CSE 4190.204 Data Structures
Exam #1
Monday, October 2, 2006

Name: _____

Student ID: _____

1. [6pt] Explain what are the data structures and why they are important in computer science.

Answer:

A data structure is a way of storing data in a computer memory so that it can be used efficiently.

In computer science, different kinds of DSs are suited to different kinds of applications.

In the design of many types of programs, the choice of appropriate DSs is crucial.

Therefore, a carefully chosen DS can lead to a more efficient algorithm.

2. [12pt] Explain each of the following concepts in Java briefly.

(a) Packages

(b) Intefaces

(c) Inheritance

(d) Exceptions

Answer:

(a) A package is an entity that groups classes together.

Packages organize and group classes, help to specify access restrictions to variables and methods, and make it easy to import other programmers' classes into your code.

(b) A Java interface is a list of zero or more static final data members and a list of zero or more method headers.

No implementation is provided and no instance is created. A class can *implement* one or more interfaces.

(c) Inheritance is an object-oriented concept that provides for the reuse and modification of an existing type.

We say that the new class inherits the members of its superclass.

We may regard the *extends* clause as implementing the "IsA" relationship.

(d) Exception means 'exceptional condition', an occurrence that alters the normal program flow.

It can be caused by a variety of happenings such as HW failures, resource exhaustion, and good old bugs.

When an exceptional event occurs, an exception is said to be *thrown* and the code that is responsible for doing something about the error is called an *exception handler*.

The exception handler *catches* the exception.

3. [10pt] The following method, *max*, finds the position of the largest element in a[0:n]. Determine the worst-case step count for *max*. You should set up a frequency table using profiling.

```
public static int max(Comparable [] a, int n)
{
  if (n < 0)
    throw new IllegalArgumentException
    ("MyMath.max:Cannot find max of zero elements");

  int positionOfCurrentMax = 0;
  for (int i = 1; i <= n; i++)
    if(a[positionOfCurrentMax].compareTo(a[i])<0)
     positionOfCurrentMax = i;
  return positionOfCurrentMax;
}
```

Answer:

The worst-case step count for n >= 0 is as below.

Statement	s/e	Frequency	Total Steps
public static int max(...)	0	0	0
{	0	0	0
if (n < 0)	1	1	1
throw ...	1	0	0
("MyMath.max:...");			
int pos = 0;	1	1	1
for (int i = 1; i <= n; i++)	1	n+1	n+1
if (a[pos] < a[i])	1	n	n
pos = i;	1	n	n
return pos;	1	1	1
}	0	0	0
Total			3n + 4

4. [12pt] Show that the following equalities are correct, using the definitions of O, Θ, Ω and o only. (Do not use the theorems in your textbook)

(a) $n! = O(n^n)$

(b) $n^{2n} + 6 * 2^n = \Theta(n^{2n})$

Answer:

(a) $n! = n(n-1)(n-2) \ldots 1 < n*n*n* \ldots *n = n^n$, $n > 1$.

So, $n! = O(n^n)$.

(b) $f(n) = n^{2n} + 6 * 2^n < n^{2n} + 6*n^{2n} = 7n^{2n}$ for $n > 1$.

So, $f(n) = O(n^{2n})$.

Also, $f(n) > n^{2n}$ for $n > 0$.

So, f(n) = Omega(n^{2n}).

Hence, f(n) = Theta(n^{2n}).

5. [10pt] The following method, *factorial*, gives a Java recursive method to compute n!. Determine the asymptotic time complexity of *factorial*.

```
public static int factorial(int n)
{
  if (n <= 1)
    return 1;
  else
    return n * factorial(n - 1);
}
```

Answer:

Statement	s/e	Frequency	Total steps
public static int factorial(int n)	0	0	Theta(0)
{	0	0	Theta(0)
if (n <= 1) return 1;	1	1	Theta(1)
else return n * factorial(n - 1);	1 + $t_{factorial}$(n-1)	1	1 + $t_{factorial}$(n-1)
}	0	0	Theta(0)

So, $t_{factorial}(n)$ = c for n <= 1 and 1 + $t_{factorial}(n-1)$ for n > 1 (here c is a constant).

Using repeated substitution, we get:

$t_{factorial}(n)$ = 1 + $t_{factorial}(n-1)$

= 2 + $t_{factorial}(n-2)$

= 3 + $t_{factorial}(n-3)$

= ...

= n - 1 + $t_{factorial}(1)$

= n - 1 + c

= Theta(n)

6. [15pt] The following program gives the class *DoublyLinkedList*. Fill in the *remove* method.

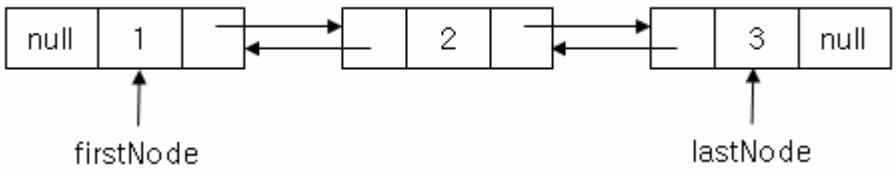


그림 1. Example doubly linked list

```
public class DoubleNode{
    DoubleNode prev;
```

```
    Object element;
    DoubleNode next;
}
public class DoublyLinkedList{
    DoubleNode firstNode;
    DoubleNode lastNode;
    int size;
    public void remove(int index){
```

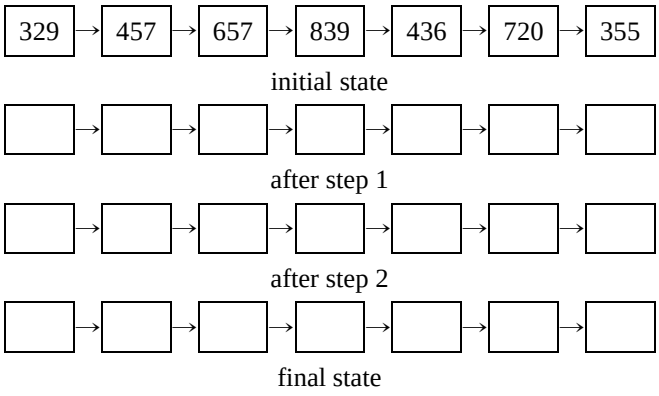
```
// Remove index-th node from the list. (0≤index≤size-1)
```

```
    }
    …
}
```

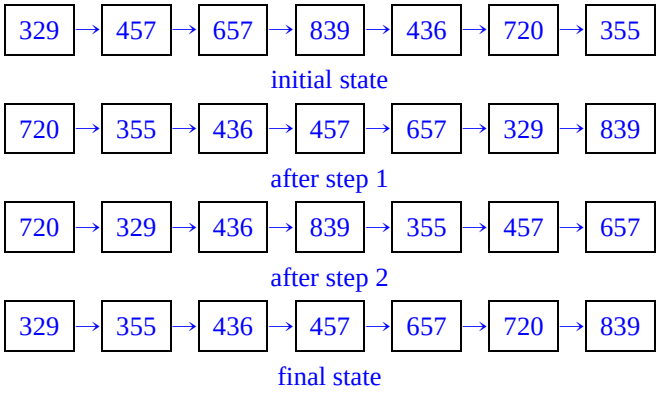
Answer:

```
public void remove(int index){
    if(index < 0 || index > size – 1) return;
    DoubleNode x = firstNode;
    for(int i = 0; i < index; i++) x = x.next;
    if(index > 0) x.prev.next = x.next; else firstNode = x.next;
    if(index < size - 1) x.next.prev = x.prev; else lastNode = x.prev;
    size--;
}
```

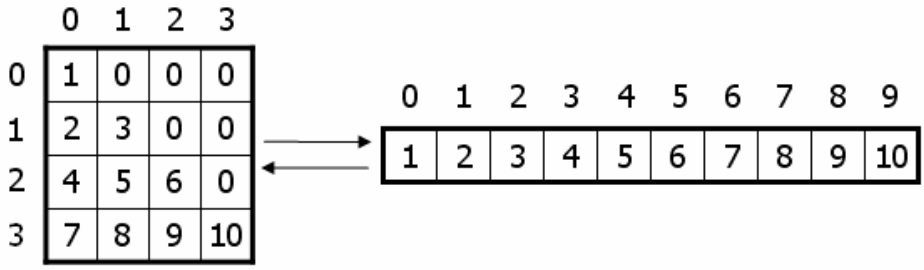
7. [10pt] The following figure shows the steps of radix sort (r=10, ascending order). Fill in the blanks.



Answer:



8. [10pt] The following figure shows the conversion between LTM(Lower Triangular Matrix) and 1D(one-dimensional) array.



Let's say that *LTMtoLinear* method returns an index of 1D array corresponding to input arguments representing a position of LTM. For example, LTMtoLinear(2, 1) = 4.

Fill in the method.

```
public int LTMtoLinear(int r, int c){
```

```
int idx;
```

```
        return idx;
```

}

Answer:

```
public int LTMtoLinear(int r, int c){
    int idx;
    idx = r * (r + 1) / 2 + c;
    return idx;
}
```

9. [15pt] Describe how to construct a simulated chain (i.e., a singly linked list of nodes that uses integer pointers; the last node's pointer is -1) c that consists of nodes 5, 2, and 7 (in that order), and draw the resulting array (of capacity 10) representing the simulated chain.

Answer:

We can construct the simulated chain in the following way:

- Set c.firstNode = 5 (pointer to first node on the simulated chain c is of type int)
- Set node[5].next = 2 (pointer to second node on simulated chain)
- Set node[2].next = 7 (pointer to next node)
- Set node[7].next = -1 (indicating that node 7 is the last node on the chain).

The following figure shows the resulting simulated chain.

node	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
next			7			2		-1		
element			e_1			e_0		e_2		