Michael Kaiser
Assignment 2
Comp 510
Fall 2012

17.1-3
Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

Let c_i be the cost of the ith operation.

c_i = {i if i is an exact power of 2
{1 otherwise

In table form, we have:

Operations	Cost
1	1
2	2
3	1
4	4
5	1
and so on...	

n operations will then cost:

$$\sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lg n} 2^j = n + (2n - 1) < 3n$$

thus we have,

Average cost of operations = Total cost (3n) / number of operations (n) < 3.

And by aggregate analysis, the amortized cost per operation is O(1).

17.2-1
Suppose we perform a sequence of stack operations on a stack whose size never exceeds k. After every k operations, we make a copy of the entire stack for backup purposes. Show that the cost of n stack operations, including copying the stack, is O(n) by assigning suitable amortized costs to the various stack operations.

From the description, we should have PUSH, POP and COPY.
If we assign a value of \$2 to each PUSH and POP, and use \$1 of it for each call, we have a \$1 credit stored. When we reach k operations, we will have \$k credits to pay for the copy on the stack. Since the amortized cost of each operation is O(1) and the amount is never negative, the total cost of n operations is O(n).

17.2-2
Redo Exercise 17.1-3 using an accounting method of analysis.

Let c_i be the cost of the ith operation.

c_i = {i if i is an exact power of 2
{1 otherwise

We will charge each operation $3:
If i is not a power of 2, pay $1 and store $2 as credit.
If i is a power of 2, pay $i with credit.

Operation	Cost	Actual cost	Credit remaining
1	3	1	2
2	3	2	3
3	3	1	5
4	3	4	4
5	3	1	6

Because the amortized cost is $3 per operation, the sum of all amortized c_i is = 3n.
From 17.1-3 we know that the sum of all c_i < 3n, thus the amount of credit is never negative.

Therefore, since the amortized cost of each operation is O(1), and the amount of credit never goes negative, the total cost of n operations is O(n).

18.2-1
Show the results of inserting the keys
F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E in order into an empty B-tree with minimum degree 2. Draw only the configurations of the tree just before some node must split, and also draw the final configuration.

I apologize ahead of time if this is unclear but this is the only way I could think of writing this out

1.
[F][Q][S]

2.
[Q][ ][ ]
[C][F][K] [S][ ][ ]

3.
[F][Q][ ]
[C][][] [H][K][L] [S][T][V]

4.
[F][Q][T]
[C][ ][ ] [H][K][L] [S][ ][ ] [V][W][ ]

5. (Combined a few steps)

```
                              [Q][ ][ ]
         [F][K][ ]                              [T][ ][ ]
[C][ ][ ]   [H][ ][ ]   [L][ ][ ]         [S][ ][ ]         [V][W][ ]
```

6.

```
                                   [Q][ ][ ]
         [F][K][ ]                              [T][ ][ ]
[C][ ][ ]   [H][ ][ ]   [L][M][N]       [R][S][ ]        [V][W][ ]
```

7.

```
                                        [Q][ ][ ]
                  [F][K][M]                             [T][ ][ ]
[C][ ][ ]  [H][ ][ ]  [L][ ][ ]  [N][P][ ]          [R][S][ ]        [V][W][ ]
```

8.

```
                                        [Q][ ][ ]
                  [F][K][M]                                    [T][ ][ ]
[A][B][C]  [H][ ][ ]  [L][ ][ ]  [N][P][ ]                [R][S][ ]        [V][W][X]
```

9.

```
                                        [Q][ ][ ]
                  [F][K][M]                                    [T][W][ ]
[A][B][C]  [H][ ][ ]  [L][ ][ ]  [N][P][ ]        [R][S][ ]       [V][ ][ ]        [X][Y][ ]
```

10. (Combined a few steps)

```
                                        [K][Q][ ]
            [B][F][ ]                   [M][ ][ ]                      [T][W][ ]
[A][ ][ ]   [C][D][ ] [H][ ][ ]       [L][ ][ ]   [N][P][ ]         [R][S][ ]  [V][ ][ ]  [X][Y][ ]
```

11. Final configuration:

```
                                        [K][Q][ ]
            [B][F][ ]                   [M][ ][ ]                       [T][W][ ]
[A][ ][ ]   [C][D][E] [H][ ][ ]       [L][ ][ ]   [N][P][ ]         [R][S][ ]  [V][ ][ ]  [X][Y][Z]
```

18.2-2
Explain under what circumstances, if any, redundant DISK-READ or DISK-WRITE operations occur during the course of executing a call to B-TREE-INSERT.

A DISK-WRITE operation can be redundant whenever the root node is changed, (so when the size of the tree is h and there are full nodes to the h-1 level).
When a B-TREE-SPLIT-CHILD function is performed recursively, we don't perform the DISK-READ operation.
Therefore, there isn't a redundant DISK-READ or DISK-WRITE on a B-TREE-INSERT.

References:

Class Textbook and solution manual (3rd ED)