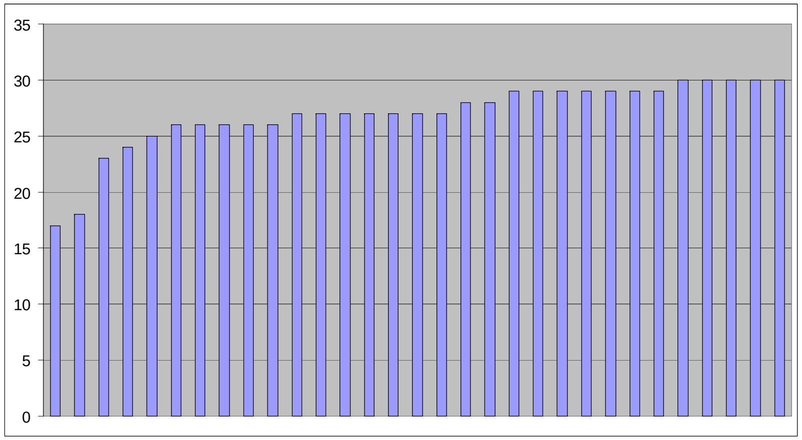
HW1 Distribution



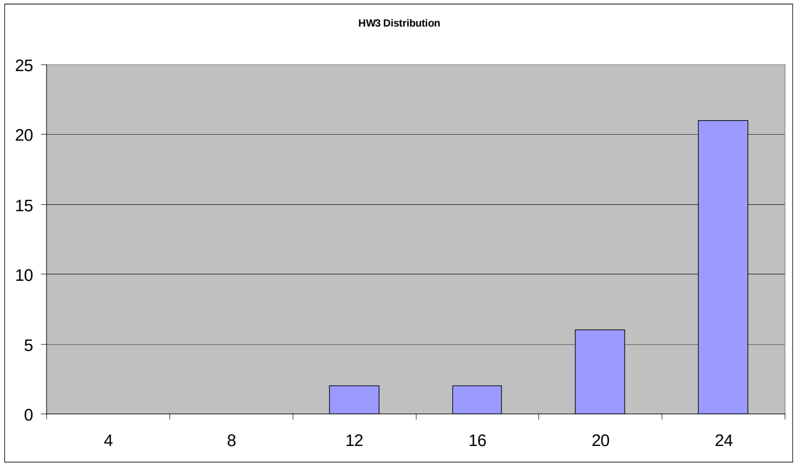


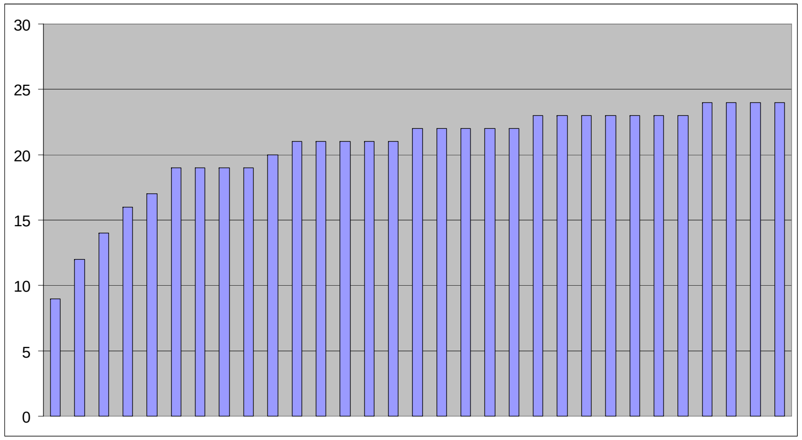
HW2 Distribution



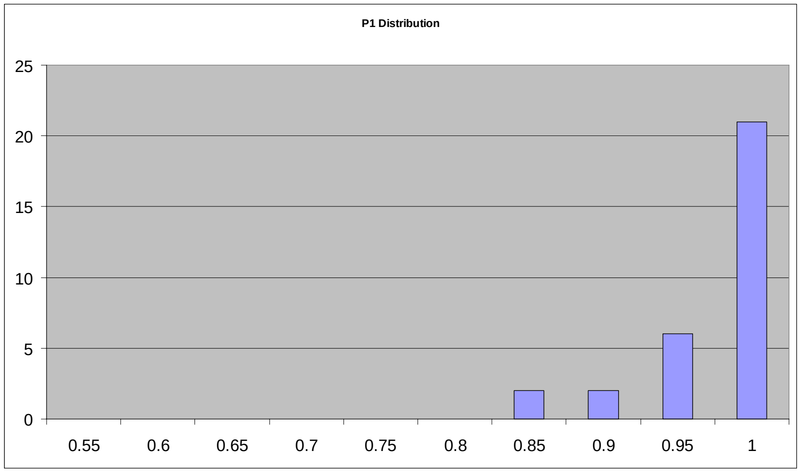


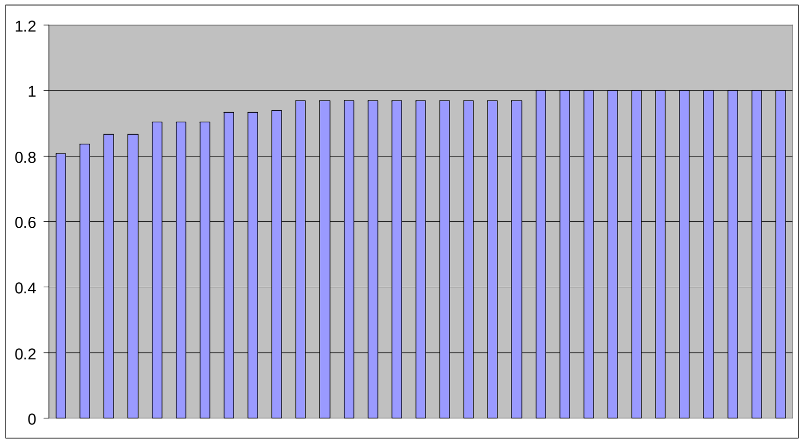
HW3 Distribution



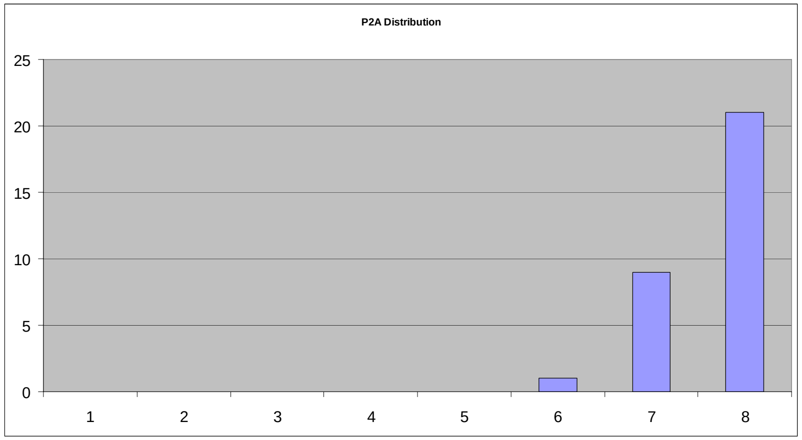


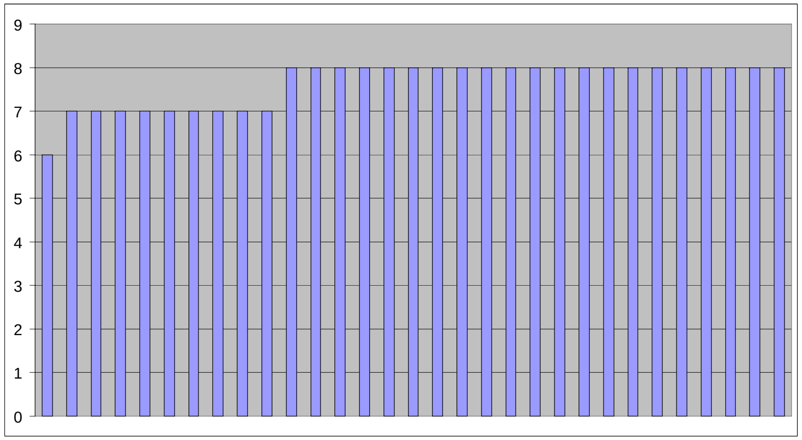
P1 Distribution



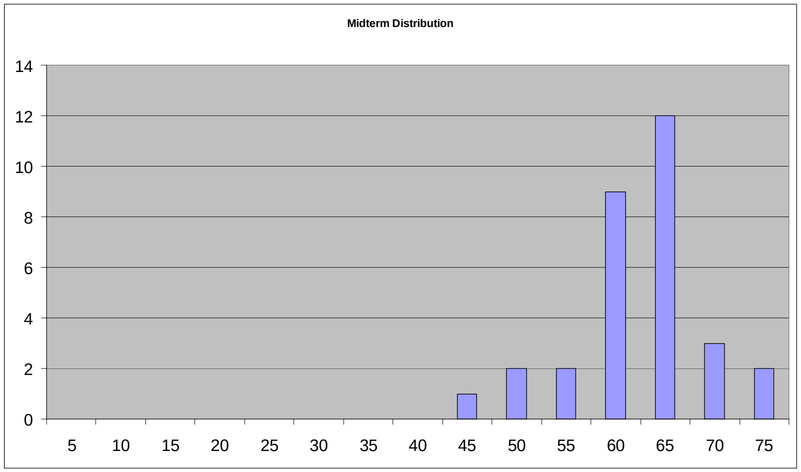


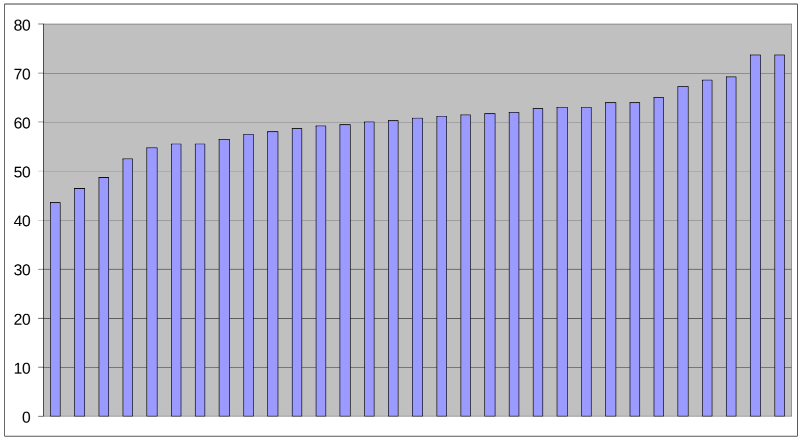
P2A Distribution



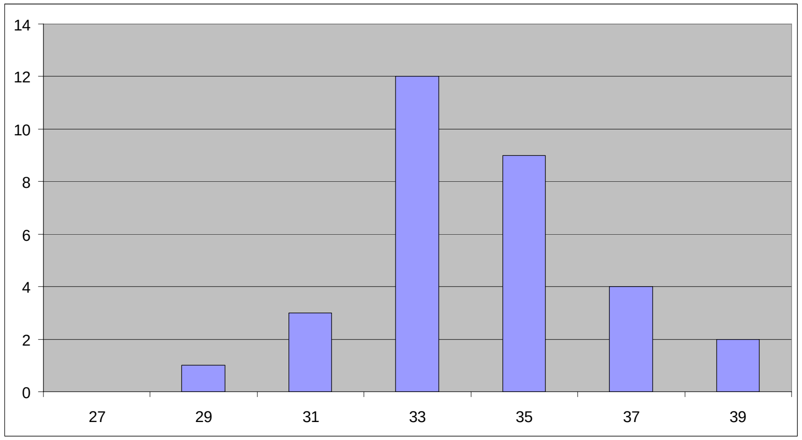


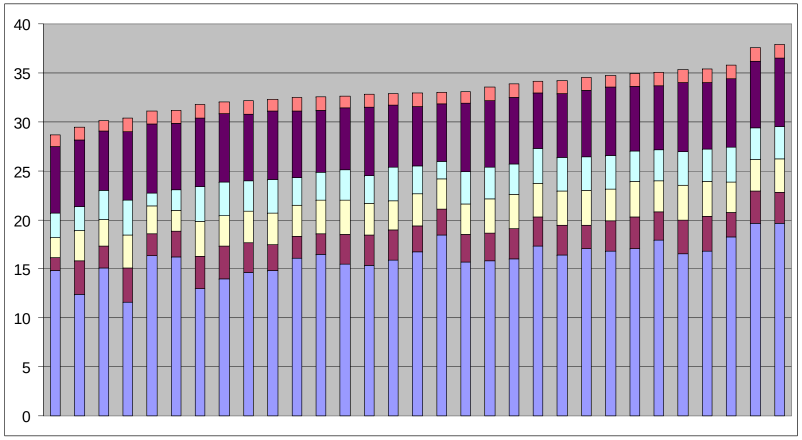
Midterm Distribution





Crudely Estimated Grades (out of 39 points so far)





1) 10 Points

List ADT

Name two potential implementations of the List ADT that were discussed in class, and provide O (Big-Oh) bounds on the running time of each operation.

Name	Insert Average	Find Average	Delete Average

Priority Queue ADT

Name an implementation of the Priority Queue ADT that was discussed in class and has the desired property. Provide O (Big-Oh) bounds on the running time of each operation.

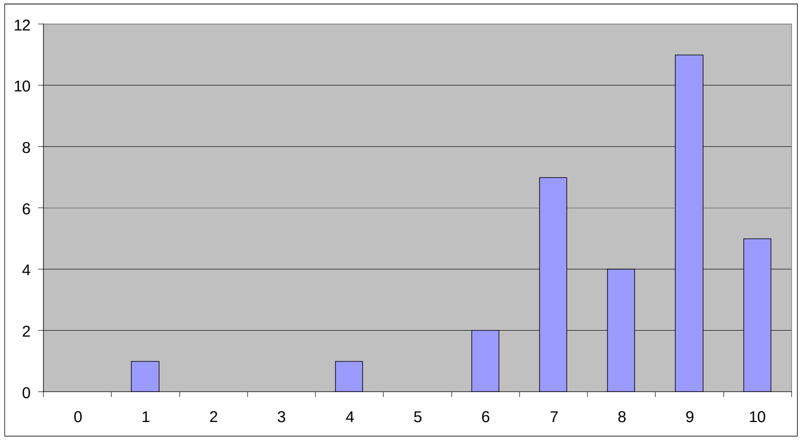
	Name	Insert Average	DeleteMin Average	Merge Average
Conceptually Simple				
Efficient Merging With Worst-Case Guarantees				
Efficient Merging With Amortized Costs				
Better Addresses Costs of Caching and Disk Access				

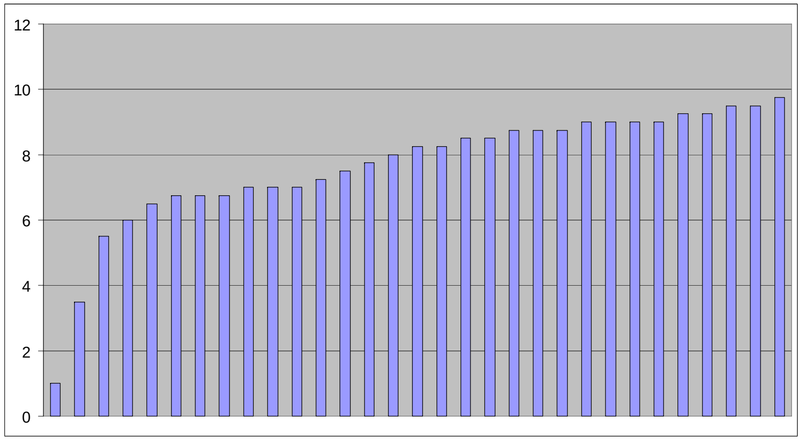
Dictionary ADT

Name an implementation of the Dictionary ADT that was discussed in class and has the desired property. Provide O (Big-Oh) bounds on the running time of each operation.

	Name	Average Insert	Worst-Case Insert	Worst-Case Series of *k* Inserts
Conceptually Simple				
Better Worst-Case Guarantees				
Efficient Amortized Costs				
Better Addresses Costs of Caching and Disk Access				

Q1 Distribution





2) 10 Points

Compute an appropriately tight O (Big-Oh) bound on the running time of each code fragment. Circle your answer for each code fragment.

a)

```
for(i = 0; i < n; i++) {
    sum++;
}
```

b)

```
for(i = 0; i < n; i++) {
    for(j = 0; j < i; j++) {
        sum++;
    }

    for(k = 0; k < i; k++) {
        sum++;
    }
}
```

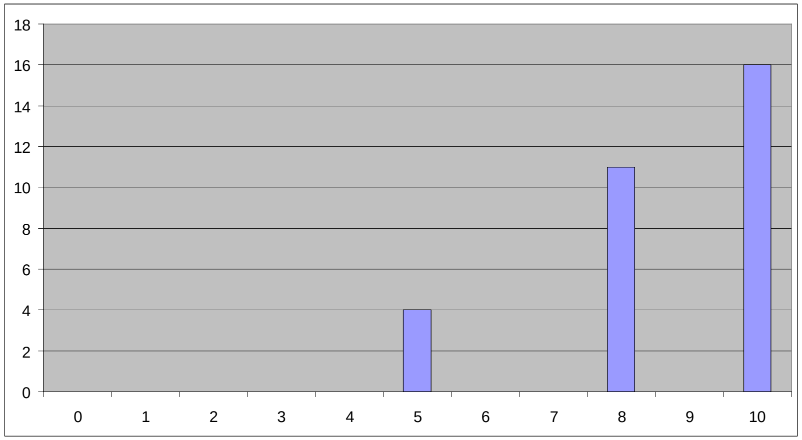
c)

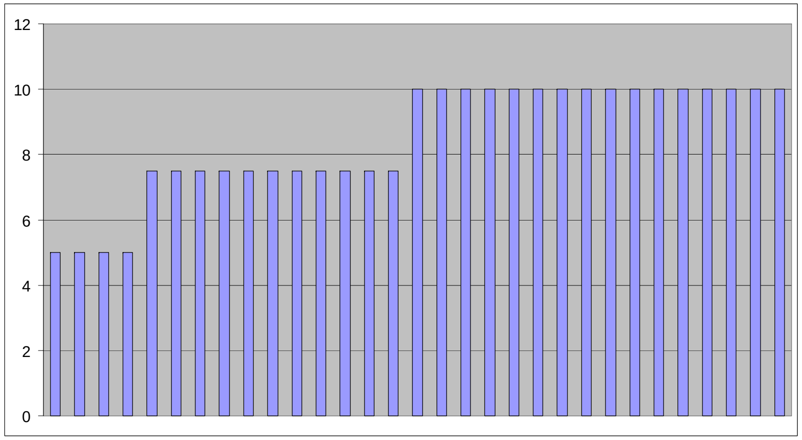
```
for(i = 0; i < n; i++) {
    for(j = 0; j < i * i; j++) {
        if(j % 2 == 0) {
            for(k = 0; k < i; k++) {
                sum++;
            }
        } else {
            for(k = 0; k < i * i; k++) {
                sum++;
            }
        }
    }
}
```

d)

```
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        if(i == j) {
            for(k = 0; k < i * i; k++) {
                sum++;
            }
        } else {
            for(k = 0; k < i; k++) {
                sum++;
            }
        }
    }
}
```

Q2 Distribution





Name: ________________________________

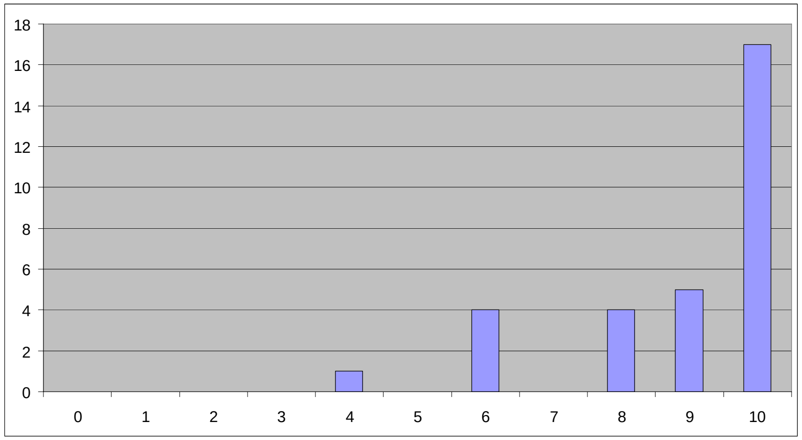
3) 10 Points

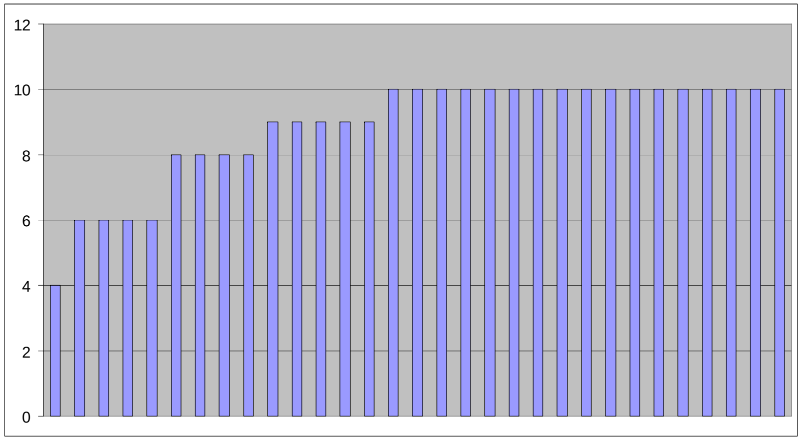
Prove by induction that:

$$1+4+9+\ldots+n^2=\frac{n(n+1)(2n+1)}{6}$$ for every positive integer n

If you find yourself unable to factor a polynomial, recall that you know what the polynomial should factor to. Multiplying what the factors should be and showing that the result is equal to your polynomial would be appropriate.

Q3 Distribution





Name: ______________________________

4) 10 Points

A level-order traversal visits each node in a tree according to the depth of that node. For example, the nodes in this binary tree:

```
      6          (depth 0)
     / \
    3   7        (depth 1)
   / \   \
  2   4   9      (depth 2)
       \
        5        (depth 3)
```

would be visited in the order 6, 3, 7, 2, 4, 9, 5. The in-order, pre-order, and post-order traversals discussed in class are based in an implicit stack, created by the recursion. This is why I noted in lecture that you cannot perform a level-order traversal using recursion. You will write pseudocode that uses a queue to print nodes in a binary tree according to a level-order traversal.

a) Define an appropriate node type (storing an int as the data element in each node):

```
class Node {







}
```

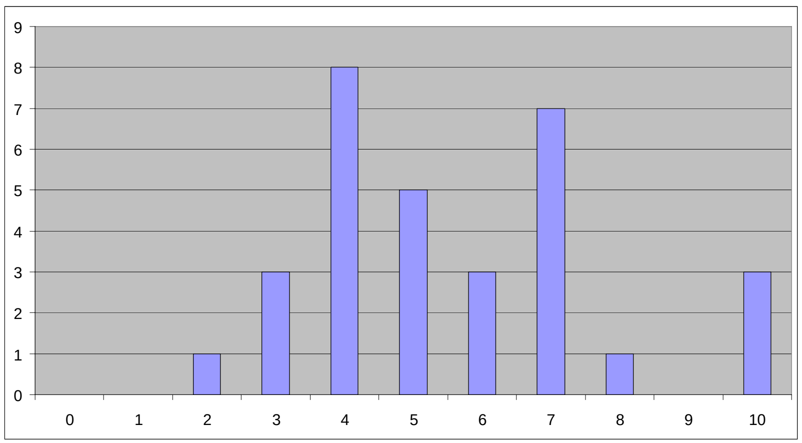
b) Define an appropriate interface for a node queue (no need to worry about generics):

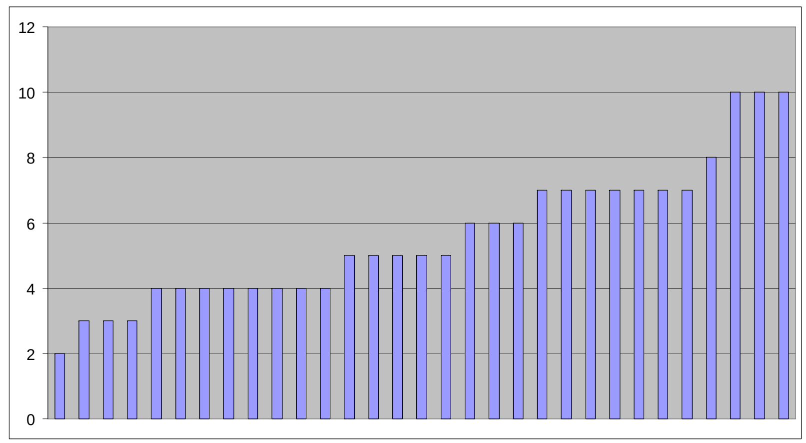
```
interface NodeQueue {







}
```


Name: ______________________________

c) Write pseudocode using your Node and NodeQueue types to implement a level-order traversal, printing the int data element from each node as it is visited.

Q4 Distribution





Name: ________________________________

5) 10 Points

Consider the following binary heap:

```
         3
       /   \
     5       6
   /   \   /   \
 20    7  13     9
```

Perform the following operations in order, drawing the resulting heap after each operation and using it as the starting point for the next operation. You need only show the result of each operation for full credit, but we will only be able to award partial credit if you show your work. If the space available here is insufficient, use the back of this sheet (clearly labeling your work). Circle the state of the heap after each operation so that we can distinguish it from your intermediate work.

a) Insert 25

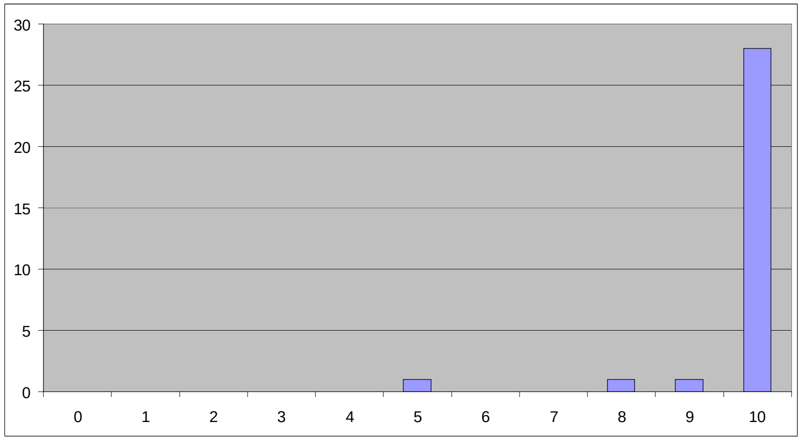
b) Insert 2

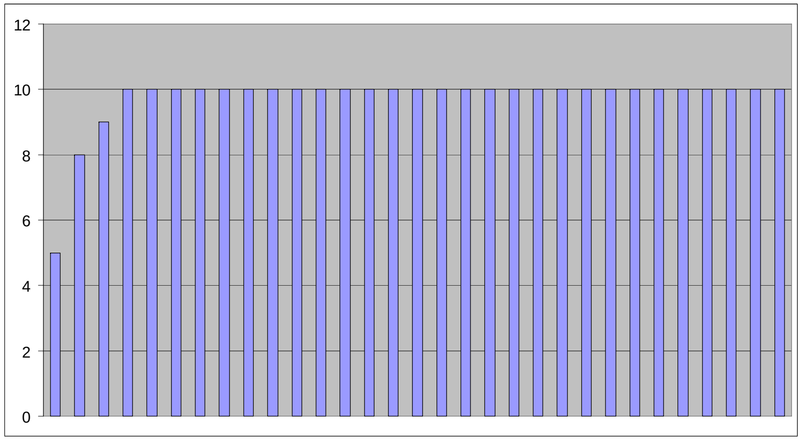
c) DeleteMin

d) DeleteMin

e) DeleteMin

Q5 Distribution





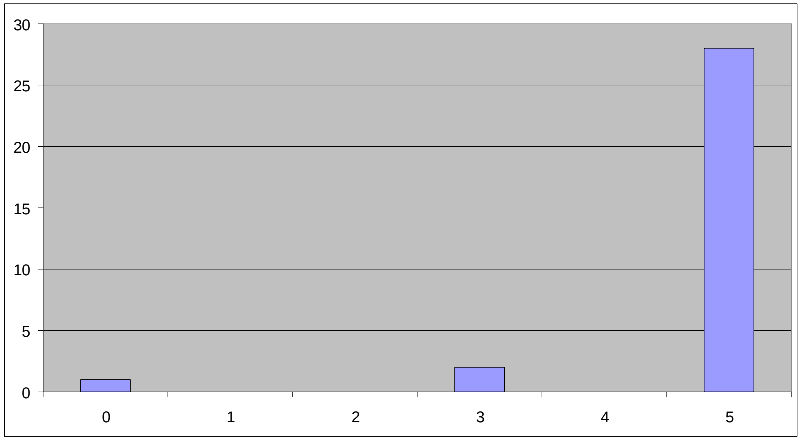
6) 5 Points

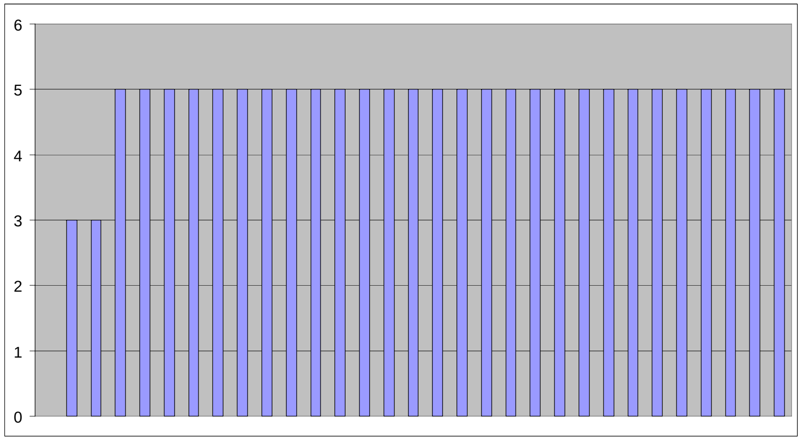
Consider the following binomial queue, currently containing 3 trees:

```
  1_______                    6__                    3
  |  \    \                   |  \
  5   9    4___               7   10
      |    |   \                   |
      13  17   11                  14
               |
               21
```

Perform a deleteMin operation on this binomial queue. You need only show the result of the operation for full credit, but we will only be able to award partial credit if you show your work. If the space available here is insufficient, use the back of this sheet (clearly labeling your work). Circle the final state of the queue so that we can distinguish it from your intermediate work.

Q6 Distribution





Name: ______________________________

7) 10 Points

Consider the following AVL tree:

```
    4
   / \
  0   5
   \
    1
```

Perform the following operations in order, drawing the resulting tree after each operation and using it as the starting point for the next operation. You need only show the result of each operation for full credit, but we will only be able to award partial credit if you show your work. If the space available here is insufficient, use the back of this sheet (clearly labeling your work). Circle the state of the tree after each operation so that we can distinguish it from your intermediate work.

a) Insert 3

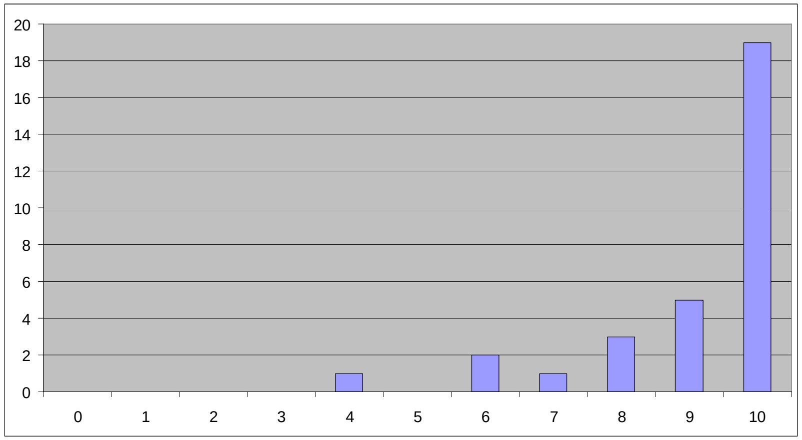
b) Insert 2

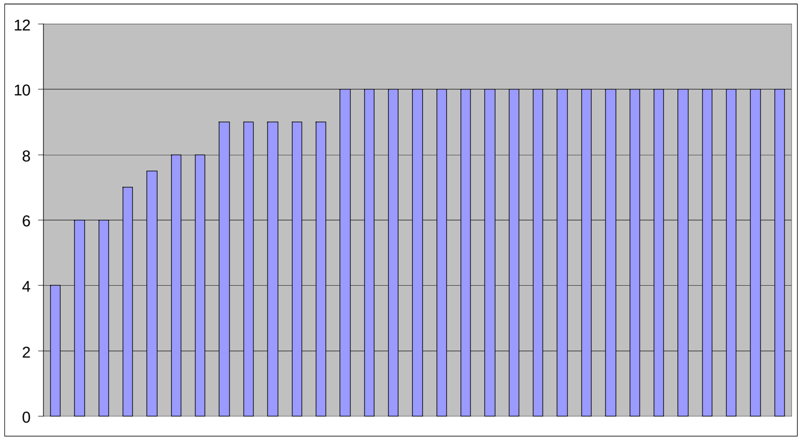
c) Insert 6

d) Delete 3

(We did not cover general AVL deletion, but you should know how to perform this delete)

Q7 Distribution





Name: ______________________________

8) 10 Points

Consider the development of a B-tree (specifically, the variant of B-trees discussed in class and in Chapter 4). Assume that the machine your B-tree will run on stores and retrieves data in blocks of 1024 bytes and that storing a reference requires 4 bytes. Assume that you are *not* storing sibling references.

You will be storing personal information indexed by a person's Social Security Number (SSN). An SSN is stored in 9 bytes. The exact information being stored about each person is classified, but requires a total of 128 bytes (including space for the SSN identifying the record).

You should provide a numeric answer for questions a, b, and c, but providing the formula you used to obtain your answer will allow us to give full or partial credit if your answer is incorrect due to a simple arithmetic error.

a) Choose an appropriate value for *M*, the branching factor of internal nodes.

b) Choose an appropriate value for *L*, the number of elements per leaf.

c) What is the maximum number of records your B-tree could contain if its height is 2?

d) In terms of *M*, *L*, and *h*, what is the maximum number of records a B-tree of height *h* might contain?

Extra Credit: (3 points) In terms of *M*, *L*, and *h*, what is the minimum number of records a B-tree of height *h* must contain?

Q8 Distribution