

**CS 14: Introduction to
Data Structures & Algorithms**

May 1, 2003

Midterm, Form:

First Name: _____

Last Name: _____

ID Number: _____

Signature: _____

This test contains two sections, a **MULTIPLE-CHOICE** section and a **SHORT-ANSWER** section. Please make sure to pace yourself properly so that you have time to work on both sections.

You may use **ONLY** the test's last sheet (front and back pages) for **scratch** work; I will **not** look at your scratch work and **nothing** there will be graded. **Do NOT** remove the scratch pages.

Good luck. :)

Part 1. MULTIPLE-CHOICE section. Please choose **only one** of the alternatives provided for each question and remember to mark your answers on the scantron form because **ONLY THOSE** will be considered. If scantron forms are not being used in this test, then make sure to **CIRCLE** your chosen answer.

Each question is worth **1 point** and **NO partial credit** is given. No negative points are given for wrong answers. Please note that you will **NOT** get any credit for leaving questions unanswered.

1. Which of the following sorting algorithms is recursive?
 - (a) BubbleSort.
 - (b) SelectionSort.
 - (c) InsertionSort.
 - (d) MergeSort.
 - (e) None of the above because sorting algorithms cannot be recursive.

2. Suppose that you have a sorted array of 500 integers and suppose also that you want to use **binary search** to find a specific value. What is the maximum number of array elements that you actually have to look at until you find the value that you are looking for (or until you can tell for sure that the value you're looking for isn't in the array)?
- (a) 8.
 - (b) 9.
 - (c) 10.
 - (d) 500.
 - (e) Using binary search in this case is a bad idea.
3. Which of the following are limitations of a pointer-based implementation of the **ADT List**?
- (a) The list will have a fixed maximum size determined at compile time.
 - (b) Element retrieval is not done in constant time.
 - (c) Insertions into the list have a **worst-case** runtime complexity of $\mathcal{O}[n^2]$.
 - (d) Both (a) and (b).
 - (e) Both (b) and (c).
4. The operation `enqueue()` affects
- (a) the front of the queue.
 - (b) the back of the queue.
 - (c) both the front and the back of the queue.
 - (d) either the front or the back of the queue, but not both at the same time, depending on the user's needs.

5. From **fastest** to **slowest** (in the **worst-case**), the correct order for the following algorithms is:
- Linear search, Binary search, Bubble Sort, Selection Sort, finding the maximum element of an unsorted array, finding the maximum element of a sorted array.
 - finding the maximum element of an unsorted array, finding the maximum element of a sorted array, Linear search, Binary search, Selection Sort, Bubble Sort.
 - finding the maximum element of a sorted array, Binary search, Linear search, finding the maximum element of an unsorted array, Selection Sort, Bubble Sort.
 - Bubble Sort, Selection Sort, Binary search, Linear search, finding the maximum element of a sorted array, finding the maximum element of an unsorted array.
 - Bubble Sort, Selection Sort, finding the maximum element of an unsorted array, Linear search, Binary search, finding the maximum element of a sorted array.
6. Deleting the **last** element of a **singly**-linked list with a last-element pointer has a **worst-case** runtime complexity of
- $\mathcal{O}[n \log_2(n)]$.
 - $\mathcal{O}[n]$.
 - $\mathcal{O}[1]$.
 - $\mathcal{O}[\log_2(n)]$.
 - $\mathcal{O}[n^2]$.
7. What is the correct **increasing** ordering of the following runtime complexities: $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^n]$, $\mathcal{O}[n]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[n^2]$?
- $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n^n]$.
 - $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[n]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n^n]$.
 - $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^n]$, $\mathcal{O}[n]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[n^2]$.
 - $\mathcal{O}[n^n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$.
 - $\mathcal{O}[n^n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$.

8. **Insertions** on the **SquareList** data structure have a **worst-case** runtime complexity of $\mathcal{O}[\sqrt{n}]$. Which of the statements below is correct?
- (a) Retrieving the minimum value has a **worst-case** runtime complexity of $\mathcal{O}[n]$.
 - (b) Retrieval of any element has a **worst-case** runtime complexity of $\mathcal{O}[\sqrt{n}]$.
 - (c) Deletion of any element has a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (d) Both (a) and (b).
 - (e) Both (a) and (c).
9. Retrieving the **minimum** value in a linked list of integers
- (a) always has a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (b) always has a **best-case** runtime complexity of $\mathcal{O}[1]$.
 - (c) always has an **average-case** runtime complexity of $\mathcal{O}[1]$.
 - (d) may never have a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (e) None of the above.
10. The **worst-case** runtime complexity of **linear search**, when applied to a problem of size n , is
- (a) Logarithmic in n .
 - (b) $\mathcal{O}[n]$.
 - (c) $\mathcal{O}[n \log_2(n)]$.
 - (d) $\mathcal{O}[n^2]$.
 - (e) Constant.
11. A **List Iterator**
- (a) is an object with responsibilities similar to that of an array index.
 - (b) lets you move from node to node, but it also **requires** you to know the details of how the list and the node structures are implemented.
 - (c) is also an ADT.
 - (d) Both (a) and (c).
 - (e) All of the above.

12. The **best-case** runtime complexity of the **naive** version of **BubbleSort**, when applied to a problem of size n , is
- (a) Logarithmic in n .
 - (b) $\mathcal{O}[n]$.
 - (c) $\mathcal{O}[n \log_2(n)]$.
 - (d) $\mathcal{O}[n^2]$.
 - (e) Constant.
13. Which of the options below could be used to indicate the end of a linked list?
- (a) The data stored in the last node is an end-of-list flag of some kind.
 - (b) The member variable `next` of the last node is set to `NULL`.
 - (c) An external pointer pointing to the last node is used.
 - (d) All of the above methods.
 - (e) None of the above methods.
14. **Reference counting**
- (a) is used to keep track of the number of node references in a list.
 - (b) is used to keep track of the number of iterators currently positioned at a given node.
 - (c) is used in the binary search algorithm.
 - (d) is used in C++ to keep track of the number of objects currently instantiated in your program.
 - (e) None of the above.
15. Testing can show that the implementation of an ADT is free of errors.
- (a) True.
 - (b) False.
16. A **stack** can be implemented using
- (a) an array.
 - (b) a singly-linked list.
 - (c) a doubly-linked list.
 - (d) all three data structures above.
 - (e) none of the data structures above.

17. **Black-box** testing of the implementation of an ADT is a testing procedure in which you don't have access to the ADT's
- (a) source code.
 - (b) interface.
 - (c) axioms.
 - (d) All of the above.
 - (e) There is no such thing as **black-box** testing of an ADT's implementation.
18. The runtime complexity of an algorithm
- (a) depends on the programming language being used.
 - (b) depends on how the data accessed by the algorithm is organized.
 - (c) depends on the speed of the computer on which the algorithm is being run.
 - (d) All of the above.
 - (e) None of the above.
19. Abstract classes in C++ must **not** have pure virtual member functions:
- (a) True.
 - (b) False.
20. Suppose that you are considering running two algorithms under the same conditions and on the same computer, on problems of size n . Furthermore, suppose that Algorithm **A** has a **worst-case** runtime complexity proportional to $\log_2(n)$ while algorithm **B** has a **worst-case** runtime complexity proportional to n . For **large enough** values of n ,
- (a) Algorithm **A** is **always faster** than algorithm **B**, in the **worst-case** scenario.
 - (b) Algorithm **A** is **always slower** than algorithm **B**, in the **worst-case** scenario.
 - (c) Algorithm **A** **may be faster or slower** than algorithm **B**, in the **worst-case** scenario, depending on those pesky constants which we typically ignore when we use O-notation.
 - (d) We cannot conclude anything about this situation when n is **large**; we should be looking at the case when n is **small**.
21. **FIFO** describes the behavior of a **queue**.
- (a) True.
 - (b) False.

22. The interface of an ADT
- (a) is the set of **operations** defined for it.
 - (b) is the set of **axioms** defined for it.
 - (c) defines the **semantics** of the ADT.
 - (d) All of the above.
 - (e) None of the above.
23. Suppose I gave you a homework assignment in which you have to implement a hierarchical file system. Which data structure would be most appropriate for this assignment?
- (a) A graph.
 - (b) A pointer-based list.
 - (c) An array-based stack.
 - (d) A pointer-based queue.
 - (e) A tree.
24. The **ADT List** operation `getLength()` can **never** have a runtime complexity of $\mathcal{O}[1]$, since you need to walk through the list to count how many nodes it has.
- (a) True.
 - (b) False.

Part 2. SHORT-ANSWER section. You may use **ONLY** the test's last sheet (front and back pages) for **scratch work**, and your final answers to this section must be **SHORT, CLEAN, COHERENT, LEGIBLE**, and written **ONLY** in the spaces provided, or your test will **NOT** be graded. **NO** exceptions will be made.

I will not look at your scratch work and nothing there will be graded. Do NOT remove the scratch pages.

Unless indicated otherwise, each question is worth 1 point and NO partial credit is given. No negative points are given for wrong answers. Please note that you will NOT get any credit for leaving questions unanswered.

1. [2 points] [This is a CS 12 question] Explain what the access modifier **protected** means within a class definition. You must write a **single sentence**. I do **not** want examples. I also do **not** want an answer of the form **Everyone has it**. That is **not** an acceptable sentence

in the context of this question [who's *everyone* referring to?]. If I have to guess what you're trying to say, you get **no credit** for your answer. Use **ONLY the boxed space provided below (it may appear on the next page)** and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!

2. [2 points] [This is a CS 10 question] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a function `int min(int a[], int length)` which takes as arguments an array of integers, `a`, and its length, `length`, and returns the *index* of the **smallest** integer stored in the array. Use **ONLY the boxed space provided below (it may appear on the next page)** and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!



3. [2 points] How does the **Selection Sort** algorithm (selecting the **minimum** on each pass) run on the array {7, 2, 3, 6} when you're sorting the array in **ascending** order? You should use the squares below to trace the algorithm. Start with the elements in the sequence written vertically, with the first (7) on the bottom-left square and the last (6) on the top-left square, then proceed horizontally to the right. You may not need all the squares, so don't feel obliged to use them all. The numbers {0,1,2,3} represent the indices of the array elements.

| | | | | | | | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 3 | 6 | | | | | | | | | | | | | |
| 2 | 3 | | | | | | | | | | | | | |
| 1 | 2 | | | | | | | | | | | | | |
| 0 | 7 | | | | | | | | | | | | | |

4. [3 points] [This is a CS 12 question] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a function **int max(Node * head)** which takes as argument the **head** of a **singly**-linked list of **Node** objects (each storing an integer value) and returns the **largest** integer stored in the list. You may assume that **head** is already *not* NULL by the time your function is called. The **Node** class is described by the UML diagram below. Use **ONLY the boxed space provided below (it may appear on the next page)** and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!

| Node |
|---------------------------|
| + int value |
| + Node * next |
| + Node(int val) |
| + Node(int val, Node * n) |

5. [3 points] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a **templated** function **bool IsAscending(T a[], int length)** to determine if an array is sorted in **ascending** order. The function should return true if the array passed to it is sorted in **ascending** order, false otherwise.

Answer Key for Exam A

Part 1. MULTIPLE-CHOICE section. Please choose **only one** of the alternatives provided for each question and remember to mark your answers on the scantron form because **ONLY THOSE** will be considered. If scantron forms are not being used in this test, then make sure to **CIRCLE** your chosen answer.

Each question is worth **1 point** and **NO partial credit** is given. No negative points are given for wrong answers. Please note that you will **NOT** get any credit for leaving questions unanswered.

1. Which of the following sorting algorithms is recursive?
 - (a) BubbleSort.
 - (b) SelectionSort.
 - (c) InsertionSort.
 - (d) MergeSort.**
 - (e) None of the above because sorting algorithms cannot be recursive.

2. Suppose that you have a sorted array of 500 integers and suppose also that you want to use **binary search** to find a specific value. What is the maximum number of array elements that you actually have to look at until you find the value that you are looking for (or until you can tell for sure that the value you're looking for isn't in the array)?
 - (a) 8.
 - (b) 9.**
 - (c) 10.
 - (d) 500.
 - (e) Using binary search in this case is a bad idea.

3. Which of the following are limitations of a pointer-based implementation of the **ADT List**?
- (a) The list will have a fixed maximum size determined at compile time.
 - (b) Element retrieval is not done in constant time.**
 - (c) Insertions into the list have a **worst-case** runtime complexity of $\mathcal{O}[n^2]$.
 - (d) Both (a) and (b).
 - (e) Both (b) and (c).
4. The operation `enqueue()` affects
- (a) the front of the queue.
 - (b) the back of the queue.**
 - (c) both the front and the back of the queue.
 - (d) either the front or the back of the queue, but not both at the same time, depending on the user's needs.
5. From **fastest** to **slowest** (in the **worst-case**), the correct order for the following algorithms is:
- (a) Linear search, Binary search, Bubble Sort, Selection Sort, finding the maximum element of an unsorted array, finding the maximum element of a sorted array.
 - (b) finding the maximum element of an unsorted array, finding the maximum element of a sorted array, Linear search, Binary search, Selection Sort, Bubble Sort.
 - (c) finding the maximum element of a sorted array, Binary search, Linear search, finding the maximum element of an unsorted array, Selection Sort, Bubble Sort.**
 - (d) Bubble Sort, Selection Sort, Binary search, Linear search, finding the maximum element of a sorted array, finding the maximum element of an unsorted array.
 - (e) Bubble Sort, Selection Sort, finding the maximum element of an unsorted array, Linear search, Binary search, finding the maximum element of a sorted array.

6. Deleting the **last** element of a **singly**-linked list with a last-element pointer has a **worst-case** runtime complexity of
- (a) $\mathcal{O}[n \log_2(n)]$.
 - (b) $\mathcal{O}[n]$.**
 - (c) $\mathcal{O}[1]$.
 - (d) $\mathcal{O}[\log_2(n)]$.
 - (e) $\mathcal{O}[n^2]$.
7. What is the correct **increasing** ordering of the following runtime complexities: $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^n]$, $\mathcal{O}[n]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[n^2]$?
- (a) $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n^n]$.
 - (b) $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[n]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n^n]$.**
 - (c) $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n^n]$, $\mathcal{O}[n]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[n^2]$.
 - (d) $\mathcal{O}[n^n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[n]$, $\mathcal{O}[\sqrt{n}]$, $\mathcal{O}[\log_2(n)]$.
 - (e) $\mathcal{O}[n^n]$, $\mathcal{O}[n^2]$, $\mathcal{O}[n]$, $\mathcal{O}[n \log_2(n)]$, $\mathcal{O}[\log_2(n)]$, $\mathcal{O}[\sqrt{n}]$.
8. **Insertions** on the **SquareList** data structure have a **worst-case** runtime complexity of $\mathcal{O}[\sqrt{n}]$. Which of the statements below is correct?
- (a) Retrieving the minimum value has a **worst-case** runtime complexity of $\mathcal{O}[n]$.
 - (b) Retrieval of any element has a worst-case runtime complexity of $\mathcal{O}[\sqrt{n}]$.**
 - (c) Deletion of any element has a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (d) Both (a) and (b).
 - (e) Both (a) and (c).
9. Retrieving the **minimum** value in a linked list of integers
- (a) always has a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (b) always has a **best-case** runtime complexity of $\mathcal{O}[1]$.
 - (c) always has an **average-case** runtime complexity of $\mathcal{O}[1]$.
 - (d) may never have a **worst-case** runtime complexity of $\mathcal{O}[1]$.
 - (e) None of the above.**

10. The **worst-case** runtime complexity of **linear search**, when applied to a problem of size n , is
- (a) Logarithmic in n .
 - (b) $\mathcal{O}[n]$.**
 - (c) $\mathcal{O}[n \log_2(n)]$.
 - (d) $\mathcal{O}[n^2]$.
 - (e) Constant.
11. A **List Iterator**
- (a) is an object with responsibilities similar to that of an array index.
 - (b) lets you move from node to node, but it also **requires** you to know the details of how the list and the node structures are implemented.
 - (c) is also an ADT.
 - (d) Both (a) and (c).**
 - (e) All of the above.
12. The **best-case** runtime complexity of the **naive** version of **BubbleSort**, when applied to a problem of size n , is
- (a) Logarithmic in n .
 - (b) $\mathcal{O}[n]$.
 - (c) $\mathcal{O}[n \log_2(n)]$.
 - (d) $\mathcal{O}[n^2]$.**
 - (e) Constant.
13. Which of the options below could be used to indicate the end of a linked list?
- (a) The data stored in the last node is an end-of-list flag of some kind.
 - (b) The member variable `next` of the last node is set to `NULL`.
 - (c) An external pointer pointing to the last node is used.
 - (d) All of the above methods.**
 - (e) None of the above methods.

14. **Reference counting**

- (a) is used to keep track of the number of node references in a list.
- (b) is used to keep track of the number of iterators currently positioned at a given node.**
- (c) is used in the binary search algorithm.
- (d) is used in C++ to keep track of the number of objects currently instantiated in your program.
- (e) None of the above.

15. Testing can show that the implementation of an ADT is free of errors.

- (a) True.
- (b) False.**

16. A **stack** can be implemented using

- (a) an array.
- (b) a singly-linked list.
- (c) a doubly-linked list.
- (d) all three data structures above.**
- (e) none of the data structures above.

17. **Black-box** testing of the implementation of an ADT is a testing procedure in which you don't have access to the ADT's

- (a) source code.**
- (b) interface.
- (c) axioms.
- (d) All of the above.
- (e) There is no such thing as **black-box** testing of an ADT's implementation.

18. The runtime complexity of an algorithm

- (a) depends on the programming language being used.
- (b) depends on how the data accessed by the algorithm is organized.**
- (c) depends on the speed of the computer on which the algorithm is being run.
- (d) All of the above.
- (e) None of the above.

19. Abstract classes in C++ must **not** have pure virtual member functions:
- (a) True.
 - (b) False.**
20. Suppose that you are considering running two algorithms under the same conditions and on the same computer, on problems of size n . Furthermore, suppose that Algorithm **A** has a **worst-case** runtime complexity proportional to $\log_2(n)$ while algorithm **B** has a **worst-case** runtime complexity proportional to n . For **large enough** values of n ,
- (a) Algorithm A is always faster than algorithm B, in the worst-case scenario.**
 - (b) Algorithm **A** is **always slower** than algorithm **B**, in the **worst-case** scenario.
 - (c) Algorithm **A** **may be faster or slower** than algorithm **B**, in the **worst-case** scenario, depending on those pesky constants which we typically ignore when we use O-notation.
 - (d) We cannot conclude anything about this situation when n is **large**; we should be looking at the case when n is **small**.
21. **FIFO** describes the behavior of a **queue**.
- (a) True.**
 - (b) False.
22. The interface of an ADT
- (a) is the set of operations defined for it.**
 - (b) is the set of **axioms** defined for it.
 - (c) defines the **semantics** of the ADT.
 - (d) All of the above.
 - (e) None of the above.
23. Suppose I gave you a homework assignment in which you have to implement a hierarchical file system. Which data structure would be most appropriate for this assignment?
- (a) A graph.
 - (b) A pointer-based list.
 - (c) An array-based stack.
 - (d) A pointer-based queue.
 - (e) A tree.**

24. The **ADT List** operation `getLength()` can **never** have a runtime complexity of $\mathcal{O}[1]$, since you need to walk through the list to count how many nodes it has.
- (a) True.
 - (b) False.

Part 2. SHORT-ANSWER section. You may use **ONLY** the test's last sheet (front and back pages) for **scratch work**, and your final answers to this section must be **SHORT, CLEAN, COHERENT, LEGIBLE**, and written **ONLY** in the spaces provided, or your test will **NOT** be graded. **NO** exceptions will be made.

I will **not** look at your scratch work and **nothing** there will be graded. **Do NOT** remove the scratch pages.

Unless indicated otherwise, each question is worth **1 point** and **NO partial credit** is given. No negative points are given for wrong answers. Please note that you will **NOT** get any credit for leaving questions unanswered.

1. [2 points] [This is a CS 12 question] Explain what the access modifier **protected** means within a class definition. You must write a **single sentence**. I do **not** want examples. I also do **not** want an answer of the form **Everyone has it**. That is **not** an acceptable sentence in the context of this question [who's *everyone* referring to?]. If I have to guess what you're trying to say, you get **no credit** for your answer. Use **ONLY the boxed space provided below (it may appear on the next page)** and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!

Answer:

Protected member variables and protected member functions of a given class are directly accessible by name **only** within that same class, within its **derived** classes, **and** by functions and classes which are friends of the class in question.

2. [2 points] [This is a CS 10 question] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a function **int min(int a[], int length)** which takes as arguments an array of integers, **a**, and its length, **length**, and returns the *index* of the **smallest** integer stored in the array. Use **ONLY the boxed space provided below (it may appear on the next page)** and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!

Answer:

```

int min(int a[ ], int length)
{
    int index_of_min = 0;
    for (int i = 1; i < length; i++)
    {
        if (a[i] < a[index_of_min])
            { index_of_min = i; }
    }
    return index_of_min;
}

```

3. [2 points] How does the **Selection Sort** algorithm (selecting the **minimum** on each pass) run on the array {7, 2, 3, 6} when you're sorting the array in **ascending** order? You should use the squares below to trace the algorithm. Start with the elements in the sequence written vertically, with the first (7) on the bottom-left square and the last (6) on the top-left square, then proceed horizontally to the right. You may not need all the squares, so don't feel obliged to use them all. The numbers {0,1,2,3} represent the indices of the array elements.

| | | | | | | | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 3 | 6 | | | | | | | | | | | | | |
| 2 | 3 | | | | | | | | | | | | | |
| 1 | 2 | | | | | | | | | | | | | |
| 0 | 7 | | | | | | | | | | | | | |

Answer:

4. [3 points] [This is a CS 12 question] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a function **int max(Node * head)** which takes as argument the **head** of a **singly**-linked list of **Node** objects (each storing an integer value) and returns the **largest** integer stored in the list. You may assume that **head** is already *not* NULL by the time your function is called. The **Node** class is described by the UML diagram below. Use **ONLY** the boxed space provided below (it may appear on the next page) and please write **LEGIBLY**. Do any scratch work on the scratch pages, **NOT** here!

Answer:

```

int max(Node * head)
{
    Node * current_node = head;
    int max_value = (head -> value);
}

```

```

while (current_node != NULL)
{
    if ((current_node -> value) > max_value)
    { max_value = (current_node -> value); }

    current_node = (current_node -> next);
}
return max_value;
}

```

5. [3 points] Write the **simplest** and most efficient (from a runtime complexity perspective) C++ implementation of a **templated** function **bool IsAscending(T a[], int length)** to determine if an array is sorted in **ascending** order. The function should return **true** if the array passed to it is sorted in **ascending** order, **false** otherwise.

Answer:

```

template<class T>
bool IsAscending(T a[ ], int length)
{
    for (int i = 0; i < (length - 1); i++)
    {
        if (a[i] > a[i+1])
            return false;
    }
    return true;
}

```