3013 - Algorithms
Fall 2021
Exam 1
February 15 2022

Name: _____

READ THESE INSTRUCTIONS

- Use pencil only
- Initial top right corner of all pages (except the first one).
- Do not remove the staple from your exam.
- Do not crumple or fold your exam.
- Handwriting that is illegible (messy, small, not straight) will lose points.
- Indentation matters. Keep code aligned correctly.
- Answer all questions on the answer sheet provided. If one is not provided, then create one using your best guess at a creation spell.
- If your answer will not fit in the space (IT SHOULD) use the blank sheets at the end of the exam. Write *"On Back"* at end of question and label that question clearly on the back sheets.
- Help me ... help you!

Failure to comply will result in loss of letter grade

Grade Table (don't write on it)

Question	Points	Score
1	20	
2	25	
3	20	
4	15	
5	30	
6	32	
7	20	
8	35	
9	20	
Total:	217	

1. (20 points) List the complexities from fastest to slowest.

Complexity Choices

$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$

Solution: The spreadsheet below shows the growth of each choice based on the N value in column 1. The columns go from least cost on the left to greatest cost on the right

N	O(1)	O(lg N)	O(N)	O(N lg N)	O(N^2)	O(2^N)	O(N!)	O(N^N)
1	1	0	1	0	1	2	1	1
2	1	1	2	2	4	4	2	4
3	1	2	3	5	9	8	6	27
4	1	2	4	8	16	16	24	256
5	1	2	5	12	25	32	120	3125
6	1	3	6	16	36	64	720	46656
7	1	3	7	20	49	128	5040	823543
8	1	3	8	24	64	256	40320	16777216
9	1	3	9	29	81	512	362880	387420489
10	1	3	10	33	100	1024	3628800	10000000000
11	1	3	11	38	121	2048	39916800	285311670611
12	1	4	12	43	144	4096	479001600	8916100448256
13	1	4	13	48	169	8192	6227020800	302875106592253

---

2. Use the following linked list for the next 3 questions. Assume for each question, no changes were made to the list.

```
1  struct Node{
2     int data;
3     Node* next;
4     Node(int d){
5        data = d;
6        next = nullptr;
7     }
8  };
```

(a) (10 points) Write a function to traverse and print the values out in the list so it looks like: $11 => 33 => 17$ etc.
Your function header should look like: $void\ printList(Node*\ head)$

Solution:

```
1  void printList(Node* head){
2     while(head){
3        cout<<head->data<<endl;
4        if(head->next){
5           cout<<"=>";
6        }
7        head = head->next;
8     }
9     return;
10 }
```

(b) (15 points) Write a function to delete a given node from the list.
Your function header should look like: $bool\ deleteNode(Node*\ head,\ int\ key)$

Solution:

```
1  bool deleteNode(Node* &head,int key){
2     Node* prev = head;
3     Node* temp = head;
4     if(head->data == key){
5        head = head->next;
6        delete temp;
7        return true;
8     }
9      while(temp && temp->data != key){
10        temp = temp->next;
11     }
12     if(temp){
13        prev->next = temp->next;
14        delete temp;
15        return true;
16     }
17     return false;
18  }
```

---

3. Assign the correct complexity to each item below.

Complexity Choices

$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$ None of These

(a) (2 points) Inserting an element into a balanced binary search tree.

(a) **O(lg n)**

(b) (2 points) Finding an element in a list.

(b) **O(n)**

(c) (2 points) Finding an element in an ordered list.

(c) **O(n)**

(d) (2 points) Removing an element from a binary heap.

(d) **O(1)**

(e) (2 points) Finding an element in a binary search tree.

(e) **O(H)**

(f) (2 points) Adding an element to a binary heap.

(f) **O(lg n)**

(g) (2 points) Building a binary heap given an array of values.

(g) **O(n)**

(h) (2 points) Building a binary heap given a linked list of values.

(h) **O(n lg n)**

(i) (2 points) Remove an item from a linked list of values.

(i) **O(n)**

(j) (2 points) Insert an item into an ordered linked list of values.

(j) **O(n)**

4. (15 points) **Heapify**: Describe what it does, and why it is significant. Be thorough.

Solution: Heapify is the generic heap function that takes an array of unordered values, and puts them into "heap order". The significance of this method is that it can complete its task in ***O(N)*** time, and is what allows heap sort to work in ***O(n lg n)***.

Why is this possible? It's because of the relationship between inner nodes and leaves. Remember that a full complete tree has more leaves than inner nodes. So if we start process at the first inner node, and work our way up the tree, we only have to process half the array. If were only processing half of an array that represents a tree that's already bounding by a height of ***lg n***, (along with some hand waving) we end up with a cost of O(N).

---

5. Given a sequence of numbers: **19, 6, 8, 11, 4, 5**

   (a) (10 points) Draw a binary min-heap (in array form) by inserting the above numbers reading them from left to right

Solution:

X	4	6	5	19	11	8
0	1	2	3	4	5	6

   (b) (10 points) Show the min-heap after a after you deleteMin()

Solution:

X	5	6	8	19	11	
0	1	2	3	4	5	6

   (c) (10 points) Show the min-heap after another call to deleteMin()

Solution:

X	6	11	8	19		
0	1	2	3	4	5	6

---

6. List vs Array based data structures. Given a statement below, choose:
   List, **Array**, **Both**, **None**
   to indicate what the statement is implying.

   Choices: List Array Both None

   (a) (2 points) Directly access element in this structure.

(a) **Array**

   (b) (2 points) Bounded by size.

(b) **Array**

(c) (2 points) Easy to insert and delete from.

(c) **List**

(d) (2 points) Easy to implement.

(d) **Array**

(e) (2 points) More overhead.

(e) **List**

(f) (2 points) Can be sorted.

(f) **Array**

(g) (2 points) Must be allocated in the heap.

(g) **List**

(h) (2 points) Expensive to resize.

(h) **Array**

(i) (2 points) Grows and shrinks easily.

(i) **List**

(j) (2 points) Binary search can be performed on this.

(j) **Array**

(k) (2 points) Easily access each element in this structure.

(k) **Array**

(l) (2 points) Cannot be allocated in the heap.

(l) **None**

(m) (2 points) Must be statically declared.

(m) **None**

(n) (2 points) Items added to front or rear.

(n) **List**

(o) (2 points) Easier to delete from middle.

(o) **List**

(p) (2 points) Can be used to represent a binary tree.

(p) **Both**

---

7. Stack based memory VS Heap based memory.

(a) (10 points) Pros and cons of a Stack

Solution:

Stack

- very fast access
- don't have to explicitly de-allocate variables
- space is managed efficiently by CPU, memory will not become fragmented
- local variables only
- limit on stack size (OS-dependent)
- variables cannot be resized

(b) (10 points) Pros and cons of the Heap

Solution:

Heap

-variables can be accessed globally
-no limit on memory size
-(relatively) slower access
-no guaranteed efficient use of space, memory may become fragmented over time as blocks of memory are allocated, then freed
-you must manage memory (you're in charge of allocating (new) and freeing (delete) variables)
-variables can be resized using realloc() Only in C (not C++) when using malloc(). C++ has things like vectors that grow and shrink for us.

---

8. Given the following list of numbers: **27, 11, 6, 8, 19, 4, 43, 49, 31** process them from left to right and draw your resulting structure as indicated below.

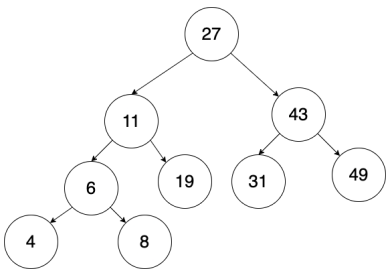
(a) (10 points) Array based binary search tree

Solution:

	27	11	43	6	19	31	49	4	8
0	1	2	3	4	5	6	7	8	9

(b) (10 points) Traditional binary search tree

Solution:



(c) (5 points) Is this tree complete?

Solution: Yes

(d) (5 points) Is this tree full?

Solution: No

(e) (5 points) Is this tree balanced?

Solution: Yes

---

9. (20 points) You are attending a party with n other people. Each other person i arrives at the party at some time s_i and leaves the party at some time t_i (where $s_i < t_i$). Once a person leaves the party, they do not return. Additionally, each person i has some coolness c_i. At all times during the party, you choose to talk to the coolest person currently at the party. (All coolness values are distinct.) If you are talking to someone, and someone else cooler arrives at the party, you leave your current conversation partner and talk to the new person. If the person you are talking to leaves the party, you go talk to the coolest person remaining at the party. (This might or might not be a person with whom you have already talked.) You are the first to arrive at the party and the last to leave. Additionally, you are the most popular person at the party, so everyone wants to talk with you.

   Describe a data structure which allows you to decide in $O(1)$ time to whom you should talk at any moment. You should be able to update this data structure in $O(logn)$ time when someone arrives or leaves.