ID:80585887

CS 2302

Dr. Olac Fuentes

TA: Zakia Al Kadri

I. Introduction

The problem presented this week in Lab 4 was to perform a series of operations to extract, traverse, print, and search for specific elements in a B-tree. These operations represented half of the lab experimentation while the other half of the experimental exercises was to draw graphically the B-tree making the use of the B-tree. The problems presented in the document for Lab 4 entailed the analysis of time complexities, the understanding of help methods, like that of counting nodes in the tree. All in conjunction with drawing the three tested my ability to make precise modifications to either the problems presented in class or the algorithms that already were used for Lab 3.

II. Proposed Solution (By order of presentation in document)

Problem 1:

For this specific problem I was unable to present a successful solution to the problem. First I tried to print the element in the tree linearly, that is level by level, and I also thought about putting the tree to sequentially divide the "X" range (0 to 100) by the number of items in that specific level of the tree. Nevertheless, I could not come up with a successful solution for that problem even tough the I was able to create 50% of what was necessary for the algorithm.

Problem "a)":

In the case of problem "a)", I was able to construct a solution by analyzing the "print-in-ascending-order" algorithm

Problem "b)":

For problem "b)", I realized that the minimum element of the B-tree is also found at the left-most position of the tree, that is that at every level the item at index "0" is the smallest. So, the recursion of the program goes to the first child of each node and checks the condition of the level variable being equal to zero. If this is true then it returns the first item of that BTreeNode and returns the value recursively to where the call is made.

Problem "c)":

In the case of problem "c)" the logic of problem "b)" applies but just in the search of the maximum element, that changes the logic of search to the right side and returns the last element in the item array of the rightmost BTreeNode at the specific level chosen by the user.

Problem “d)”:

For problem “d)” the execution has to count how many items there are in the tree. This is made recursively by calling a new recursion with the next level of recursion and returning the “T.n” of that node and adding up all the number of items of the child of that specific tree. This mechanically, once executed, provides the user with the number of elements in the array, at a given depth.

Problem “e)”:

For problem “e)” the execution basically has to find the elements at a given level and then print them out. This is made by the basic recursion process of subtracting one to the level integer provided as a parameter in the method body. If the count reaches zero then the execution has found a node that is at the desired level and therefore it should print its elements out.

Problem “f)”:

For problem “f)” the execution recursively checks every node and determines if its length is full, that is “5”, and returns a 1 or a zero in any other case. This problem makes the use of a similar code to that of problem “d)” to traverse the tree and check conditions to perform operations.

Problem “g)”:

For problem “g)” the execution recursively checks if every node is a leaf and determines if its length is full, that is “5”, at the same time and returns a 1 or a zero in any other case. This problem makes the use of a similar code to that of problem “f)” to traverse the tree and check conditions to perform operations. The difference between problem f and g is fundamentally just to check for leaves instead of any node.

Problem “h)”:

For problem “h)” the execution recursively makes a search in the tree, it actually implements the same code as in search operations with the difference that if the recursion makes necessary to search at a sub-level then it makes a recursive call that will return an integer value, that integer will be negative -1 if the element that is being searched is not present or else the level integer value of the tree at which the element is found.

Problem “i)”:

For problem “i)” the method created performs a search on the tree, with the difference that if the element is found at a specific node then all the elements at that node are printed with the use of a for loop.

III. Experimental Results

Problem 1:

To obtain this readings I make the use of a sorted array of different lengths was used to produce a tree and then the incomplete version of printing the tree was used for testing.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
                40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
                37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                 1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: draw_tree(T,0,100, y_max-5, y_max/B.height, 0);

Calls	Time (nanoseconds)
S Array	15785674
S1 Array	16395174
D Array	17467861
F Array	26671613

Problem “a)”:

To obtain the results for problem “a)” readings I made the use of sorted arrays of different lengths then the search with iteration was used, but always searching for the last element of the tree.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
                40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
                37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                 1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: printAscendingtoArray(T);

Calls	Time (nanoseconds)
S Array	186023
S1 Array	218195
D Array	276065
F Array	272851

Problem “b)”:

To obtain the results for problem “b)” readings I made the use of sorted arrays of different lengths but the execution has to reach level 2 and find the smallest value at this level.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
          40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
          37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
          -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
           1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: printSmallest(T,2);

Calls	Time (nanoseconds)
S Array	8820
S1 Array	10529
D Array	13845
F Array	14023

Problem “c)”:

For the obtention of time complexity times for problem “c)” a tree is created and then it asked to find its largest at level 2.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
               40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
               37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
               -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: printLargest(T,2);

Calls	Time (nanoseconds)
S Array	11880
S1 Array	12369
D Array	11167
F Array	13151

Problem "d)":

For problem "d)" a number of differently-sized arrays were used to detect the number of nodes at a specific level, in this case the level was always "2".

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
          40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
          37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
          -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
           1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: printNodeDepth(T,2);

Calls	**Time (nanoseconds)**
S Array	9879
S1 Array	10403
D Array	13720
F Array	12326

Problem “e)”:

For problem “e)” a number of differently-sized arrays were used to detect the number of nodes at a specific level, in this case the level was always “2”, so after detecting the nodes at this level the execution printed them out.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
          40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
          37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
          -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
           1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: printDepth(T,2);

Calls	Time (nanoseconds)
S Array	43638
S1 Array	60107
D Array	71541
F Array	71908

Problem “f)”:

For problem “f)” a number of differently-sized arrays were used to detect the number of nodes that had a full sized “items” array in the tree. The condition to check is the length of the items array which must be 5 to say it is full.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
                40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
                37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                 1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: nodesFull(T);

Calls	Time (nanoseconds)
S Array	4800
S1 Array	5898
D Array	6653
F Array	6124

Problem “g)”:

For problem “g)” a number of differently-sized arrays were used to detect the number of leaves, or terminal nodes, that had a full sized “items” array in the tree. The condition to check is the length of the items array which must be 5 to say it is full.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
          40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
          37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
          -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
           78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
           1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: leavesFull(T);

Calls	Time (nanoseconds)
S Array	3677
S1 Array	4677
D Array	5023
F Array	5820

Problem “h)”:

For problem “h)” a number of differently-sized arrays were used to create a tree, and then the execution/recursive method was called to search for a number specifically at the tree so it returned the depth at which it can be found or else return -1. For the purposes of this experiment I tried to locate a non existen number in the tree “10000”.

Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
              40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
              37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
               78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
              -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
               78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
               1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: searchLevel(T,10000);

Calls	Time (nanoseconds)
S Array	8708
S1 Array	9439
D Array	10516
F Array	12935

Problem "i)":

For problem "i)" a number of differently-sized arrays were used to create a tree, and then the execution/recursive method was called to search for a number specifically at the tree so it printed the numbers at the same node at which any number that is being searched-for is found. For this experiment the highest number in each tree was used.

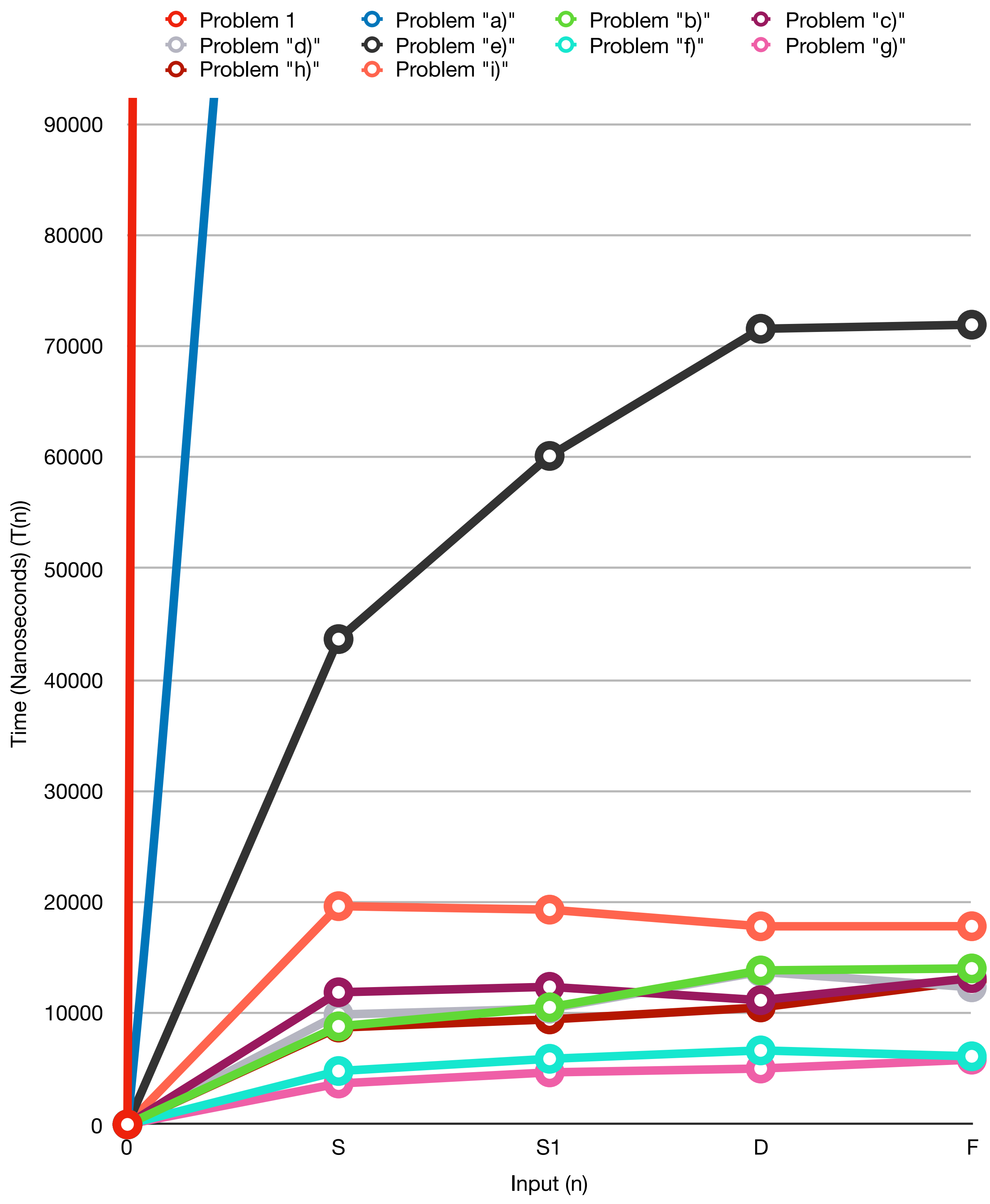
Calls:

```
int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37,
                40, 42, 50, 16 }
int [] S1 ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34,
                37, 40, 42, 50, 16,100,200,300,-10,20,19 };
int [] D = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                -32,1000,320,-10000};
 int [] F = {10, 15, 4, 8 ,9, 5, 2, 3, 7, 12, 18, 1,24, 60 ,6,
                 78,115,31,76,148,23,-15,-20,-30,-64,-128,-32,
                 1000,320,-10000, 34,56,61,64,90,83,85,45,67,34};
```

For all different magnitudes the call for the duplicate evaluation was: SearchBTree(T, highestNumber);

Calls	Time (nanoseconds)
S Array	19623
S1 Array	19310
D Array	17811
F Array	17819

Graphing of Time Complexities (T(n)).



IV. Appendix

BTree, BTreeNode, and StdDraw codes are not included in this appendix as they are of reference and non-fundamental for the educational purposes of this lab.

```
/////////////////////////////////////////
//Author: Oscar Galindo Molina          //
//ID: 80585887                          //
//CS 2302                               //
//Professor: Dr.Olac Fuentes            //
//TA: Zakia Al Kadri                    //
//Lab 4                                 //
/////////////////////////////////////////
import java.util.Random;
public class BTreeTest
{
  public static int num[];
  public static int index = 0;
  public static BTreeNode start;
  public static void printAscending(BTreeNode T)
  {
     //Prints all items in the tree in ascending order
    if (T.isLeaf)
    {
      for(int i =0; i<T.n;i++)
        System.out.print(T.item[i]+" ");
    }
    else
    {
      for(int i =0; i<T.n;i++)
      {
        printAscending(T.c[i]);
        System.out.print(T.item[i]+" ");
      }
       printAscending(T.c[T.n]);
    }
  }

   public static int m1(BTreeNode T)
  {
    if (T.isLeaf)
      return T.n;

    return T.n + m1(T.c[T.n]);
  }

  public static int m2(BTreeNode T)
  {
    int sum =0;
    for(int i =0; i<T.n;i++)
       if (T.item[i]%2 ==0)
            sum++;
    if (!T.isLeaf)
      for(int i =0; i<=T.n;i++)
    sum += m2(T.c[i]);
    return sum;
   }

  public static int m3(BTreeNode T, int k)
  {
     int i=0;
    while ((i<T.n )&&(k>T.item[i])) //Sequentially search for k
      i++;
    if (T.isLeaf)
     return i;
    else
      return i + m3(T.c[i], k);
  }

  public static void main(String[] args)
  {
    //int [] S ={ 21, 2, 20, 12, 11, 18,  9, 17, 25, 30, 7, 13,  5, 14, 10,  6,  3, 19, 15, 16,
26, 8,  4, 24, 31,1};
    //int [] S = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int [] S ={8, 9, 11, 4, 7,  12, 13, 17, 24, 15, 27, 28, 30, 33, 34, 37, 40, 42, 50, 16 };
     int x_max =100;
    int y_max =100;
    StdDraw.setXscale(-10, 110);
    StdDraw.setYscale(-10, 110);
    StdDraw.setPenColor(StdDraw.BLACK);
    BTree B = new BTree(3);
    //Build B-tree from array
    for (int i=0;i<S.length;i++){
      System.out.println("New element: "+S[i]);
      B.insert(S[i]);


      System.out.println("Tree (pre-order traversal)");
        B.printNodes();
      System.out.println("*********************");
    }
    BTreeNode T = B.root;
    num = new int[countItems(T)];
```

```
//draw_tree(T, 0, x_max, y_max-5, (y_max-10.0)/(B.height+1),0);
start = T;
//toArray(T, num.length/2);
printAscendingtoArray(T);

for(int i = 0;i < num.length; i++)
  System.out.print(num[i]+" ");
draw_tree(T, 0, 100, y_max-5, y_max/(B.height+1), 0);

SearchBTree(T,50);
//System.out.println(printNodeDepth(start,1));
//printAscending(T);
//System.out.println();
//System.out.println(m1(T));
//System.out.println(m2(T));
//System.out.println(m3(T,42));
//System.out.println(T.item);
//System.out.println(T.n);
//System.out.println(T.item[T.n-1]);
//printLargest(T, 1);
//System.out.println(T.item[0]);

//SearchBTree(T, 50);
//System.out.println(searchLevel(T, 4));
//System.out.println(searchLevel(T, 27));
//printSmallest(T, 1);
//printLargest(T,1);
//printDepth(T,2);
//System.out.println(printNodeDepth(T, 2));
// treeExtract(T,num.length/2);
// for(int i = 0; i < num.length; i++)
```

```
        // System.out.print(num[i]+" ");
      /*
      //Build B-tree with random elements
      Random rn = new Random();
      BTree R = new BTree(3);
      for (int i=0;i<30;i++){
         int s = rn.nextInt(100);
         System.out.println("New element: "+s);
            R.insert(s);
          System.out.println("Tree (pre-order traversal)");
         R.printNodes();
         System.out.println("*********************");
      }
      T = R.root;
      printAscending(T);
      */
   }


   public static void draw_tree(BTreeNode T, double x0, double x1, double y, double
y_inc, int level)
   {
     if(T ==null)
        return;
     double xm = (x0+x1)/2;
     double yn = y-y_inc;//correct

     // for(int i = 0; i < T.n+1; i++,)
     // {
        // printLinear(T,y,x1,level+1);
     // }
      // if(T.isLeaf)//correct
      // {
        // for(int i = 0; i < T.n; i++,xm+= 6)
        // {
          // StdDraw.setPenColor(StdDraw.WHITE);
          // StdDraw.filledCircle(xm,y, 3);
          // StdDraw.setPenColor(StdDraw.BLACK);
          // StdDraw.circle(xm,y,3);
          // StdDraw.text(xm,y,Integer.toString(T.item[i]));
        // }
      // }
     // else //T!isLeaf
     // {
      // //double k = x1/(T.n);
      // //double j = 0;
      // for(int i = 0; i < T.n+1; i++,j = j + k)
      // {
        // StdDraw.line(xm,y,j,yn);
        // draw_tree(T.c[i],x0,j,yn,y_inc,level+1);
      // }

      // for(int i = 0; i < T.n; i++, xm+= 6)
      // {
        // StdDraw.setPenColor(StdDraw.WHITE);
        // StdDraw.filledCircle(xm,y, 3);
        // StdDraw.setPenColor(StdDraw.BLACK);
        // StdDraw.circle(xm,y, 3);
        // StdDraw.text(xm,y,Integer.toString(T.item[i]));
      // }
     // }
    }
```

```
public static double printLinear(BTreeNode T, double y, double totalXchange, int level)
{
   double change = totalXchange/printNodeDepth(T,level);
   double xm = 0;
   for(int i =0;i < T.n+1;i++,xm+=6)
   {
     StdDraw.setPenColor(StdDraw.WHITE);
     StdDraw.filledCircle(xm,y, 3);
     StdDraw.setPenColor(StdDraw.BLACK);
     StdDraw.circle(xm,y, 3);
     StdDraw.text(xm,y,Integer.toString(T.item[i]));
  }
  return xm;
}

 public static void printAscendingtoArray(BTreeNode T)//Problem A
{
   //Prints all items in the tree in ascending order
  if (T.isLeaf)
      toArray(T);
  else
  {
    for(int i =0; i<T.n;i++)
    {
      printAscendingtoArray(T.c[i]);
      toArray(T.item[i]);
    }
     printAscendingtoArray(T.c[T.n]);
  }
}
```

```
public static void toArray(BTreeNode T)//Problem A
{
   for(int additionIndex = 0; additionIndex < T.n; additionIndex++)
   {
     num[index] = T.item[additionIndex];
     index = index +1;
   }
}

public static void toArray(int number)//Addition to Problem A
{
  num[index] = number;
  index = index +1;
}

public static int countItems(BTreeNode root)//Addition to Problem A
{
  if(root.isLeaf)
      return root.n;
  else // root not leaf
  {
    int sum = root.n;
    for(int i = 0; i < root.n+1; i++)
      sum += countItems(root.c[i]);
    return sum;
  }
}

public static void printSmallest(BTreeNode root, int level)//Problem B
{
  if(level == 0)
      System.out.println(root.item[0]);
  else if(root.isLeaf && level != 0)
      return;
  else
  {   //System.out.println(level);
      printSmallest(root.c[0],level-1);
  }
}

public static void printLargest(BTreeNode root, int level) //Problem C
{
  if(level == 0 && root != null)
      System.out.println(root.item[root.n-1]);
  else if (root.isLeaf && level != 0)
      return;
  else //root is not a leaf and level != 0
      printLargest(root.c[root.n], level-1);
}

public static int printNodeDepth(BTreeNode root, int level)//Problem D
{
   if(level == 0 && root != null)
  {
    return 1;
  }

  else if(root != null && root.isLeaf && level !=0)
  {
    return 0;
  }
  else //root is not a leaf and level is not 0
  {
    int sum = 0;
    for(int i = 0; i < root.n+1; i++)
      sum += printNodeDepth(root.c[i], level-1);
    return sum;
  }
}

public static void printDepth(BTreeNode root, int level)//Problem E
{
  if(level == 0 && root != null)
  {
    for(int i = 0; i < root.n; i++)
      System.out.print(root.item[i] +" ");
  }

  else if(root.isLeaf && level !=0)
  {
    return;
  }
  else //root is not a leaf and level is not 0
  {
    for(int i = 0; i < root.n+1; i++)
      printDepth(root.c[i], level-1);
  }
}

public static int nodesFull(BTreeNode root)//Problem F
{
  if(root.n == 5 && root != null && !(root.isLeaf))
  {
    int sum = 1;
    for(int i = 0; i < root.n+1; i++)
      sum += nodesFull(root.c[i]);
    return sum;
  }
  else if(root.n != 5 && root != null && !(root.isLeaf))
  {
    int sum = 0;
    for(int i = 0; i < root.n+1; i++)
      sum += nodesFull(root.c[i]);
    return sum;
  }
  else if(root.isLeaf && root.n == 5)
    return 1;
  else //if(root.isLeaf && root.n != 5)
    return 0;
}

public static int leavesFull(BTreeNode root)
{
  if(root != null && root.isLeaf && root.n == 5)
  {
    return 1;
  }

  else if(root != null && root.isLeaf && root.n != 5)
  {
    return 0;
  }
  else //root is not a leaf and level is not 0
  {
    int sum = 0;
    for(int i = 0; i < root.n+1; i++)
      sum += leavesFull(root.c[i]);
    return sum;
  }
}

public static int searchLevel(BTreeNode root, int k) //Problem H
{
  int i = 0;
  while(i < root.n && k > root.item[i])
      i++;
  if(i == root.n || k < root.item[i])
      if(root.isLeaf)
          return -1;
      else
      {
                      (Addition to the code made after submission)
          int sum = searchLevel(root.c[i], k);
          if(sum != -1)
           return 1 + sum;
          else
           return -1;
      }

  else
      return 0;
}

public static void SearchBTree(BTreeNode root, int k) //Problem I
```

```
 {
   int i = 0;
   while(i < root.n && k > root.item[i])
      i++;
   if(i == root.n || k < root.item[i])
      if(root.isLeaf)
         return;
      else
          SearchBTree(root.c[i], k);
   else
      for(int j = 0; j < root.n; j++)
         System.out.print(root.item[j]+" ");
 }
}
```

V. Conclusion

This lab made me realize about the weaknesses I have in terms of being able to perform certain operations with data structures in the form of trees. In terms of what knowledge I acquired from the design of activities “a)” - “i)” I believe that I was able to show that the operations to determine certain aspects of the tree are very simple, interchangeable as required. For example, the search algorithm which is a standard for B-trees can be redesigned so it can find how many full nodes and full leaf nodes are present in the structure.

Furthermore, I think that by not being able to finish the graphic display of the tree I was at least able to think about how to solve other problems, for example finding the maximum and the minimum. In general I believe that lab 4 was an instrumental learning experience in my path to become computer scientists.

It is my belief that this lab taught me several things about my skills as a computer scientist, even though I was successful in producing an answer for all but one exercise I felt accomplished of being able to be able to navigate all the tree/data structure in search for very specific qualities of each node.

Statement of Academic Honesty:

"I certify that this project is entirely my own work. I
wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class."

OGalindoMolina