Answer Pages

Question 21 (`pushAll`) answer:

```
Void KStep::pushAll(TreeNode *n)
    if(n != NULL) {
        St.push(n);
        pushAll(n->left);
    }
    return;
}
```

Question 22 (`KStep`) answer:

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

Question 23 (`hasMore`) answer:

```
bool KStep::hasMore() {
    return st.isEmpty();
}
```

Question 24 (`step1`) answer:

```
int KStep::step1() {
TreeNode temp = St.pop();
        return temp->data;
}
```

Question 25 (~~step1 running time~~ step(k)) answer:

```
int KStep::step(int k){
    int ret = step1();
    for(int i=0; i< k-1; i++){
        TreeNode temp = st.pop()
        if(!hasMore()){
            return ret;
        }
        pushAll(temp->right);
    }
    return ret;
}
```

Question 26 answer:

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

Question 27 (`buildPerfectTree`) answer:

```
QuadtreeNode * Quadtree::buildPerfectTree(int K, RGBAPixel p){
    if((K-1) == 0 || (K-1) < 0) {
        Quadtree Node * temp = new Quadtree Node();
        temp->element = p;
        temp->nwChild = NULL;
        temp->ne Child = NULL;
        temp->swChild = NULL;
        temp->se Child = NULL;
        return temp;
    }
    Quadtree Node * rt = new QuadtreeNode();
    rt->element = p;
    rt->nwChild = buildPerfectTree(K-1, p);
    rt->neChild = buildPerfectTree(K-1, p);
    rt->swChild = buildPerfectTree(K-1, p);
    rt->seChild = buildPerfectTree(K-1, p);
    return rt
}
```

Question 28 (`perfectify`) answer:

```
void Quadtree::perfectify(int levels) {
    perfector(levels, root);
}

void Quadtree::perfector(int levels, QuadtreeNode * ptr) {
    if(ptr == NULL){ return;}
    if(ptr->nwChild==NULL){ ptr->nwChild = buildPerfectTree(levels -1, ptr->element);}
    if(ptr->neChild==NULL){ptr->neChild = buildPerfectTree(levels -1, ptr->element);}
    if(ptr->seChild==NULL){ptr->seChild = buildPerfectTree(levels -1, ptr->element);}
    if(ptr->swChild==NULL){ptr->swChild = buildPerfectTree(levels-1, ptr->element);}
    perfector(levels -1, ptr->nwChild);
    perfector(levels -1, ptr->neChild);
    perfector(levels -1, ptr->swChild);
    perfector(levels -1, ptr->seChild);
    return;
}
```

Question 29 (`perfectify` running time) answer:

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

$\frac{H(n)}{3} \leq \log_2 n$

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

$2^{h/3} \leq h$

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

By an inductive hypothesis, which states:

e) for every $j \geq 0$, an savl tree of height h will have $3(h-3)$ nodes minimum., (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.