

CS 325 Data Structures

Exam #2 Review



This exam will cover selected topics from chapter 1, 2, 3 and 11 and topics introduced in class. You may use a single sheet of paper 8.5" by 11" front/back as a reference guide during the exam. Students must use their own reference sheet. You should also review your class notes, textbook, and assignments from this course.

Linked-Lists

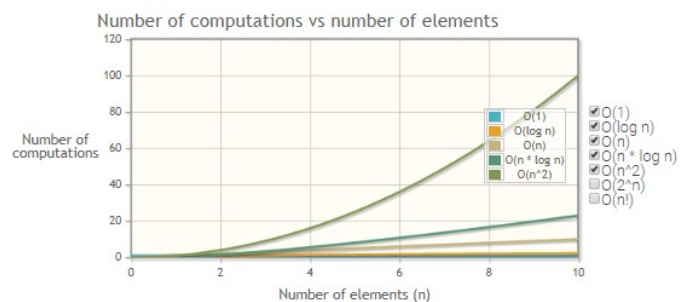
- ◆ Insert, Remove, Empty List, Full List, Search, Create List/Destroy List
- ◆ sorted, unsorted, single linked, double linked, circularly linked
- ◆ code and show memory diagrams for implementations with array or dynamic memory structure
 - dynamic memory using node class with data and next
 - memory requirements (4 bytes int, 4 bytes pointer/address, 1 byte boolean/char, 8 bytes double)
- ◆ big-O operations for each implementation with array vs. dynamic memory

Java Collections

- ◆ Declare and use Stack and Queue
- ◆ push(), add(), peek(), element(), pop(), remove(), isEmpty()

Algorithm Analysis

- ◆ Describe concept of best, average, and worst case
- ◆ Scaling and limits of empirical testing
- ◆ Identify highest order term
- ◆ Identity code as $O(1)$, $O(N)$, $O(N^2)$
- ◆ Relative efficiencies of $O(1)$, $O(N)$, $O(\log_2 N)$, $O(N \log_2 N)$, $O(N^2)$



Sorting

- ◆ Be able to describe Bubble, Insertion, Selection, and Quick-Sort (overall approach, pivot) algorithms including swaps/comparisons, and big-O analysis
- ◆ Overall cost of swap vs. compare
- ◆ Describe best and worst case scenarios.

1. What is the big-O growth rate for the following?

```
int mystery1( int[] a, int value) -O(N)
    result = 0
    for x = 0 to a.length
        result = result + a[x]
    return result

int mystery2( int[][]a, int I)  - O(N)
    result = 0
    for c = 0 to 5
        for r = 0 to a.#cols-1
            result = result + a[r][c];
    return result

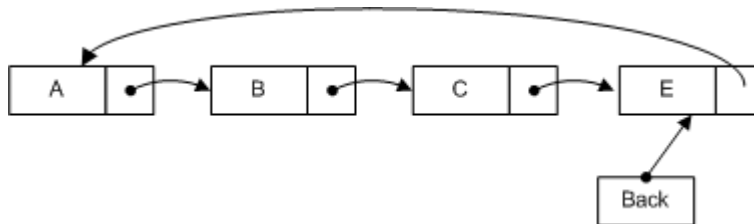
int mystery3( int[] data )  - O(1)
    result = 0
    for r = 0 to 1000
        result = result + data[ data.length-1 ];
    return result
```

2. Show the output for the following code. Assume a node with get and set data methods.

```
node i = new node;
i.data = 7;

node p;
node q;
p = new node();
p.data = i.data;
q = i;
q.data = 8;
System.out.println( i.data + "," + p.data + "," + q.data );
```

3. Given the following singly-linked circular list, show the code needed to delete node A. You can assume each node has data (char), and next (reference to a node).



4. How much memory would be required to hold the list from #3? Assume object references are 4 bytes each. Show your calculations.

5. How much memory would be required to hold the list from #3 if it were a double-linked list? Show your calculations.

6. *** won't ask this question on exam, but answers are shown in red ***

Show how you could represent the circular list from #3 with two arrays (char data[], and int next[]).

How much memory would be required to store this list and allow for up to 100 items?

(up to 6 characters = 6 bytes, 6 ints for next = 24 bytes, back = 1 int = 4 bytes => total = 34 bytes)

	<u>data</u>	<u>next</u>
0	C	2
1	A	4
2	E	1
3		
4	B	0
5		

back **2**

7. Show bubble, insertion, selection, and quicksort (data after 1st pass) for following:

20 15 40 30 50 10

Bubble Sort with 6 items

Bubble sort is a $O(N^2)$ sort

notice that the number of comparisons is $n-1$ down to 1,
swaps will vary

$n-1$ passes, unless there is a pass with no swaps, then done

Original Data

20 15 40 30 50 10

pass: 1 - comparisons: 5, swaps: 3

50 now in place

15 20 30 40 10 50

pass: 2 - comparisons: 4, swaps: 1

40 now in place

15 20 30 10 40 50

pass: 3 - comparisons: 3, swaps: 1

30 now in place

15 20 10 30 40 50

pass: 4 - comparisons: 2, swaps: 1

20 now in place

15 10 20 30 40 50

pass: 5 - comparisons: 1, swaps: 1

15 now in place

10 15 20 30 40 50

Sorted Data

10 15 20 30 40 50

Selection Sort with 6 items - note: finding next smallest,
can also sort by finding next largest

Selection sort is a $O(N^2)$ sort

notice that the number of comparisons is $n-1$ down to 1
notice that there will be at most 1 swap at each pass

Original Data

20 15 40 30 50 10

pass: 1 - comparisons: 5, swaps: 1

10 now in place

10 15 40 30 50 20

pass: 2 - comparisons: 4, swaps: 0

15 now in place

10 15 40 30 50 20

pass: 3 - comparisons: 3, swaps: 1

20 now in place

10 15 20 30 50 40

pass: 4 - comparisons: 2, swaps: 0

30 now in place

10 15 20 30 50 40

pass: 5 - comparisons: 1, swaps: 1

40 now in place

10 15 20 30 40 50

Sorted Data

10 15 20 30 40 50

Insertion with 6 items

Insertion Sort is a $O(N^2)$ sort

best case is that there will be one swap per pass
must use $n-1$ passes no matter

Original Data
20 15 40 30 50 10

pass: 1 - comparisons: 1, swaps: 1
15 now in place
15 20 40 30 50 10

pass: 2 - comparisons: 0, swaps: 0
40 now in place
15 20 40 30 50 10

pass: 3 - comparisons: 1, swaps: 1
30 now in place
15 20 30 40 50 10

pass: 4 - comparisons: 0, swaps: 0
50 now in place
15 20 30 40 50 10

pass: 5 - comparisons: 5, swaps: 5
10 now in place
10 15 20 30 40 50

Sorted Data
10 15 20 30 40 50

Quick Sort with 6 items

Quick Sort is a $O(N \log N)$ sort

Pick pivot point (first item, ... also could be middle, last, or random)

At end of partition pass, the pivot is in place

All data less than pivot is to left

All data greater than pivot is to right

Recursively call Quick Sort for left and right of pivot

Original Data

20 15 40 30 50 10

*//note various quick sort algorithms for swapping data minimally
so that items less than pivot are to left, and greater are to right,
for exam we are not going to show this detail*

After partition

-pivot value of 20 is in position

10 15 **20** 30 50 40

Now call quick sort with left of pivot ie. Quicksort([10, 15])

Now call quick sort with right of pivot ie. Quicksort([30, 50, 40])