University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

SEMESTER I EXAMINATION – 2011/2012

COMP 20010/20210

Data Structures & Algorithms I

Prof. ?

Mr. J. Dunnion

Dr Eleni Mangina

Time allowed: 2 hours

Instructions for candidates

Answer two questions. All questions carry equal marks (50).

Use of calculators is prohibited.

Instructions for invigilators

Use of calculators is prohibited.

1. Answer parts (a) to (f). **50 marks in total**

(a) (5 marks)

Briefly define the following variable modifiers in terms of scope (or visibility) of instance variables: public, protected, private, static, final.

Solution: (1 Mark each)
Public: Anyone can access public instance variables
Protected: Only methods of the same package or of its subclasses can access protected instance variables
Private: Only methods of the same class (not methods of the subclass) can access private instance variables.
Static: It is used to declare a variable that is associated with the class, not with individual instances of that class.
Final: A final instance variable is one that must be assigned an initial value, and then can never be assigned a new value after that.

(b) (10 marks)

Briefly define the following terms in the context of object-oriented design and provide an example: Inheritance and polymorphism.

Solution:
The O-O paradigm provides a modular and hierarchical organising structure for reusing code, thrugh inheritance. It allows the design of generic classes that can be specialised to more particular classes, with the specialised classes reusing the code from the generic class (super class). In O-O polymorphism refers to the ability of an object variable to take different forms.

- *(5 Marks) Inheritance is a mechanism for extending existing classes by adding instance variables and methods:*
  class SavingsAccount extends BankAccount
  {
  added instance variables
  new methods
  }
- *(5 Marks) Polymorphism: Ability to treat objects with differences in behavior in a uniform way*
  The first method call
  withdraw(amount);
  is a shortcut for
  this.withdraw(amount);
  this can refer to a BankAccount or a subclass object

(c) (5 marks)

Using examples describe the purpose of Java's interface and implements constructs.

Solution:
Interfaces allow one to specify a set of methods that must be implemented by a class. This encourages modularity by separating the behaviour that an ADT must support, from various implementations. For example, a list might have the following methods

interface List {
int size();
void insert(Object o);
void remove(Object o);}

One can now go on to implement List in various ways:

class LinkedList extends LinkedStructures implements List {…}
class ArrayList extends ArrayStructures implements List {…}

etc. The key point is that the "promises" made by the implements constructs need bear no relation at all to the inheritance hierarachies used by the implementation.

(d) (5 marks)

You have been hired to build an object-oriented Java program to manage University's grade book. All **students** have a **name** and **type** (represented as Strings). For **undergraduate** students, the database records the number of years until graduation (an integer). For **postgraduate** students the database stores the years of study (an integer). Finally, an student's **description** method returns the concatenation of the student's name and type separated by space. Write Java code that implements these three classes. *[5 marks]*

Solution:

```
class Student {
        String name;
        String type;
        String description() {return name + " " + type;}}
class UnderGraduateStudent extends Student {
        int yearsleft;}
class PostGraduateStudent extends Student {
        int yearsStudy;}
```

(e) (10 marks)

In the following program, called ComputeResult, what is the value of result after each numbered line executes? [*10 marks*]

```
public class ComputeResult {
    public static void main(String[] args) {
            String original = "software";
            StringBuffer result = new StringBuffer("hi");
            int index = original.indexOf('a');
              /*1*/   result.setCharAt(0, original.charAt(0));
              /*2*/result.setCharAt(1,
original.charAt(original.length()-1));
              /*3*/   result.insert(1, original.charAt(4));
              /*4*/   result.append(original.substring(1,4));
              /*5*/    result.insert(3, (original.substring(index, index+2)
+ " "));
            System.out.println(result); }}
```

Solution:

1. si, 2. se, 3.swe, 4. sweoft, 5. swear oft

(f) (15 marks)

Consider the following implementation of a class Square

```
public class Square
{
        private int sideLength;
        private int area;

        public Square(int initiallength)
        {
                sideLength = initiallength;
                area = sideLength * sideLength;
        }

        public int getArea ()  {  return area; }
        public void grow ()  {  sideLength = 2 * sideLength;}
}
```

What error does this class have? How would you fix it?

Solution:

If the method grow is called, the value of the instance variable area will no longer be correct. You should make area a local variable in the getArea method and remove the calculation from the constructor.

```
public class Square
{
  private int sideLength;
  public Square(int length)
  {     sideLength = length;   }
  public int getArea ()
  {     int area; // local variable
    area = sideLength * sideLength;
    return area;
  }
  public void grow ()
  {    sideLength = 2 * sideLength;   }
}
```

2. Answer parts (a) to (c). **50 marks in total**

(a) (20 marks)

(i) A technique that can be used to describe the running time of an algorithm is known as *asymptotic notation*. One form of this technique is known as "Big-Oh" notation. Give the definition for this notation. **(4 marks)**

Solution:
The "Big-Oh" Notation (4 marks):
Given functions ***f(n)*** *and g(n), we say that* ***f(n)*** *is* ***O(g(n))*** *if and only if there are positive constants* ***c*** *and* n_0 *such that* $f(n) \leq c\,g(n)$ *for all* $n \geq n_0$.

(ii) In the following, use either a direct proof (by giving values for c and n_0 in the definition of "Big-Oh" notation) or cite one of the rules given in the textbook or in the lecture slides:
Show that if f(n) is O(g(n)) and d(n) is O(h(n)), then f(n)+d(n) is O(g(n)+h(n)).

(6 marks)

Solution:
Recall the definition of the "Big-Oh" Notation: we need constants $c>0$ *and* $n_0 \geq 1$ *such that* $f(n) + d(n) \leq c(g(n) + h(n))$ *for every integer* $n \geq n_0$. *f(n) is O(g(n)) means that there exists* $c_f > 0$ *and an integer* $n_{0f} \geq 1$ *such that* $f(n) \leq c_f\, g(n)$ *for every* $n \geq n_{0f}$. *Similarly, d(n) is O(h(n)) means that there exists* $c_d > 0$ *and an integer* $n_{0d} \geq 1$ *such that* $d(n) \leq c_d\, g(n)$ *for every* $n \geq n_{0d}$. *Let* $n_0 = \max(n_{0f}, n_{0d})$, *and* $c = \max(c_f, c_d)$. *So* $f(n) + d(n) \leq c_f g(n) + c_d h(n) \leq c(g(n) + h(n))$ *for* $n \geq n_0$. *Therefore f(n)+d(n) is O(g(n)+h(n)).*

(iii) In the following, use either a direct proof (by giving values for c and n_0 in the definition of big-Oh notation) or cite one of the rules given in the textbook or in the lecture slides:
Show that $3(n+1)^7 + 2n\log n$ *is* $O(n^7)$. **(4 marks)**

Solution:
$\log n$ *is* $O(n)$
$2n\log n$ *is* $O(2n^2)$
$3(n+1)^7$ *is a polynomial of degree 7, therefore it is* $O(n^7)$
$3(n+1)^7 + 2n\log n$ *is* $O(n^7+2n^2)$ *[based on (a)(ii) above]*
$3(n+1)^7 + 2n\log n$ *is* $O(n^7)$

(iv) Algorithm A executes $10n\log n$ operations, while algorithm B executes n^2 operations. Determine the minimum integer value n_0 such that A executes fewer operations than B for $n \geq n_0$ **(6 marks)**

Solution:
We must find the minimum integer n_0 *such that* $10n\log n < n^2$. *Since n describes the size of the input data set that the algorithms operate upon, it will always be positive. Since n is positive, we may factor an n out of both sides of the inequality, giving us* $10\log n < n$. *Let us consider the left and right hand side of this inequality. These two functions have one intersection point for* $n > 1$, *and it is located between* $n = 58$ *and* $n = 59$. *Indeed,*

10log58 = 58.57981 > 58 and 10log59 = 58.82643 < 59. So for 1≤ n ≤ 58, 10nlogn ≥ n^2, and for n ≥ 59, 10nlogn < n^2. So n_0 we are looking for is 59.

(b) (10 marks)

(i) When comparing algorithms, the typical approach adopted is to evaluate each algorithm's performance in terms of "Big-Oh" and then to compare performances based on a "hierarchy of functions". What do we mean by "hierarchy of functions" and how is it used to compare algorithms?

(4 marks)

Solution:
Hierarchy of Functions (2 marks):
There is a mathematical hierarchy for functions whereby, for integer values, n > 0:
$1 < \log n < n < n^2 < n^3 < 2^n < ....$
Application (2 marks):
Algorithm performance is compared by working out "Big-Oh" for each algorithm and then using the hierarchy to work out which algorithm has better performance. For example, an algorithm that is O(1) is generally seen as being more efficient than an algorithm that is O(log n) or worse.

(ii) Based on the properties of the "Big-Oh" notation provide the running times of the following algorithms: **(6 marks)**

$5n^4 + 3n^3 + 2n^2 + 4n + 1$
$5n^2 + 3n\log n + 2n + 5$
$20n^3 + 10n\log n + 5$
$3\log n + 2$
2^{n+2}
$2n + 100\log n$

Solution:
$5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$. Note that $5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq (5+3+2+4+1)n^4 = cn^4$ for c = 15, when $n \geq n_0 = 1$ (1 mark)
$5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$. Note that $5n^2 + 3n\log n + 2n + 5 \leq (5+3+2+5)n^2 = cn^2$ for c = 15, when $n \geq n_0 = 2$ (note that nlogn is zero for n=1) (1 mark)
$20n^3 + 10n\log n + 5$ is $O(n^3)$. Note that $20n^3 + 10n\log n + 5 \leq 35n^3$ for $n \geq n_0 = 1$.(1 mark)
$3\log n + 2$ is O(logn). Note that $3\log n + 2 \leq 5\log n$, for $n \geq 2$ (note that logn is zero for n=1. That is why we use $n \geq n_0 = 2$ in this case) (1 mark)
2^{n+2} is $O(2^n)$. Note that $2^{n+2} = 2^n 2^2 = 4*2^n$; hence, we can take c = 4 and $n_0 = 1$ in this case (1 mark)
$2n + 100\log n$ is O(n). Note that $2n + 100\log n \leq 102n$, for $n \geq n_0 = 2$; hence we take c = 102 in this case (1 mark)

(c) (20 marks)

Another way of carrying out the analysis of an algorithm is to estimate the running time of the algorithm. To do this, we identify a set of *primitive*

operations to which we assign a fixed and equivalent running time. List the seven primitive operations that we used in the course and work out the running time of the following pseudo code algorithms. Which algorithm is better? Explain your answer.

```
Algorithm PrefixAverages1(A, n):
    Input: An integer array A of size n.
    Output: An array X of size n such that X[j] is the average of
            A[0], ..., A[j].
    Let X be an integer array of size n
    for j=1 to n-1 do
        a ← 0
        for k=1 to j do
            a ← a + A[k]
        X[j] ← a / (j+1)
    return X
```

```
Algorithm PrefixAverages2(A, n):
    Input: An integer array A of size n.
    Output: An array X of size n such that X[j] is the average of
            A[0], ..., A[j].
    Let X be an integer array of size n
    runningSum ← 0
    for j=0 to n-1 do
        runningSum ← runningSum + A[k]
        X[j] ← runningSum / (j+1)

    return X
```

Solution:

The seven primitive operations are : (7 marks)

- *Assigning a value to a variable*
- *Calling a method*
- *Performing an arithmetic operation*
- *Comparing two numbers*
- *Indexing into an array*
- *Following an object reference*
- *Returning from a method*

Solution (5 marks)

i.

```
Algorithm PrefixAverages1(A, n):
    Input: An integer array A of size n.
    Output: An array X of size n such that X[j] is the average of
    A[0], ..., A[j].

    Let X be an integer array of size n
    for j=1 to n-1 do
```

```
        a ← 0
        for k=1 to j do
            a ← a + A[k]
        X[j] ← a / (j+1)

    return X
```

Number of Operations

= 1 + op(outer_loop) + 1

= 2 + op(outer_loop)

*= 2 + (1 + (B-A+2) op(j>n-1) + (B-A+1) * 2 + op(inner_loop)) B = n-1 A = 1*

*= 2 + 1 + 2n + (n – 1) * 2 + op(inner_loop)*

= 1 + 4n + op(inner_loop)

op(inner_loop)

= $\Sigma_{j=1}^{n-1}$ 5j+7

= ($\Sigma_{j=0}^{n-1}$ 5j+7) – 7

= (5$\Sigma_{j=0}^{n-1}$ j + $\Sigma_{j=0}^{n-1}$ 7) – 7

= (5$\Sigma_{j=0}^{n-1}$ j + 7n) – 7

= $5n^2/2 + 7n – 7$

Number of Operations

= $1 + 4n + 5n^2/2 + 7n – 7$

= $5n^2/2 + 11n – 6$

(or as in page 178 of the book)

Solution (5 marks)

ii. **Algorithm** PrefixAverages2(A, n):

Input: An integer array A of size n.

Output: An array X of size n such that X[j] is the average of A[0], …, A[j].

```
    Let X be an integer array of size n
    runningSum ← 0
    for j=0 to n-1 do
        runningSum ← runningSum + A[k]
        X[j] ← runningSum / (j+1)

    return X
```

Number of Operations

= 1 + op(loop) + 1

= 2 + op(loop)

*= 2 + 1 + (B-A + 2)*op(j > n-1) + (B – A + 1) * (2+ op(content))*

*= 3 + (n-1 – 0 + 2)*2 + (n-1 – 0 + 1) * (2 + op(content))*

*= 3 + 2n -2 + (n) * (2 + op(content))*
= 2n + 1 + n(2+7)*
= 11n + 1
(or as in page 179 of the book)

Answer (3 marks):
The second algorithm is better because it has a linear number of operations, which means that it runs in linear time O(n), whereas the first algorithm has a polynomial number of operations, which means that it runs in polynomial time $O(n^2)$.

3. Answer parts (a) to (e). **50 marks in total**

(a) (10 marks)
What is a Vector? List the operations that are associated with this ADT.

Answer (10 marks):
Definition (4 marks): A Vector is a sequence data structure that supports accessing of its elements by their rank. The rank of an element e is an integer value that specifies the number of elements that come before e in the sequence. Elements may be inserted at any rank in the range [0, n+1] where n is the number of elements in the Vector. The Vector ADT operations are (6 marks):

- *size() Returns the number of elements.*
- *isEmpty() true if the Vector is empty, false otherwise*
- *elemAtRank(r): Returns the element with rank r: an error condition occurs if r < 0 or r > size() ¡V1*
- *replaceAtRank(r, e):Replace with e the element at rank r: an error condition occurs if r < 0 or r > size() ¡V1*
- *insertAtRank(r, e): Insert a new element e into S to have rank r: an error condition occurs if r < 0 or r > size()*
- *removeAtRank(r): Remove from S the element at rank r: an error condition occurs if r < 0 or r > size() ¡V1*

(b) (10 marks)
A key feature of Vectors is their use of extendable arrays. Explain the concept of an extendable array and list the steps that must be performed to extend an array when inserting a new element into a Vector.

Concept Definition (6 marks):
Vectors are commonly implemented using arrays. A key problem with this approach is that arrays have a fixed size, and therefore have an upper limit on the number of elements that can be held within the Vector. One solution to this problem is to use an extendable array. The idea behind extendable arrays is that, whenever the array that holds the elements of a Vector is filled up, we create a new larger array and copy the existing elements into it, maintaining rank ordering.
The steps to be followed when extending an array, A, are (4 marks):
1. Allocate a new array B of capacity 2N (where N is the size of A)
2. Let B[j] = A[j] for all j in the range [0, N-1]
3. Let A = B, that is replace array A with array B
4. Insert the new element into A

(c) (10 marks).
Give the algorithm for insertion at a given rank into a Vector that uses extendable arrays.

The insertion algorithm is:

```
Algorithm: insertAtRank(r, e):
  if (r < 0) or (r > n) then
    throw a BoundaryViolationException
```
2 Marks

```
  if n = capacity then
    capacity := capacity * 2;
    B := new array of size capacity
    for i = 0, 1, ..., n-1 do
      B[i] ← A[i]
    A ← B
```
4 Marks

```
  for i = n-1, n-2, ..., r do
    A[i+1] := A[i]
  A[r] := e
  n := n + 1
```
4 Marks

(d) (10 marks).

What is a List, and how does it differ to a Vector? Describe briefly how the List ADT can be implemented using a doubly linked list. Illustrate your answer by drawing boxes and lines diagrams that show the following: a list containing the strings "London", "New York", and "Paris"; and an empty list.

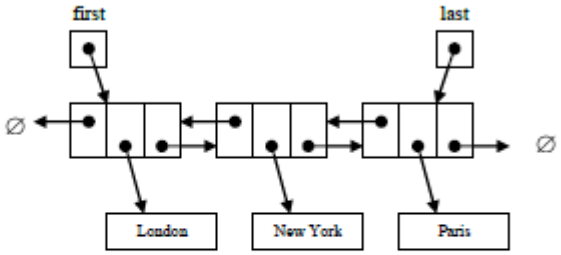
Definition (4 marks): A Vector is a sequence ADT that supports insertion and removal by rank, while a List is a sequence ADT that supports insertion and removal by position. Doubly linked list definition (5 marks): The List ADT can be implemented as a doubly linked list. Central to this implementation strategy is the definition of a node. For a doubly linked list, a node is broken into three parts:

Element: The actual value that is stored in the node

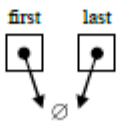
Next: A reference to the next node in the list, or null if no next node exists.

Prev: A reference to the previous node in the list, or null if no previous node exists. References to the first and last nodes in the list must also be maintained. Typically, the current size of the list is also maintained.

List containing the strings "London", "New York", and "Paris" (4 marks):



Empty List (2 marks):



(e) (10 marks).

Compare the array-based and linked-list based implementation strategies of a vector in terms of their runtime performance. Which one is better and why?

2 Marks

4 Marks

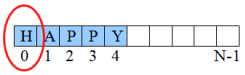
Vectors: Impl. Strategies

- Array-based Implementation:
  - Objects stored in an array
    - After each operation, the index of the object corresponds to the rank
  - Keep track of n, the current size of the vector
  - Finite Capacity (for now)
- Link-based Implementation:
  - Objects stored in special "nodes"
  - Nodes maintain ordering information
    - Link to the next and previous objects in the Vector.
    - Need auxiliary references for "front" and "rear" nodes.

Runtime Performance of Vectors

- The majority of methods for the Vector class have a running time of O(1).
- The exceptions to this are the insertAtRank(r,e) and the removeAtRank(r) methods.
  - These methods have O(n) running time because of the shift operations that must be performed on the arrays.
  - In the worst case, elements are inserted or removed with rank 0.
  - In this case, n-1 elements must be shifted, hence the O(n) running time.

Method	Size
size()	O(1)
isEmpty()	O(1)
elemAtRank(r)	O(1)
replaceAtRank(r,e)	O(1)
insertAtRank(r,e)	O(n)
removeAtRank(r)	O(n)



4 Marks

Linked Lists and Vectors

- Each Vector update method requires we traverse the list to a specified rank.
  - This means that ALL of these methods will have a running time of **O(n)**.
- For insertAtRank(r,e) and removeAtRank(r):
  - We have replaced the need for shift operation with the need for a traversal operation.
- For elemAtRank(r) and replaceAtRank(r,e):
  - We have ***introduced*** the need for a traversal operation.

Method	Size
size()	O(1)
isEmpty()	O(1)
elemAtRank(r)	O(n)
replaceAtRank(r,e)	O(n)
insertAtRank(r,e)	O(n)
removeAtRank(r)	O(n)

- This means that array-based implementations of Vectors are better!

4. Answer parts (a) to (c). **50 marks in total**

(a) (20 marks)

(i) What is a *Stack*? List the operations commonly associated with the Stack Abstract Data Type (ADT).

(5 marks)

Solution:
Definition (1.5 marks):A stack is an example of a LIFO data structure. Items can be added at any time, but only the most recently added item can be removed.
Stack Operations (2.5 marks):
push(o) - Inserts object o onto top of stack
pop() - Removes the top object of stack and returns it; if the stack is empty, an error occurs
isEmpty() - Returns the number of objects in stack
size() - Return a boolean indicating if stack is empty.
top() - Return the top object of the stack, without removing it; if the stack is empty, an error occurs.

(ii) Describe the output of the following series of stack operations on a single, initially empty stack: *push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().*

(4 marks)

Solution (bottom of the stack is to the left):
5
5 3
5
5 2
5 2 8
5 2
5
5 9
5 9 1
5 9
5 9 7
5 9 7 6
5 9
5 9 4
5 9
5

(iii) Describe in pseudo code a linear-time algorithm for reversing a queue Q. To access the queue, you are only allowed to use the methods of queue ADT. *Hint: Consider using a Stack.*

(5 marks)

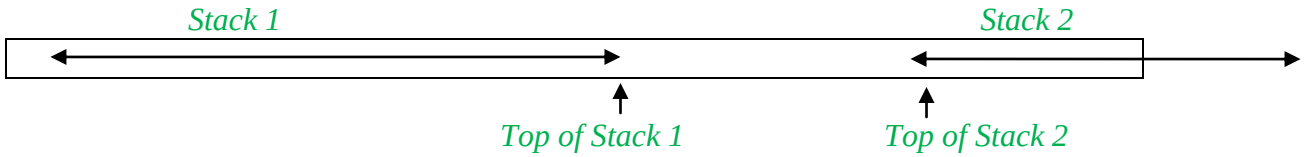
Solution:

We empty queue Q into an initially empty stack S, and then empty S back into Q.
Algorithm *ReverseQueue(Q)*
Input: queue Q
Output: queue Q in reverse order
S is an empty stack
while (!*Q.isEmpty()) **do***
*S.push(Q.dequeue()) **do***
while*(!S.isEmpty()) **do***
Q.enqueue(S.pop))

(iv) Describe how to implement two stacks using one array. The total number of elements in both stacks is limited by the array length; all stack operations should run in O(1) time.

(6 marks)

Solution:
Let us make the stacks (S_1 and S_2) grow from the beginning and the end of the array (A) in opposite directions. Let the indices T_1 and T_2 represent the tops of S_1 and S_2 correspondingly. S_1 occupies places $A[0...T_1]$, while S_2 occupies places $A[T_2...(n-1)]$. The size of S_1 is T_1+1; the size of S2 is $n-T_2+1$. Stack S_1 grows right while stack S_2 grows left. Then we can perform all the stack operations in constant time similarly to how it is done in the basic array implementation of stack except for some modifications to account for the fact that the second stack is growing in a different direction. Also to check whether anyone of these stacks is full, we check if S_1.size() + S_2.size()≥ n. In other words the stacks do not overlap if their total length does not exceed n.



(b) (15 marks)

(i) What is a *Queue*? List, giving a brief description for each one, the operations commonly associated with the Queue Abstract Data Type (ADT).

(5 marks).

Solution:
Definition (2 marks):
A queue is a FIFO data structure: items can be added at any time, but only the oldest item can be removed.
Queue Operations (3 marks):
enqueue(o) - Inserts object o onto the rear of the queue
dequeue() - Removes the object at the front of the queue
isEmpty() - Return a boolean indicating whether the queue is empty.
size() - Returns the number of objects in the queue

front() - Return the front object of the queue, without removing it

(ii) What is a *Deque*? Give the pseudo-code algorithm for insertion into the front of a Deque. Show, using diagrams, how this algorithm works by inserting Bejing followed by Shanghai into the front of a Deque. **(10 marks)**

Solution:
Definition (2 marks):
A Deque is a First or Last In First or Last Out (FLI-FLO) ADT

- *Elements may be added to either the front or back of the deque*
- *Elements may be removed from either the front or the back of the dequeu*

Algorithm (4 marks):
***Algorithm** insertFirst(o):*

```
// Create the node, setting prev to be first, and next to
// be the node that is next to first currently
node := new Node(o, first, first.next)
// set the prev of the node that is currently next to first to
// be the new node
first.next.prev := node
// set the next of the first node to be the next node
first.next := node
size := size + 1;
```

Illustration: Add Bejing to an Empty Dequeue (2 marks)

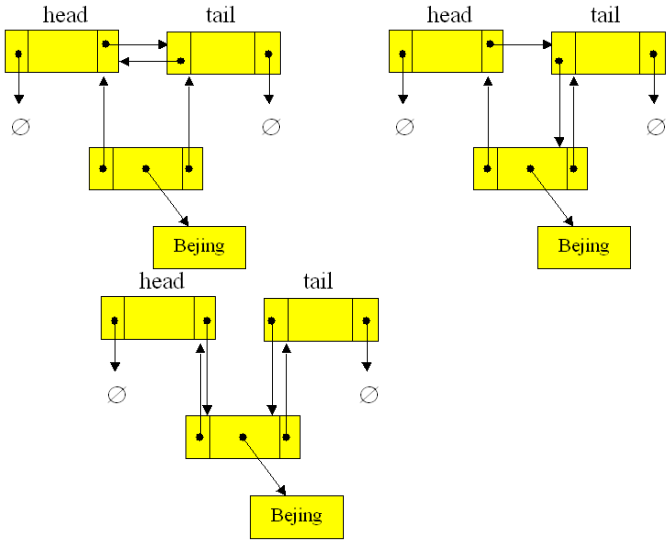
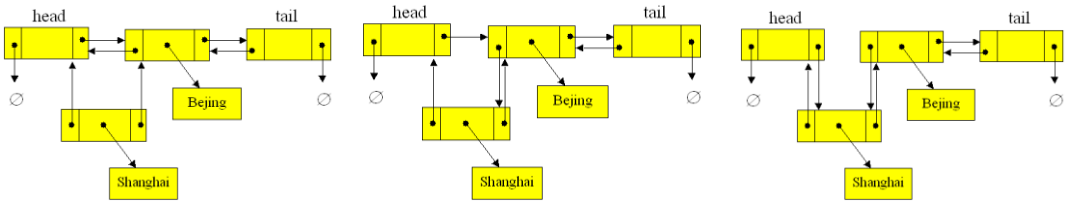


Illustration: Add Shanghai to the above Dequeue (2 marks)



(c) (15 marks)

Another variant of the basic Queue Abstract Data Type is a *Priority Queue*. Explain how this Priority Queue's work. Illustrate your answer by showing how the state of a sorted list-based implementation changes based on the following operations: insert(15, "mat"); insert(2, "the"); insert(8, "sat"), insert(7, "cat"); removeMin(); removeMin(); insert(12, "the"); removeMin(); insert(10, "on"); removeMin(); removeMin(); removeMin(). Show the state of the priority queue after each insert / remove operation.

Solution:
Definition (4 marks):
A priority queue is an abstract data type for storing a collection of prioritized elements. It supports arbeitary element insertion but requires that elements be removed based on their priority. Specifically, the remove operation removes the element with the highest priority (which, by convention, is the lowest priority value). In a Priority Queue, elements are stored based on their "priority", and as such, no concept of position exists.
State (6 marks):

Insert(15, "mat"): { (15, "mat") }
Insert(2, "the"): { (2, "the"), (15, "mat") }
Insert(8, "sat"): { (2, "the"), (8, "sat"), (15, "mat") }
Insert(7, "cat"): { (2, "the"), (8, "sat"), (15, "mat") }
RemoveMin(): { (7, "cat"), (8, "sat"), (15, "mat")} => "the"
RemoveMin(): { (8, "sat"), (15, "mat") } => "cat"
Insert(12, "the"): { (8, "sat"), (12, "the"), (15, "mat") }
RemoveMin(): { (12, "the"), (15, "mat") } => "sat"
Insert(10, "on"): { (10, "on"), (12, "the"), (15, "mat") }
RemoveMin(): { (12, "the"), (15, "mat") } => "on"
RemoveMin(): { (15, "mat") } => "the"
RemoveMin(): { } => "mat"