CSE 214

Recitation 2: Linked Lists

1. [5 Minutes] For the following problems, state the most efficient data structure(s) to use for the situation and explain why (Array, Singly Linked List, Doubly Linked List):
   1. Elements can frequently be accessed randomly

Array

* 1. Elements can be accessed sequentially both forwards and backwards

Doubly-Linked List

* 1. Values can be inserted to the front of the structure

LinkedList

* 1. Values can be accessed in reverse order from the end to the beginning

Doubly-Linked List

* 1. The size of the collection has a fixed size to store data

Array

1. [10 Minutes] In the following table, fill out the worst case time complexities for each operation. Assume sorted means from least to greatest and that you have access to both **head** and **tail** (unless otherwise stated):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Unsorted | Sorted | Unsorted | Sorted |
| Singly | Singly | Doubly | Doubly |
| Linked | Linked List | Linked List | Linked List |
| List |  |  |  |
| Traversing Values | O(n) | O(n) | O(n) | O(n) |
| Inserting a node | O(1) | O(n) | O(1) | O(n) |
| Deleting a node that you have access to | O(n) | O(n) | O(1) | O(1) |
| Finding the minimum value | O(n) | O(1) | O(n) | O(1) |
| Remove head | O(1) | O(1) | O(1) | O(1) |
| Remove tail | O(n) | O(n) | O(1) | O(1) |
| Insert new head | O(1) | O(1) | O(1) | O(1) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Insert new tail | O(1) | O(1) | O(1) | O(1) |

1. [10 Minutes] Fill in the expressions for the following methods, which belong to a class SinglyLinkedList, which contains reference to the **head**, the **tail**, the **cursor**:

a. /\*\*

\* Reverses the linked list

\*\*/

public void reverse() {

Node curr = this.head; Node prev = null; Node next = null; while (curr != null) {

next = a ;

b ; prev = curr;

curr = c ;

}

this.head = d ;

}

a:curr.getNext b: curr.setNext(prev)

c: next

a. /\*\*

\* Removes the tail node and returns it

\*\*/

public Node removeTail() { Node curr = this.head; if(curr == a )

d: prev

return null; //The list is already empty Node prev = null;

while ( b != null) {

prev = curr; curr = c ;

}

this.tail = prev; if( d )

prev.setNext(null); return e ;

a: null

b: curr.getNext

c: curr.getNext() d: prev!= null

e:

1. [10 Minutes] We wish to store a sequence of doubles using either an array or a singly-linked list of nodes with a head reference. Each node stores a data value and a

reference to the next node. We know our sequence can contain up to 500 values. Assume that a double is 8 bytes and a memory reference is 4 bytes. You may ignore the references of the array and the head.

* 1. If we want to store 300 numbers in the sequence, which structure would be more memory efficient?

12 (b/n) \* 300 = 3600 bytes for SLL

500(b/n) \* 8 = 4000 bytes for array

* 1. If we want to store 600 numbers in the sequence, which structure would be more memory efficient?

12\*600 = 7200 for SLL

8\*600 = 4800 for array

* 1. How many numbers could we store in the sequence such that neither structure is more efficient than the other?

12\*x = 8\*500 -🡪 4000/12 = 333.33

* 1. Assuming the number of numbers in our sequence is N and that this number is known, find the time complexities of the following operations for both the array and the singly-linked list and explain:
     1. Find the first item in the sequence

SLL AND Array both O(1)

* + 1. Find the last item in the sequence

SLL is O(n) and Array is O(1)

* + 1. Insert a new head into the sequence

SLL is O(1) and Array is O(n)

* + 1. Insert a new tail into the sequence

SLL is O(n) and Array is O(1)