Answer Pages

Question 21 (`pushAll`) answer:

```
~~template <class T>~~
void Tree::KStep::pushAll(TreeNode *n)
{   TreeNode * curr = n;
    while ( curr != NULL )
    {
        st.push(curr);
        curr = curr -> left;

    }
}
```

Question 22 (`KStep`) answer:

```
Tree::KStep::KStep ( )
{
    pushAll( root );
}
```

Question 23 (`hasMore`) answer:

```
bool Tree::KStep::hasMore ( )
{
    ~~ife~~
    return (!st . ~~isEpt~~ isEmpty ( ));
}
```

Question 24 (`step1`) answer:

```
int Tree :: KStep :: step1( )
{
    ~~if (st -> right != NULL)~~
        ~~TreeNode * temp = st -> right;~~
    ~~TreeN~~
    TreeNode *temp = st.pop( );          ... O(1)
    ~~st.push(~~
    if ( temp -> right != NULL )
        st.push ( temp -> right );       ... O(1)
    return temp -> data;
}.
```

Question 25 (step~~1 running time~~ (k)) answer:

```
int Tree::KStep::step(int k)
{
  if (st.isEmpty())
  {
    return;
  }
  int temp = step1();
  if (k == 1) return temp;
  else
    step(k-1);
  return temp;
}
```

Question 26 answer:

Lower Bound	O(1)
Average	O(1)
Upper Bound Case	O(1)

Question 27 (buildPerfectTree) answer:

```
~~Quadtree * Quadtree~~
QuadtreeNode * Quadtree::buildPerfectTree( int k ,
                                             RGBAPixel p )
{   QuadtreeNode * temp ;
        k == 0
    if ( ~~nwChild == NULL~~ )
    temp->nwChild->element = temp->neChild->element = temp->swChild->element
                                   = temp->seChild->element = P;

    else
    {
    temp->nwChild = buildPerfectTree ( k-1 , p );
    temp->neChild = buildPerfectTree ( k-1 , p );
    temp->swChild = buildPerfectTree ( k-1 , p );
    temp->seChild = buildPerfectTree ( k-1 , p );
        ~~this->el~~
    temp->element = p ;
    }  return temp;
}
```

Question 28 (perfectify) answer:

```
void Quadtree::perfectify(int levels )
{   QuadtreeNode * curr = root;
    if (levels == 0 )
        return ;

    else
    {                          NULL
        if ( root->nwChild == 0 )
        {
            curr->nwChild = buildPerfectTree ( curr->element , levels-1);
            curr->neChild = buildPerfectTree ( curr->element, levels-1 );
            curr->swChild = buildPerfectTree ( curr->element , levels-1);
            curr->seChild = buildPerfectTree ( curr->element , levels-1 );
        }
        Quadtree nw *( curr->nwChild );     Quadtree se ( curr->seChild );
        Quadtree ne ( curr->neChild );      nw . perfectify (levels -1 );
        Quadtree sw ( curr->swChild );      ne . perfectify (levels - 1 );
                                            sw . perfectify (levels -1 );
                                            se . perfectify (levels -1 );
    }
}
```

Question 29 (perfectify running time) answer:

~~Quad~~

O(log4 n)

Question 30 answer:

You may answer this question by filling in these blanks, or use the blank space for your own proof/disproof.

Preliminaries Let $H(n)$ denote the maximum height of an n-node SAVL tree, and let $N(h)$ denote the minimum number of nodes in an SAVL tree of height h. To prove (or disprove!) that $H(n) = \mathcal{O}(\log n)$, we attempt to argue that

$\frac{h}{3} \leq \log_2 n^3$

$$H(n) \leq 3\log_2 n, \text{ for all } n$$

Rather than prove this directly, we'll show equivalently that

a) $N(h) \geq$ $2^{h/3}$, (1pt)

Proof For an arbitrary value of h, the following recurrence holds for all SAVL Trees:

b) $N(h) =$ 1 + $N(h-1)$ + $N(h-3)$, (2pt)

c) and $N(0) =$ 1, $N(1) =$ 2, $N(2) =$ 3, (2pt)

We can simplify this expression to the following inequality, which is a function of $N(h-3)$:

d) $N(h) \geq$ 2 $\times$ $N(h-3)$, (1pt)

By an inductive hypothesis, which states:

e) for $n \leq h$, $N(n) \geq 2^{h/3}$, (1pt)

we now have

f) $N(h) \geq$ $2^{h/3}$ = part (a) answer, (1pt)

which is what we wanted to show.

Given that $2^0 = 1, 2^{1/3} \approx 1.25$, and $2^{2/3} \approx 1.58$, what is your conclusion?

Is an SAVL tree $\mathcal{O}(\log n)$ or not? (Circle one): (2pt)

YES NO

Overflow Page

Use this space if you need more room for your answers.

~~O(n)=~~

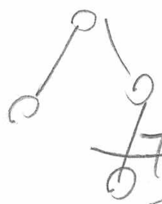
~~T(n) = 4T(n-1) + c~~

~~T(1) = d~~

~~$T(n) = 4^{(n-1)} T(1) + \sum_{i=0}^{n-1} c \cdot 4^i$~~

~~$= 4^{(n-1)} d + c\left[\frac{4^n - 4}{4-1}\right]$~~

~~$O(4^n)$~~



~~$T(n) = 4[T(\frac{n}{4})] + c$~~

~~T(1) = d~~

~~$4c \log_4 n + nd$~~

~~N(h)~~

~~= N(h-1) + N(h-3)~~

~~$T(n) = 4 \cdot T(\frac{n}{4})$~~

~~= N(h-2) + 2N(h-3)~~