## Answer Pages

Question 21 (pushAll) answer:

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

}
```

Question 22 (KStep) answer:

```
KStep::KStep ( )  {

    pushAll (root);

}
```

Question 23 (`hasMore`) answer:

```
bool KStep::hasMore () {
    if ( st.isEmpty()) ~~{~~
        return ~~{~~ false;
    return true;
}
```

Question 24 (`step1`) answer:

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

Question 25 (`step1` running time) answer:

```
int KStep:: step (int k) {
    if (k==0) return 0;
    ~~int temp = step1();~~
    if ( st.isEmpty())
        return 0;
    int temp = step1 ();
    step ( k-1) ;
    return temp ;
}
```

Question 26 answer:

| | |
|---|---|
| Lower Bound | O(1) |
| Average | O(logn) |
| Upper Bound Case | O(n) |

Question 27 (`buildPerfectTree`) answer:

```
QuadtreeNode *   Quadtree :: buildPerfectTree (int k, RGBAPixel p){
        QuadtreeNode * ~~My~~ myNode = new QuadtreeNode();
            if ( k == 0 )
                myNode -> element = p ;
        myNode -> nwChild = buildPerfectTree (k-1, p );
        myNode -> neChild = buildPerfectTree (k-1 , p );
        myNode -> swChild = buildPerfectTree (k-1, p );
        myNode -> seChild = buildPerfectTree (k-1, p );

        return myNode ;

}
```

Question 28 (`perfectify`) answer:

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


        ~~buildPerfectTree (h~~ ,
   void Quadtree :: perfectify ( int h, QuadtreeNode *&n ) {
            if (n -> nwChild == NULL) {
               n = buildPerfectTree (h, n -> element );
               ~~return ;}~~
     return;   perfectify (h-1, n -> nwChild );
     }         perfectify (h-1, n -> neChild);
               perfectify (h-1, n -> swChild);
               perfectify (h-1, n -> seChild);
   }
```

Question 29 (`perfectify` running time) answer: $O(\log n)$

Question 30 answer:

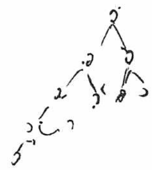
You may answer this question by filling in these blanks, or use the blank space for your own proof/disproof.

**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

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

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

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



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

b) $N(h) =$ $N(h-2)$ + $N(h-1)$ + $0$ , (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) $N(h) \geq$ $2^{\frac{h}{3}}$ = 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.