Data Structures

Homework #9 Solution

By Po-Chuan

Q1

Proof of time complexity

The target of this problem

- Prove that
$$f(n) = O(\log_a n) \leftrightarrow f(n) = O(\log_b n)$$

Please recall that…

- When we say $f(x) = O(\,g(x)\,)$, we mean that there is a positive constant M such that for all sufficiently large values of x, the absolute value of $f(x)$ is at most M multiplied by the absolute value of $g(x)$.
- It's an upper bound estimation.
- $|f(x)| \leq M|g(x)|\ \forall\ x \geq x_0$

The proof 1/2

- Proof of $f(n) = O(\log_a n) \rightarrow f(n) = O(\log_b n)$
- $|f(n)| \leq M|\log_a n| \; \forall \; n \geq n_0$
- We take $N = \frac{M}{\log_b a}$
- $|f(n)| \leq M|\log_a n| = N|\log_b n| \; \forall \; n \geq n_0$
- $f(n) = O(\log_b n)$
- $f(n) = O(\log_a n) \rightarrow f(n) = O(\log_b n)$ is proved.

The proof 2/2

- ◇ $f(n) = O(\log_a n) \leftarrow f(n) = O(\log_b n)$ can be proved in the same manner
- ◇ $f(n) = O(\log_a n) \leftrightarrow f(n) = O(\log_b n)$

Q2

Classify sorting algorithms
as stable or unstable

Stable algorithms

1. Insertion sort
2. Bubble sort
3. Merge sort

Unstable algorithms

1. Selection sort
2. Quick sort

- Generally, quick sort is unstable, but stable implementation of quick sort also exists

Why it's stable or not?

1. Selection sort – unstable

- Because selection sort swaps the minimum element by the first element after the sorted segment, which causes one element to go after its counterpart.

Why it's stable or not?

2. Bubble sort – stable

- Because bubble sort doesn't sort elements with the same value, the relative order of all elements with the same value is therefore reserved.

Q3

Prove that a strictly binary tree with n leafs has exactly 2n-1 nodes

The proof

- When n=1, the proposition holds.
- Suppose the proposition hold when n=k.
- When n=k+1, we have to insert 2 nodes under one of the leaf nodes of a strictly binary tree with n nodes. The tree is still a strictly binary tree since the only status-changed node has 2 children.
- By M.I., the proposition is true.

Q4

Level-order traversal implementation

Hint: using BFS

The pseudocode

```
Queue<node> bfs;
bfs.push( root );
while bfs is not empty
   print bfs.front
   for all nodes v under bfs.front
      bfs.push( v )
   bfs.pop
```

Q5

Preorder traversal of a general tree

The code…

```
template<typename T>
void preorder( GeneralTree<T>* root )
{
   cout << root->getItem() << endl;
   if ( root->getLeftChildPtr() != nullptr )
      preorder( root->getLeftChildPtr() );
   if ( root->getRightChildPtr() != nullptr )
      preorder( root->getRightChildPtr() );
}
```

The end~