

## Data Structures and Algorithms

## Mid-term Exam, Sec. Z, Solutions

1. 3 pts. Assume that function  $f = 5n^2 + 20n^2 \log n + 15\sqrt{n^5}$  represents the run-time of a program  $P$ .
- (a) Consider the 3 terms in  $f$  and state which is the most important term, the second most important term, the least important term:  
 (1)  $15\sqrt{n^5}$ , (2)  $20n^2 \log n$ , (3)  $5n^2$ .
- (b) Indicate using the big  $O$  notation the time complexity of  $P$ :  
 $O(\sqrt{n^5})$
2. 4 pts. Consider the following function in which only the loops are indicated. All other statements in the function need time  $O(1)$ .

```
void EX(int n)
{
  int i = 2*n;
  while (i >= 1){
    ...
    for (int j= 3; j <= n-2; j+=2) {
      ...
    }
    i = i/ 3;
  }
}
```

Determine the run-time complexity of EX as a function of  $n$ , using the big  $O$  notation:

The while loop is repeated  $\log_3 2n$  times and the for loop is repeated  $(n - 4)/2 + 1$  times. Loops are nested, thus the run-time complexity is  $O(n \log n)$ .

3. 5 pts.
- (a) Give an informal definition of a queue:  
 It is a restricted *list* in which any item is inserted at one end of the list and any deletion is done at the other end of the list.
- (b) Assume we have a circular array implementation of a queue. Draw the picture of the queue and the position of *front* and *rear* indices resulting from the following sequence of operations (assume we have a queue of integers):

```
Queue v(6);
v.enqueue(4);
v.enqueue(2);
int i=v.dequeue();
v.enqueue(3);
v.enqueue(i);
```

queue is array[0]...array[6].  
queue contains 2 in location 2, 3 in location 3, 4 in location 4.  
front=1, rear = 4.

4. 10 pts. Assume that we have a class *list*

```
class list {      // a linked list
private:
  link* head;     // pointer to the list header node
  link* tail;     // pointer to the tail
  link* curr;     // pointer to the current element
public:
  list(const int = LIST_SIZE); //constructor
  ~list();        // destructor
                  //here we have the usual operations
                  //on lists
};
```

This list is implemented using a singly linked list, so the class link is defined as

```
class link {
public:
  ELEM element;  // ELEM value for this node
  link* next;    // pointer to the next node
  link(const ELEM& elemval, link* nextp=NULL); //constructor
          { element = elemval; next = nextp}
  ~link() { };   // destructor
};
```

We want to add to the class *list* a function

```
void remEvery(const ELEM & item);
```

which removes, in the linked list pointed to by *head*, every node containing *item*.

```
void remEvery(const ELEM & item){
  if (head==tail) return;      // list is empty
  link * ltemp1 = head;        // start from the head node
  link * ltemp2 = ltemp1->next; // ltemp2 is the node following ltemp1
                               // in the list
  while (ltemp2 != NULL) {
    if (ltemp2->elem == item)   // delete ltemp2 node
      { ltemp1 ->next = ltemp2->next;
        if (ltemp2 == curr) curr = ltemp1; // node pointed to by
                                             // current pointer is deleted, adjust curr pointer
        delete(ltemp2);
      }
    else ltemp1 = ltemp2;      // shift ltemp1 to the next node in the list
    ltemp2 = ltemp1->next      // shift ltemp2 to the next node in the list
  }
  tail = ltemp1;              // adjust tail pointer
};
```

5. 5 pts.

(a) How many internal nodes are there in  $T$ ?

6

(b) Give the preorder and postorder traversals of the  $T$ .

preorder:  $a, b, d, e, g, c, f, h, i$

postorder:  $d, g, e, b, i, h, f, c, a$

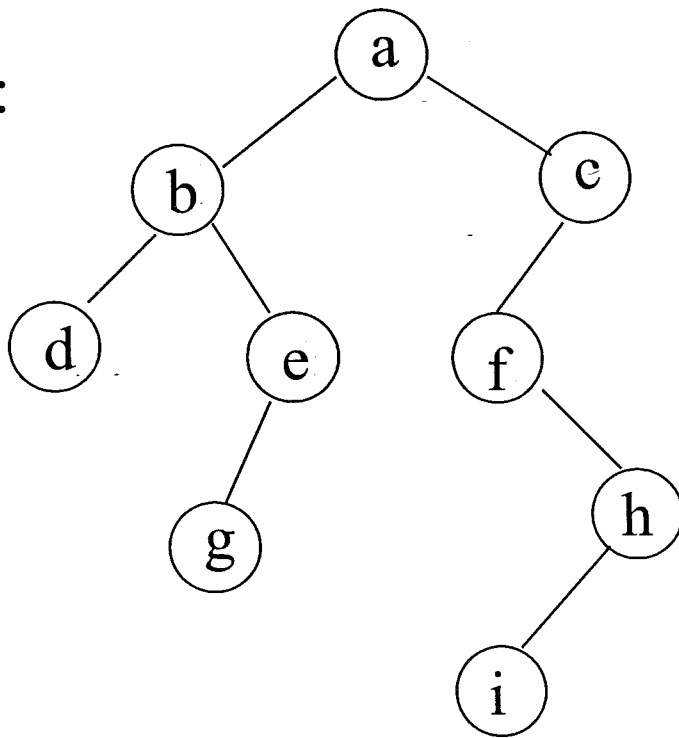
(c) At least how many nodes do you have to add to  $T$  to make it a full binary tree?

4

(d) At least how many nodes do you have to add to  $T$  to make it a complete binary tree?

17

T:



15/20

CONCORDIA UNIVERSITY  
COMP 352 : Data Structures and Algorithms  
Fall 2000  
Section: V

Midterm Examination

October 17<sup>th</sup>, 2000  
Total Marks: 20 (20% of the final grade)

Name: \_\_\_\_\_  
Student ID: \_\_\_\_\_

Question1 (3 marks)

0.5

What is the asymptotic time complexity in the average case for the following operations

- (a) Create an array-based stack  $\Theta(1)$
- (b) Create a linked stack  $\Theta(1)$
- (c) Find an element in a linked list  $\Theta(n)$
- (d) Insert an element into a linked queue (enqueue operation)  $\Theta(1)$
- (e) Insert an element into an array-based queue (enqueue operation)  ~~$\Theta(n)$~~   $\Theta(1)$
- (f) Clear an array-based list  $\Theta(1)$

**Question 2** (3 marks)

(a) For the following code fragment give

$T(n)$  and  $\Theta(n)$  in the best case  $T(n) = c_1$  ;  $\Theta(1)$

$T(n)$  and  $\Theta(n)$  in the average case  $T(n) = \frac{n(n+1)}{2}$  ;  $\Theta(n)$

$T(n)$  and  $\Theta(n)$  in the worst case  $T(n) = n$  ;  $\Theta(n)$

```

int search (int* array, int k) { // Find element k
    for (int i=1; i<n; i++) // For each element
        if (array[i] = k) // If found
            return i; // Return its position
    return NOT_FOUND; // Return const - flag
}
    
```

(b) For the following code fragment give

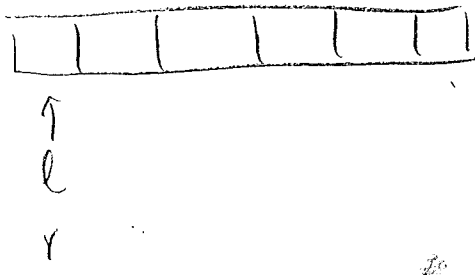
$T(n)$  and  $\Theta(n)$  in the best case  $T(n) = c_1$  ;  $\Theta(1)$  //  $c_1$  is constant 1

$T(n)$  and  $\Theta(n)$  in the average case  $T(n) = \log n$  ;  $\Theta(\log n)$

$T(n)$  and  $\Theta(n)$  in the worst case  ~~$T(n) = n/2$  ;  $\Theta(n)$~~  - 0.5

```

int binarySearch (int* array, int k, int left, int right) {
    // find element k
    int l = left + 1; //l and r beyond the bounds
    int r = right + 1;
    while (l + 1 != r){
        int i = (l + r)/2
        if ( k == array[i]) return i; // return its position
        else if ( k < array[i]) r = i; // in left half
        else ( k > array[i]) l = i; //in right half
    }
    return NOT_FOUND; // return const - flag
}
    
```



**Question 3**

(a) (1 mark) Convert the following infix expression into a postfix form

$$(5 + 1) * (3 - 2) / 3 - 4 * 2$$

$$\underline{51+32-3/42-}$$

(b) (1 mark) Evaluate the following postfix expression

$$52+6*93/-$$

$$\begin{array}{l} 549, 693/- \\ (5+2) * 6, 13/- \\ (5+2) * 6 - (9/3) \end{array}$$

$$\underline{(5+2) * 6 - (9/3) = 39}$$

**Question 4 (2 marks)**

Each data element is 16 bytes. Size of a pointer is 8 bytes. Maximum number of elements is 100. Consider a queue that contains  $n$  elements. Calculate break-even point beyond which the array based implementation is more space efficient.

$$P = 8 \text{ bytes}$$

$$E = 16$$

$$D = 16 \text{ bytes } 100$$

$$\begin{array}{l} \text{Array} = DE \\ \text{LIST} = n(P+E) \end{array} > n > \frac{DE}{P+E} \Rightarrow n > \frac{1600}{24} \Rightarrow n > 14.815$$

$$\text{DE} = -0.5$$

Thus, by the break-even analysis, it requires at least 15 elements ( $n \geq 15$ ) in order for the array-based implementation to be more efficient.

or  $\underline{n \geq 15}$

**Question 5 (3 marks)**

Fill in the code to remove an element from the top of the stack, and return it to the calling routine, in a linked stack implementation with the following definitions:

```
class Node { // Node class
public:
    char aValue;
    Node* next;
};

class Stack { // Linked stack class
private:
    Node* topOfStack; // Pointer to top element
public:
    Stack() {topOfStack = NULL;} // Constructor
    Node* Remove ();
    ...
    ...
};
```

```
Node* Stack::Remove()
{
    Node* temp = topOfStack; // make "temp" point to same
    topOfStack = topOfStack->next; // memory location as "topOfStack"
    // make topOfStack point to
    // next node in the list
    return temp; // return pointer "temp" since
    // the return argument is
    // expecting a pointer
    Empty Stack.
}
```

**Question 6 (7 marks)**

Using the public functions for the class `List` given below, write a C++ function,

```
List Alternate (List*& L1, List*& L2)
```

that creates a list from elements in L1 and L2 by alternating elements in the two lists and then appending the remaining nodes of the longer of two lists.

For example, if L1 = (10,20, 30) and L2 = (15, 25, 35, 45, 55), the call to `Alternate(L1, L2)` produces the list (10, 15, 20, 25, 30, 35, 45, 55), and L1 and L2 become empty lists. Assume that list elements are of type `Elem`.

```
class List { // Linked list class
private:
    Link* head;           // Pointer to list header
    Link* tail;          // Pointer to last Elem
    Link* curr;          // Pos of "current" Elem
public:
    List();              // Constructor
    ~List();            // Destructor
    void clear();        // Remove all Elems
    void insert(const Elem); // Insert at current pos
    void append(const Elem); // Insert at tail
    Elem remove();      // Remove/return Elem
    void setFirst();    // Set curr to first pos
    void prev();        // Move curr to prev pos
    void next();        // Move curr to next pos
    int length() const; // Return length
    void setPos(int);   // Set current pos
    void setValue(const Elem); // Set current value
    Elem currValue() const; // Return current value
    bool isEmpty() const; // TRUE if list is empty
    bool isInList() const; // TRUE if now in list
    bool find(Elem);    // Find value
};
```

```
while (L2.isInList())
```

```
{
```

```
alter.insert(L1.currValue());
```

```
alter.insert(L2.currValue());
```

```
L1.remove(); // remove current pointer of L1
```

```
L2.remove(); // " " " " " L2
```

```
L1.next(); // go to next node in L1
```

```
L2.next(); // " " " " " L2
```

```
} //end of while
```

```
while (L1.isInList())
```

```
{
```

```
alter.append(L1.currValue());
```

```
L1.remove();
```

```
L1.next();
```

```
} //end of while
```

```
} //end of else
```

```
} //end of Alternate
```

void Alternate (List \* & L1, List \* & L2) // given

```
List alter ; // create object of type "List" to make new list  
L1->setFirst(); // set pointer to first node in L1  
L2->setFirst(); // " " " " " " " " L2
```

if (L1->length() <= L2->length())

while (L1->isInList())

```
alter.insert(L1->currValue());  
alter.insert(L2->currValue());
```

```
L1->remove(); // remove current pointer in L1  
L2->remove(); // " " " " " " L2
```

```
L1->next(); // go to next node
```

```
L2->next(); // go to " node
```

```
} // end of while
```

```
while (L2->isInList())
```

```
alter.append(L2->currValue());
```

```
L2->remove();
```

```
L2->next();
```

```
} // end of while
```

```
else // end of if
```

```
if
```

```
// see next page
```

-2,5

CONCORDIA UNIVERSITY  
COMP 352 : Data Structures and Algorithms  
Summer 2001

**Midterm Examination**

May 30<sup>th</sup>, 2000

Total Marks: 50 (25% of the final grade)

Name: \_\_\_\_\_

Student ID: \_\_\_\_\_

**Question 1** (9 marks)

Give the asymptotic time complexity in the average case for the following operations:

- (a) Clear a linked list \_\_\_\_\_
- (b) Delete an element in an array based list \_\_\_\_\_
- (c) Find an element in a linked list \_\_\_\_\_
- (d) Create a linked stack \_\_\_\_\_
- (e) Insert an element into a linked stack (push operation) \_\_\_\_\_
- (f) Insert an element into an array-based stack (push operation) \_\_\_\_\_
- (g) Create a linked queue \_\_\_\_\_
- (h) Insert an element into a linked queue (enqueue operation) \_\_\_\_\_
- (i) Insert an element into an array-based queue (enqueue operation) \_\_\_\_\_

**Question 2** (8 marks)

Give  $T(n)$  and  $\Theta(n)$  for the following code fragments

```
(a) sum = 0;
    for (j=1; j<=n; j++)
        for (i=1; i<=n; i++)
            sum++;
    for (k=0; k<n; k++)
        A[k] = k;
```

```
(b) sum2 = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<=i; j++)
            sum2++;
```



```
(c) sum1 = 0;
    for (k=1; k<=n; k*=2)
        for (j=1; j<=k; j++)
            sum1++;
```



```
(e) sum = 0;
    k = 5;
    for (i=0; i<k; i++)
        for (j=0; j<n; j++)
            sum++;
```



$1 + 2 + 4 + 8 = 15$

$5n$

$2n - 1$

**Question 3** (3 marks)

Each data element is 16 bytes. Size of a pointer is 8 bytes. Maximum number of elements is 100.

- (a) Determine the space required for a linked list implementation (Number of elements in the list is  $n$ )

---

- (b) Determine the space required for an array based implementation (Number of elements in the list is  $n$ )

---

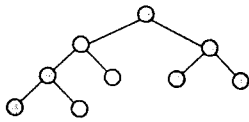
- (c) Calculate break-even point (Break-even point is the number of elements at which both the linked list and array-based list implementations are of equal space efficiency.)

---

**Question 4** (5 marks)

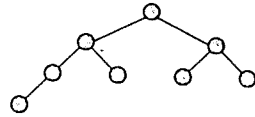
For each of the following trees please state whether it is full, complete, neither, or both

(a)



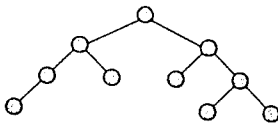

---

(d)



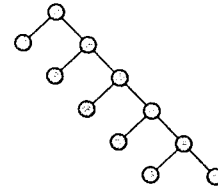

---

(b)



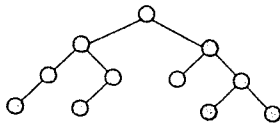

---

(e)




---

(c)




---

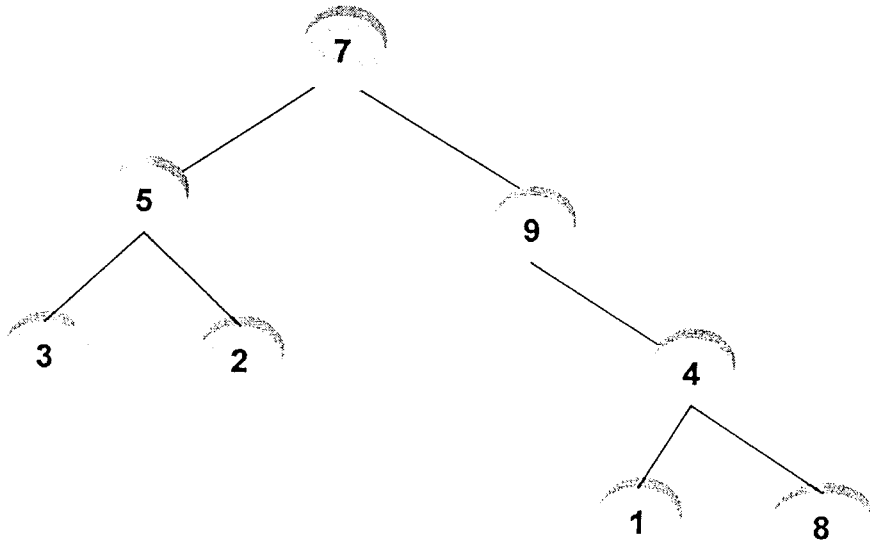
**Question 5** (4 marks)

(a) Traverse the tree below using inorder traversal (Write the numbers corresponding to the nodes).

---

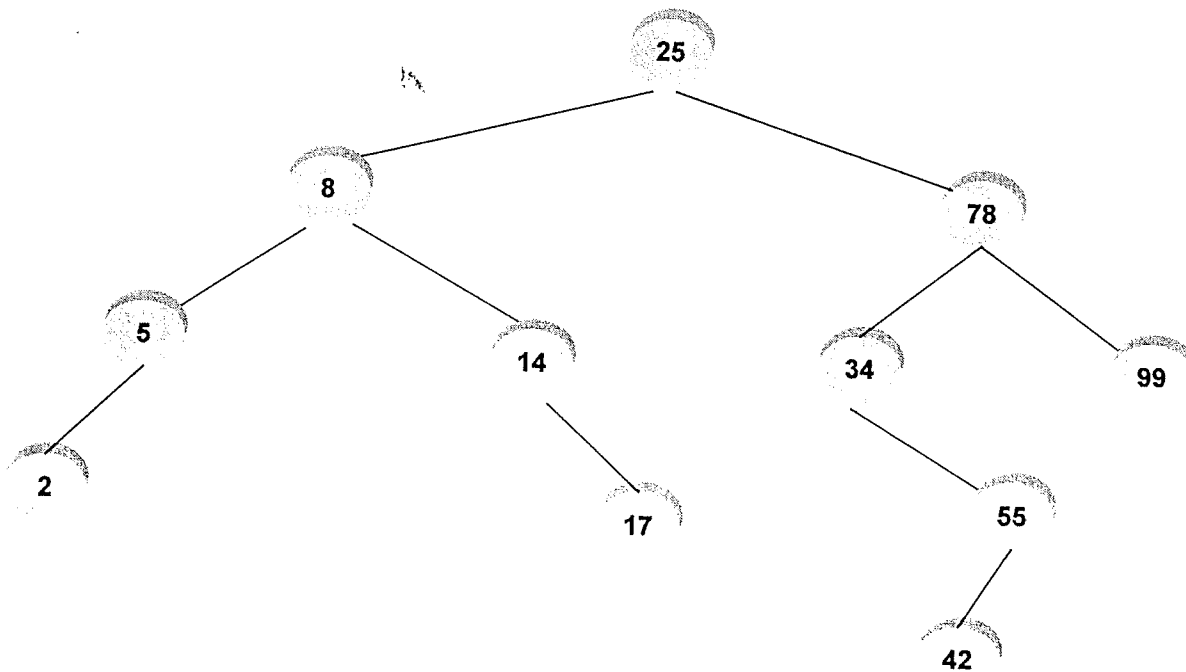
(b) Traverse the tree below using preorder traversal (Write the numbers corresponding to the nodes).

---



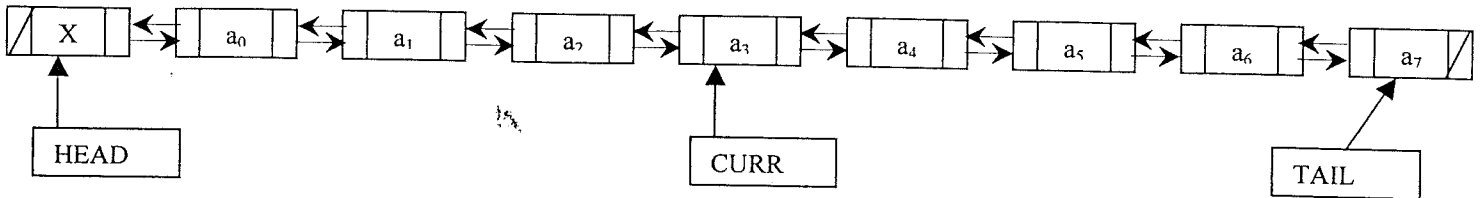
**Question 6** (5 marks)

Draw the BST that results from deleting value 25 from the BST below

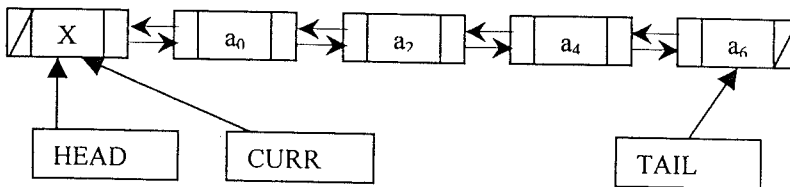


**Question 7** (16 marks)

Given a doubly linked list A



the *trim* operation applied to list A gives



that is, all elements  $a_i$  with odd subscript  $i$  are removed.

Add to the doubly linked list class implementation a member function

```
void trim();
```

which trims the list. To keep this question simple, `trim()` should just reset the `curr` pointer to the head of the list.

**Try to make your function as efficient as possible.**

You should assume the doubly – linked list node and doubly – linked list classes as given in your textbook. For your reference, the declarations for the list node class and for the linked list class are reproduced below:

```
class Link { // Doubly - linked node
public:
    Elem element; // Elem value for node
    Link *next; // Pointer to next node
    Link* prev; // Pointer to prev node
    Link(const Elem elemval, Link* nextval =NULL,
         Link* prevp =NULL)
    { element = elemval; next = nextval; prev = prevp;}
    Link(Link* nextval =NULL, Link* prevp = NULL)
    { next = nextval; prev = prevp;}
};

class List { // Linked list class
private:
    Link* head; // Pointer to list header
    Link* tail; // Pointer to last Elem
    Link* curr; // Pos of "current" Elem
public:
    List(); // Constructor
```

```

~List();
void clear();
void insert(const Elem);
void append(const Elem);
Elem remove();
void setFirst();
void prev();
void next();
int length() const;
void setPos(int);
void setValue(const Elem);
Elem currValue() const;
bool isEmpty() const;
bool isInList() const;
bool find(Elem);
};

// Destructor
// Remove all Elems
// Insert at current pos
// Insert at tail
// Remove/return Elem
// Set curr to first pos
// Move curr to prev pos
// Move curr to next pos
// Return length
// Set current pos
// Set current value
// Return current value
// TRUE if list is empty
// TRUE if now in list
// Find value

```