Trinity College Dublin
The University of Dublin

Faculty of Engineering, Mathematics & Science

School of Computer Science and Statistics

Integrated Computer Science Programme **Trinity Term 2015**
BA (Mod) CSLL
BA (Mod) Business & Computing
BA MSISS

CS2010:
Algorithms and Data Structures

Wednesday 29 April 2015 RDS Main Hall **9:30 – 12:30**

Dr. Hugh Gibbons, Dr. Vasileios Koutavas

Instructions to Candidates:

- This exam paper has TWO PARTS.
- Answer TWO of the three questions in PART A.
- Answer TWO of the three questions in PART B.
- Use SEPARATE answer books for each part.
- Each question is worth 25 marks.

Materials permitted for this examination:

- None.

✓

PART A

Question 1 [25 marks]

(a) i. Write the main operations in the API of the **LIFO stack** Abstract Data Type (ADT) and describe how they work. Give the **worst-case running time** of each of these operations when we implement the ADT using a **resizable array**. Briefly explain why each operation has the running time you gave.

Describe an optimal strategy for resizing the array that will result in $O(1)$ **amortized running time** of the ADT operations. *[10 marks]*

(b) The following is the class of the nodes of a linked list.

```
class LLNode {
    public int data;
    public LLNode next;
    public LLNode(int d, LLNode nxt) { data = d; next = nxt: }
}
```

Implement the method

```
boolean deleteSecond(LLNode head, int num)
```

which deletes the **second** occurrence of `num` in the list with head node `head`.

If the list has fewer than two occurrences of `num` then this method leaves the list unaffected. If the list has more than two occurrences of `num` then this method will only delete the second occurrence. The method returns **`true`** if it performs a deletion, and **`false`** otherwise.

(Note that because the method deletes the *second* occurrence of `num`, it never deletes the head of the list; this is why this method does not return an updated head of the list.) *[15 marks]*

Question 2 [25 marks]

(a) Write the following asymptotic costs in order, from the smallest to the largest, writing one of the symbols $<$ or $=$ between them. If two asymptotic costs are equal then their relative order is unimportant.

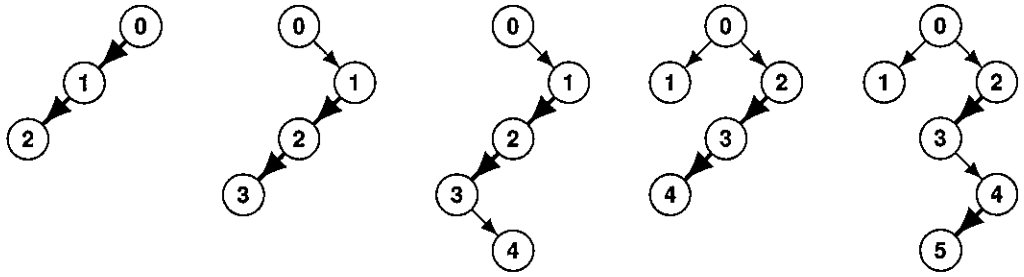
$\Theta(2n^2 + lg(n))$ $\Theta(2n^2 + n)$ $\Theta(2n^2 \cdot lg(n))$ $\Theta(10^{10})$ $\Theta(10^{10} log_3(n))$ $\Theta(log_2(n))$

[10 marks]

(b) A **binary tree** has nodes which are objects of the following class:

```
class TreeNode {
  int key;
  TreeNode left, right;
}
```

The **left height** of such a tree is the maximum number of **left links** in a path from the root to a leaf. Please note that a leaf is never `null`. As a convention, an empty tree has left height **-1**. A tree with only one node has left height **0**. Moreover, the left height of all the following trees is **2**.



Implement the following incomplete method which returns the left height of the tree whose root is the parameter of the method.

```
/**
 * @param node : the root of a TreeNode tree.
 * @return the left height of the tree whose root is the parameter.
 */
public int leftHeight(TreeNode node) {
  if (node == null) return -1;
  //TODO...
}
```

[15 marks]

Question 3 [25 marks]

A software engineer was asked to design an algorithm which will input two **unsorted** arrays of integers, A (of size N) and B (also of size N), and will output **true** when all integers in A are present in B. The engineer came up with **two** alternatives:

```
boolean isContained1(int[] A, int[] B) {
  boolean AInB = true;
  for (int i = 0; i < A.length; i++) {
    boolean iInB = linearSearch(B, A[i]);
    AInB = AInB && iInB;
  }
  return AinB;
}
boolean isContained2(int[] A, int[] B) {
  int[] C = new int[B.length];
  for (int i = 0; i < B.length; i++) { C[i] = B[i] }
  sort(C); // heapsort
  boolean AInC = true;
  for (int i = 0; i < A.length; i++) {
    boolean iInC = binarySearch(C, A[i]);
    AInC = AInC && iInC;
  }
  return AinC;
}
```

Here `sort(C)` sorts array C using **heapsort**, and `binarySearch(C,A[i])` uses binary search to return **true** if `A[i]` appears in C and **false** otherwise. Similarly, `linearSearch(B,A[i])` uses linear search to return **true** if `A[i]` appears in B and **false** otherwise.

(a) Calculate the worst-case running time of each of the two implementations using the asymptotic Θ notation. Your answer should take advantage of equalities between Θ expressions to present **the simplest possible expression** of the running time. You should adequately explain your answer. *[10 marks]*

(b) For each implementation, how much extra memory space is it required to store copies of the elements in A and B? You should take into account any copies made within the methods `sort`, `linearSearch`, and `binarySearch`, and explain your answer. *[10 marks]*

(c) Which of the two implementations would you advise the engineer to pick and why? What would be the benefits and drawbacks of this choice? *[5 marks]*

PART B

[*In presenting Java methods, give clear explanations with the code*]

Question 4 [25 marks]

a) The function

```
double sqrt_bin(double x)
{
      double y = 1.0;
      while ( y*y < x )
        y = 2.0*y;
      return binary_sqrt(0, y, 0.01, x);
} // sqrt_bin
```

returns an approximation of the square root of a number, x, making use of the function, `binary_sqrt`, which finds the the square root of a number using the techique of binary search.

Implement the Java function:

```
double binary_sqrt(double low, double high,
                   double tol, double x)
{ //Pre : low² ≤ x < high²
```

using the technique of Binary Search that will return a result, r, such that

$$r \leq \sqrt{x} < r+tol.$$

where `tol` is a small number, e.g. `0.01` and initially, $low^2 \leq x < high^2$.

[8 marks]

b) The n^{th} fibonacci can be calculated using the following recursive definition:

```
fib(0) = 0
fib(1) = 1
fib(n+2) = fib(n+1) + fib(n), if n ≥ 0
```

i) Implement a non-recursive Java method, i.e. an iterative Java method

```
int fib(int k)
```

that will return the k^{th} fibonacci number.

ii) Based of the mathematical result

$$fib(2*n+1) = (fib(n))^2 + (fib(n+1))^2$$
$$fib(2*n) = fib(n)*(2*fib(n+1) - fib(n))$$

Implement a recursive Java method that will find the k^{th} fibonacci number in O(log k), i.e. complete the missing code in the following method:

```
int fib_fast(int k)
{
    if ( k == 0 )
        return 0;
    else if ( k == 1 || k == 2 )
          return 1;
    else
    {
          « missing code »
    }
} // fib_fast
```

[9 marks]

c) A derangement of items 1..n is a permutation, p, such p(k) ≠ k.

The number of derangements of n items is given by the recursive definition

$$D(1) = 0$$
$$D(2) = 1$$
$$D(n) = (n-1)(D(n-1) + D(n-2)), \quad n>2$$

Implement a non-recursive i.e. an iterative method

```
int der(int k)
```

that will return the number of derangement of k items.

i.e. fill In the body of the loop in the following method:

```
int der(int k)
{
    int prev = 0;
    int curr = 1;
    int next = 0;

    for (int j = 2; j < k; j = j+1)
    {
        « loop body »
    }
    return next;
} // der
```

[8 marks]

Question 5 [25 marks]

a) Implement a Java method

```
int[] left_shift(int[] arr, int k)
```

that will return an array that shifts left cyclically the array, arr, by k places.

For example,

if arr = {11, 13, 17, 19, 23}, then shift_left(3) would return the array {19, 23, 11, 13, 17}.

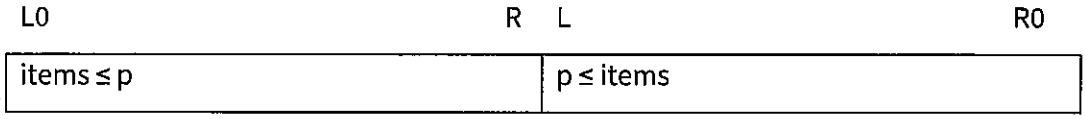
[6 marks]

b) Assume a Java class has an array attribute, String [] arr, and integer attributes, int L,R.

i) Present a Java method

```
void partition(int L0, int R0, String p)
```

that will partition in place the array, arr, into 2 sections. After partitioning, the items in the left section are less than or equal to the pivot item , p, and the pivot item, p, is less than or equal to all the items in the right section, i.e. after partitioning, the array, arr, satisfies



[7 marks]

ii) Show how the method, `partition`, would partition an array with the following ten String items about the pivot item, "jun".

k	0	1	2	3	4	5	6	7	8	9
arr[k]	feb	mar	apr	may	jun	jul	aug	sep	oct	nov

[6 marks]

iii) Using the routine, `partition`, present a Java method

```
void qsort(int left, int right)
```

that will use the algorithm for Quicksort to sort the array section `arr[left..right]` of the array attribute, `arr`, with the property that the first recursive call sorts the smaller of the two sections of the partition.

[6 marks]

Question 6 [25 marks]

Present a Java class that will provide methods that will:

a) Generate all combinations of choosing k numbers from the N numbers 0 to N-1.

If N = 5 and k = 3 then the the combinations of choosing 3 from 5 are:

```
0 1 2     0 1 3     0 1 4     0 2 3     0 2 4     0 3 4
1 2 3     1 2 4     1 3 4
2 3 4.
```

[10 marks]

b) Find a solution to the 8-Queens problem of placing 8 queens on an 8x8 chessboard so that no queen can take another with the constraint that no queen lies on any of the two main diagonals.

As an example, there is a solution such as:

	0	1	2	3	4	5	6	7
0			Q					
1					Q			
2		Q						
3								Q
4	Q							
5							Q	
6				Q				
7						Q		

[15 marks]