Name (print): ____

PID: ____________________________

COMP 410 Spring 2019

Midterm Exam

*This exam is closed book, notes, calculators, cell phones, classmates, smart watches, everything but your own brain. You have 75 minutes to complete the exam. Do all your work on these exam pages. Please sign here (**and print your name up top of each page**) pledging that the work you submit is your own:*

Signature: ____

Q1 (4%): Consider a node M in a minimum binary heap; M is stored in the heap array at index **73**.

a) At what array index will we find the parent of node M ? **answer:** ____________

b) At what array index will we find the right child of node M ? **answer:** ____________

Q2 (2%): What is the maximum number of nodes that might be in a *complete binary tree* with height K, if we also have that the tree is *not a full binary tree*:

a) 2^(K+2)–1 b) 2*(K+1) c) 2^K – 1 d) 2^(K+1) e) 2^(K+1)–2 **answer:** __________

For Q3 through Q9, fill in the table cells with the best (tightest, closest) Big Oh time complexities. If you think an answer is "O(M log k)" (for example), you may just write "M log k", leave off O(…).

Q3 (3%) Fill in the table with worst-case time complexity for **list (array implementation)** of N items

operation	**add at i**	**remove at i**	**get ith**
Big Oh	**a)**	**b)**	**c)**

Q4 (3%) Fill in the table with worst-case time complexity for **stack (array implementation)** of N items

operation	**push**	**pop**	**top**
Big Oh	**a)**	**b)**	**c)**

Q5 (3%) Fill in with worst-case time complexity for **binary search tree** of N items (vanilla BST, not being balanced)

operation	**add**	**delete**	**contains**
Big Oh	**a)**	**b)**	**c)**

Name (print): ____

Q6 (3%) Fill in with average-case time complexity for **binary search tree** of N items (vanilla BST, not being balanced)

operation	add	delete	contains
Big Oh	a)	b)	c)

Q7 (3%) Fill in the table with worst-case time complexity for **queue (doubly linked cells)** of N items

operation	enque	deque	front
Big Oh	a)	b)	c)

Q8 (3%) Fill in the table with average-case time complexity for **min binary heap** of N items

operation	add	getMin	delMin
Big Oh	a)	b)	c)

Q9 (8%) Fill in this table comparing sort methods for N items. Use theoretical Big Oh notation

Time complexity	Worst case	Average case
bubble sort (on an array) N items	a)	b)
sort N items with a minimum binary heap	c)	d)
BST sort N items (vanilla BST, not being balanced)	e)	f)
put N items into linked list, keep it sorted each insert (inSort operation)	g)	h)

Name (print): ____

Q10 (12%) True or False (T / F):

a) ______ If P has worst case time complexity O(log N) then P also has worst case time O(N)

b) ______ Making a binary heap of N items by calling the insert operation N times is never as efficient as using the "magic" build operation

c) ______ Garbage collection in Java makes it impossible to run out of run-time stack space during execution.

d) ______ The run-time stack is dynamic memory from which objects are allocated on calls to "new"

e) ______ The principle of time/space trade off says that if a program runs efficiently in time then it must use storage (space) inefficiently

f) ______ Any program that uses recursion can be rewritten to use no recursion and still produce the same results

g) ______ Items put into a priority queue will never come out in LIFO order

h) ______ Pre-order traversal on the tree that represents a minimum binary heap always produces the elements in increasing priority sequence

i) ______ Array representation for a general binary tree is always fast to use, and space efficient

j) ______ For any set of unique data elements, if we insert these elements into an empty BST in different orders, we can get different final BST structures

k) ______ An N-ary tree is a tree with every node (except leaves) having exactly N children.

l) ______ In practical terms, it is possible for a recursive function to fail to produce results, when an iterative version of that function will succeed in producing correct results

Q11 (3%): Consider this code fragment for function **bubb**:

answer: _______________

```
public static long bubb(int N) {
  int[] arr = new int[N];
  for (int n=0; n<N; n++) { arr[n]=genRandInt(); }
  for (int i=0; i<N; i++) {
    for (int k=0; k<i*i; k++) { bubblesort( arr ); }
}
```

If we limit N to being a positive integer (not 0), and *assume bubblesort has worst case complexity* each time it runs, what is a good "Big Oh" complexity for the worst case execution time of function **bubb** ?

Q12 (3%): Consider this code fragment for function **mash**:

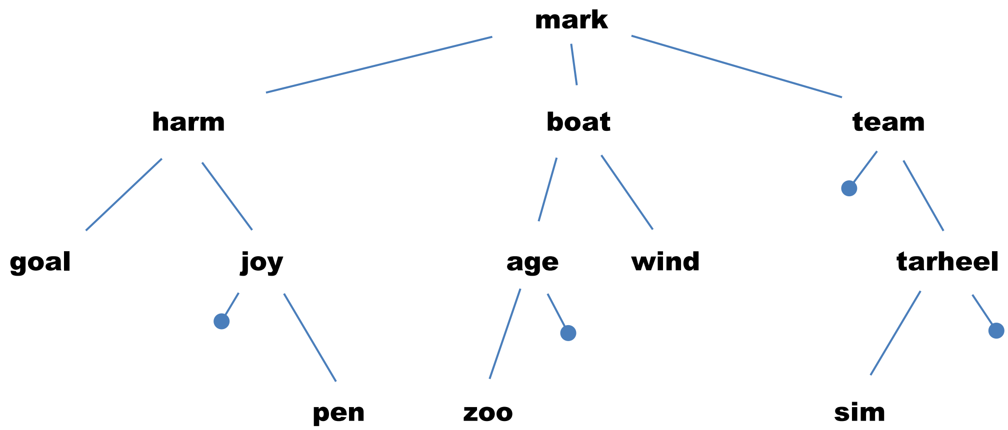
answer: _______________

```
public static long mash(int N) {
  int x = 2;
  for (int i=N; i>0; i--) {
    for (int j=i; j<i+3; j++) {
      for (int k=0; k<i; k++) {x *= i + j*k;} } }
}
```

If we limit N to being a positive integer (not 0), what is a good "Big Oh" complexity for the worst case execution time of function **mash** ?

Name (print): ____

Q13 (8%) Consider this 3-ary tree



Here are your answer choices:

1) mark harm goal joy pen boat age zoo wind team tarheel sim
2) goal harm joy pen mark boat zoo age wind team sim tarheel
3) mark harm boat team tarheel sim zoo pen goal joy age wind
4) goal pen joy harm zoo age wind boat sim tarheel team mark
5) mark harm boat team goal joy age wind tarheel pen zoo sim
6) none of the above

a) which sequence is a breadth-first traversal? __________

b) which sequence is a pre-order traversal? __________

c) which sequence is an in-order traversal? __________

d) which sequence is a post-order traversal? __________

Q14 (3%): Consider this code fragment for function **magic**:

```
public static long magic(int N) {
  if (N <= 1) return 2;
  return magic(N-1) * magic(N-1);
}
```

answer: ______________

If we limit N to being a positive integer (not 0), what is a good "Big Oh" complexity for the worst case execution time of function **magic** ?

Q15 (3%) Consider the program code to the right:

Which of these is most accurate when "main" is run?

a) the amount of run-time stack space that might be needed is finite
b) the amount of run-time stack space that might be needed is finite, but unbounded
c) the amount of run-time stack space that might be needed is infinite

answer: ______________

```
function main ( ) {
   var x = getUserInput();
   var result = foo(x);
   alert(result);
}
function foo ( n ) {
  if (n<=1) return 1;
  return n * foo(n-1);
}
```

Name (print): ____

Q16 (3%) Consider the program code to the right:

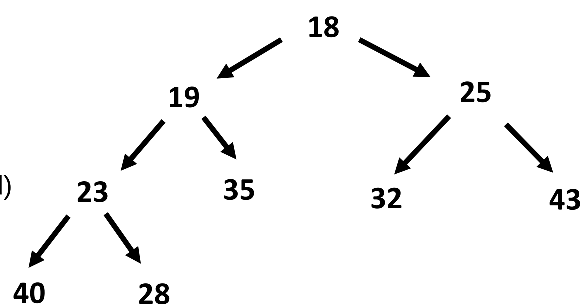
```
function main ( ) {
   var x = 7683910024;
   var result = foo(x);
   alert(result);
}
function foo ( n ) {
  if (n==1) return 1;
  return n * foo(n-2);
}
```

Which of these is most accurate when “main” is run?

a) the amount of run-time stack space that might be needed is finite
b) the amount of run-time stack space that might be needed is finite, but unbounded
c) the amount of run-time stack space that might be needed is infinite

answer: _______________

Q17 (3%) Consider the data structure represented at right

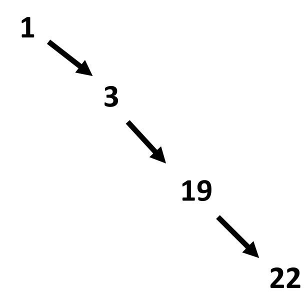


a) (T/F) ______ This could be a binary heap

b) **(T/F)** ______ This could be a BST (not being balanced)

c) **(T/F)** ______ This could be a doubly linked list

Q18 (4%) Consider the data structure/sequence represented at right



a) (T/F) ______ This could be a queue

b) (T/F) ______ This could be a stack

c) (T/F) ______ This could be a priority queue (done as list)

d) (T/F) ______ This could be a BST (not being balanced)

Q19 (5%): Binary Search Tree (not being balanced)

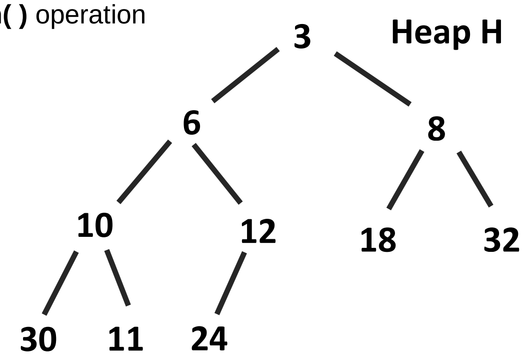
Starting with an initially empty Binary Search Tree (*vanilla, not being balanced*), show the tree that results after inserting the following string values in the order given left to right:

input, cpu, port, disk, usb, ram, net, keys, audio, screen

Name (print): ____

Q20 (5%) Consider the heap **H** shown to the right:

Show (in box below) the heap that results after a **delMin( )** operation



Q21 (5%) Consider the heap **H** shown above right (in previous question):

Show (in the box below) the heap that results after **add(5)** followed by **add(2)**

Name (print): ____

Q22 (3%): Lets add an operation to STACK. The operation is **max**, and it will return the largest element stored in the stack. If we implement a STACK with an array, which of the following expressions gives the most accurate description of the worst case time complexity of the **max** operation?

a) O(1)
b) O(N) **answer:** ___________
c) O(2N)
d) O(N^2)
e) O(2^N)

Q23 (3%): Consider this way to sort. You are given N integers in an array, the data source array. You are also told that the integer values will be in the range 0 to K inclusive, that there are no duplicate values, and that N < K. To sort them smallest to largest, you build another array of boolean with subscripts 0 to K.. You set every element in the boolean array to **false**. You go through the data source array and for each element you use the integer value stored as a subscript into the boolean array and mark that slot **true .** Finally, to get the sorted sequence, you go through the boolean array from subscript 0 up and print the subscript for every element that contains **true**

Which of the following expressions is the best description of the worst case time complexity for this sort :

a) O(2N)
b) O(N^2) **answer:** ___________
c) O(N * K)
d) O(N + K)
e) O(N^2 + K)
f) O(N + K + (N^2)/K)

Q24 (5%): Consider the BST **B** (basic, not balanced) below. Show its structure after "**delete (18)**" is complete. Show your final tree in the box: