Answer Pages

Question 21 (pushAll) answer:

```
void kStep::pushAll (TreeNode * n)
{
    if (n== NULL)
        return;
    st.push(n->data);
    pushAll (n-> left);
}.
```

Question 22 (KStep) answer:

```
kStep: kStep (){
    pushAll (root);
}.
```

Question 23 (hasMore) answer:

```
bool kstep::hasMore(){
    // if the stack is not empty,
it means there are elements left
    return (!st.isEmpty());
}
```

Question 24 (step1) answer:

```
int kStep::step1(){
    TreeNode * temp = st.pop();
  if (temp->right != NULL){
     pushAll(temp->right);
   }

}
```

Question 25 (`step1` running time) answer:

```
int KStep :: step (int k)
{   int returnval = step1 ();
    for (int i=1 ; i< k ; i++) // counting
    {   if (hasMore ()) // if the stack is not exhausted.
            step1 ();
    }
    return returnval;
}
```

Question 26 answer:

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

Question 27 (buildPerfectTree) answer:

```
QuadtreeNode * ~~Qud~~ Quadtree :: buildPerfectTree (int k,
RGBAPixel p )
{
    ~~Qud~~ QuadtreeNode * temp = new QuadtreeNode ();
    temp -> element = p;
    if (k==0) return temp; // only one node in this tree
    else {
        treebuilder (temp, k, p);
    }
    return temp; }

void Quadtree :: treebuilder (QuadTreeNode *& subroot, int k,
RGBAPixel p)   // helper function
{   if (k==0)  return;
    subroot -> nwChild = new QuadtreeNode ();
    subroot -> neChild = new QuadtreeNode ();
    subroot -> swChild = new QuadtreeNode ();
    subroot -> seChild = new QuadtreeNode ();
    ~~se~~ nwChild -> element = neChild -> element =
        swChild -> element = seChild -> element = p;
    Treebuilder (subroot -> nwChild, k-1, p);
    Treebuilder (subroot -> neChild, k-1, p);
    treebuilder (subroot -> swChild, k-1, p);
    treebuilder (subroot -> seChild, k-1, p);
} // make new nodes recursively
```

Question 28 (perfectify) answer:

```
void Quadtree :: perfectify (int h)
{
    ~~void~~ perfectHelper (root, 0, h);
}

void Quadtree :: perfectHelper
(QuadtreeNode *& subroot, int i, int h) // helper function
{
    if (i == h) return; // its level equals the level we need
    if (subroot -> nwChild == NULL) { // its level is less than requirement
        buildPerfectTree (h-i, subroot->element); // make it to be a perfect tree
        return;
    }
    else { perfecthelper (subroot -> nwChild, i+1, h);
        perfecthelper (subroot -> neChild, i+1, h);
        perfecthelper (subroot -> swChild, i+1, h);
        perfecthelper (subroot -> seChild, i+1, h); }
}
```

Question 29 (perfectify running time) answer:

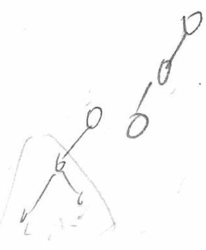
$O(4^h)$

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

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



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

a) $N(h) \geq$ $\sqrt[3]{2^h} = 2^{\frac{h}{3}}$, (1pt)

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

b) $N(h) =$ $N(h-1)$ + $N(h-3)$ + 1, (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) $N(h-3) \geq 2^{\frac{h-3}{3}}$, (1pt)

we now have

f) $N(h) \geq$ $2 \times 2^{\frac{h-3}{3}} = 2^{\frac{h}{3}}$ = 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.