Data Structures and Algorithms – MidTerm Exam

1. Let 'I' be "**push**", 'O' be "**pop**" and 'P' be "**print**" directly. To convert the infix expression `a+(b*c-d)/e` to its postfix through a stack, the sequence of operations are ___PIIPIPOIPOOIPOO___
(For example: `(a+b)` is converted to `ab+` by `IPIPOO`.) (5 points)

2. In a binary tree of N nodes, there are __N+1__ NULL pointers representing children. (2 points)

3. A sorting algorithm is ***stable*** if elements with equal keys are left in the same order as they occur in the input. Which of the following algorithms is/are stable? Answer: ___(a) (c)___ (8 points)
(a) insertion sort; (b) quick sort; (c) merge sort; (d) heap sort

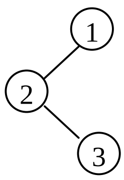
4. The following routine removes duplicates from an array-based list `A[0]` … `A[N-1]`. **LastPosition** is initially `N-1`.

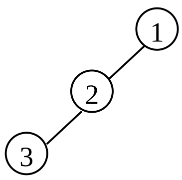
```
for ( i = 0; i < LastPosition; i ++ ) {
    j = i + 1;
    while ( j < LastPosition )
        if ( A[i] == A[j] )   Delete(j);
        else   j ++;
}
```

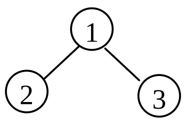
(a) What is the function of **Delete**? (3 points)
Delete A[j] by shifting A[j+1] … A[LastPosition-1] to the left.
LastPosition - - .

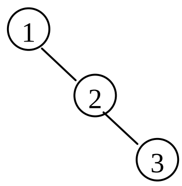
(b) $T_{worst}(N) =$ ___$O(N^2)$___. (2 points)
(c) Using linked list implementation, $T_{worst}(N) =$ ___$O(N^2)$___. (2 points)

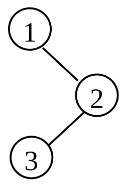
5. Among the given trees, __d__ has the same inorder and preorder traversal results, and __b__ has the same postorder and inorder traversal results. (4 points)

(a)


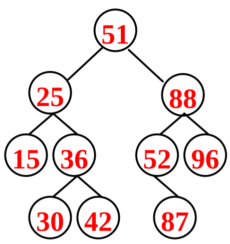
(b)


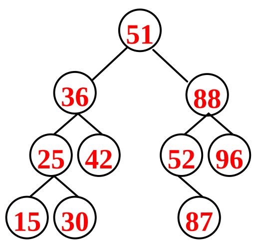
(c)


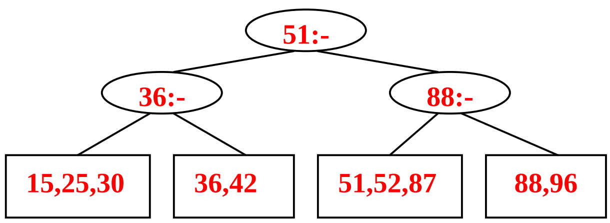
(d)


(e)


6. Show the result of inserting { 51, 25, 36, 88, 42, 52, 15, 96, 87, 30 } into
(a) an initially empty binary search tree; (b) an initially empty AVL tree; (c) an initially empty 2-3 tree.
(30 points)







7. Please fill in the blanks in the programs. (12 points)

(a) Insertion for separate chaining hash table:

```
void Insert( ElementType Key, HashTable H )
{   Position   Pos, NewCell;
    List L;
    Pos = Find( Key, H );
    if ( Pos == NULL ) {
        NewCell = malloc( sizeof( struct ListNode ) );
        L = H->TheLists[ Hash( Key, H->TableSize ) ];
        NewCell->Element = Key;

        NewCell->Next = L->Next                    ;

        L->Next = NewCell                          ;
    }
}
```

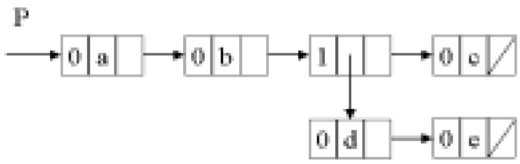
(b) Percolate down a max heap

```
void PercolateDown( int p, PriorityQueue H )
{   int   child;
    ElementType   Tmp = H->Elements[ p ];
    for ( ; p * 2 <= H->Size; p = child ) {
        child = p * 2;

        if ( child!=N-1&&H->Elements[child+1]>H->Elements [child] )
            child++;
        if ( H->Elements[ child ] > Tmp )

            H->Elements[ p ] = H->Elements[ child ];
        else    break;
    }
    H->Elements[ p ] = Tmp;
}
```

8. Assume that we represent trees using the list representation and that we define the node structure as:

TAG	DATA	LINK

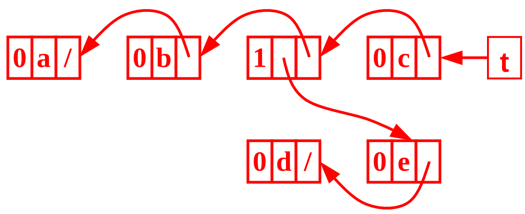
where LINK is a pointer pointing to the next element in the list; TAG is a field that holds the value of TRUE if the node is a link node in which DATA is a pointer pointing to the sublist, and a value of FALSE if the node is an atom node in which DATA is the data field. A sample tree is shown by the figure:



Please describe the function of the following program (7 points) and draw the resulting tree for the above example (5 points).

```
void   r ( GLNode_ptr   p,   GLNode_ptr   *t   )
{   GLNode_ptr temp1, temp2;
    if ( ! p ) { *t=NULL;   return; }
    if ( ! p->tag )
    {   temp1 = p; p = p->link; temp1->link = NULL; }
    else
    {   r ( p->data.sublist, t );
        temp1 = p; p = p->link; temp1->link = NULL; temp1->data.sublist = *t; }
    if ( p )
    {   r ( p, t );
        temp2 = *t;
        while( temp2->link )   temp2 = temp2->link;
        temp2->link = temp1;
    }
    else   *t = temp1;
}
```

Reverse p, and t is the new head pointer.



9. Please write a C program to obtain the *k*th largest integer **without destroying** the original integer list. Your algorithm must have an average run time no worse than O(N log N). (20 points)

```
int   Find_kth ( int A[ ], int N, int k )
/* A[ ] stores the integer list; N is the size of the list;          */
/* and you are supposed to return the kth largest integer. */
```

Algorithm:
Define a table[ ] and make table sort; /* (+ quicksort or mergesort or heapsort) */
/* Note: the list must be sorted in decreasing order */
Return A[table[k-1]].
/* Or if the list is sorted in increasing order */
Return A[table[N-k]].

A sample program – quicksort + table:

Assume that Swap, Cutoff, and MAX_SIZE are pre-defined.

```
int   Median3_with_table( int A[ ], int table[ ], int Left, int Right )
{   int   Center = ( Left + Right ) / 2;

    if ( A[ table[ Left ] ] > A[ table[ Center ] ] )
          Swap( &table[ Left ], & table[ Center ] );
    if ( A[ table[ Left ] ] > A[ table[ Right ] ] )
          Swap( & table[ Left ], & table[ Right ] );
    if ( A[ table[ Center ] ] > A[ table[ Right ] ] )
          Swap( & table[ Center ], & table[ Right ] );

    /* Invariant: A[ table[ Left ] ] <= A[ table[ Center ] ] <= A[ table[ Right ] ] */

    Swap( & table[ Center ], & table[ Right - 1 ] );   /* Hide pivot */
    return A[ table[ Right - 1 ] ];   /* Return pivot */
}

void   Qsort_with_table( int list[ ], int table[ ], int Left, int Right)
{   int   i, j;
    int   Pivot;

    if ( Left + Cutoff <= Right ) {
          Pivot = Median3_with_table( A, table, Left, Right );
          i = Left; j = Right - 1;
          for( ; ; ) {
               while( A[ table[ ++i ] ] < Pivot ) { }
               while( A[ table[ - -j ] ] > Pivot ) { }
               if ( i < j )
                     Swap( &table[ i ], &table[ j ] );
               else
                     break;
          }
          Swap( &table[ i ], &table[ Right - 1 ] );   /* Restore pivot */
          Qsort( A, table, Left, i - 1 );
          Qsort( A, table, i + 1, Right );
    }
    else   /* Do an insertion sort on the subarray */
          InsertionSort( A + Left, table + Left, Right - Left + 1 );
}
```

```
void InsertionSort( int A[ ], int table[ ], int N )
{
      int   j, P;
      int   Tmp;

      for ( P = 1; P < N; P++ ) {
            Tmp = A[ table[ P ] ];
            for ( j = P; j > 0 && A[ table[ j - 1 ] ] > Tmp; j-- )
                  A[ table[ j ] ] = A[ table[ j - 1 ] ];
            A[ table[ j ] ] = Tmp;
      }
}

int   Find_kth ( int A[ ], int N, int k )
{   int   i, table[ MAX_SIZE ];
    for ( i = 0; i < N; i++ )
       table[ i ] = i;   /* initialize table */
    Qsort_with_table( A, table, 0, N-1);
    return   A[ table[ N-k ] ];
}
```