Regis University CC&IS
CS324 Algorithms & Analysis
Alternate Homework 1 (v1)

Assignment
Place your answers directly under each question below -- do not delete the questions.

Part 1: Algorithm Complexity Analysis

1) Give the big-Oh characterization, in terms of **n**, of the running time of the pseudocode loops shown below: (*4 pts each*)

a)
```
x ← n + 10;
y ← n + 20;
while x > 0 do
    y ← y + x;
    x ← x / 2
```
It is known that when an algorithm divides or multiplies by 2: the resultant analysis can be generalized to the formula

$$f(n) = log_2(n + 10)$$

```
x ← n + 10;
y ← n + 20;
    AND
y ← y + x;
```

Are all constnt time assignments

Therefore the Big-Oh of the above is:

$$\boldsymbol{O(log_2(n))}$$

b)
```
w ← 0

for i ← 1 to $n^3$ do
    for j ← 2 to n do
        w ← w * i
```

Run times for nested loops are the result of multiplying the inner loops number of runs, vs the outer loop.

In this example the assignment and multiplication steps are constant.
Therefore the runtime is:

$$\boldsymbol{O(n^4)}$$

c)

```
x ← 1
for i ← 2n downto 1 do
    x ← x * i
```

Assignment and multiplication steps are constant.
The loop executes 2n since I is assigned 2n and iterates through every number to 1.

Therefore the time complexity is

$O(2n)$

d)

```
a ← 100
for i ← 1 to n do
    for j ← 1 to n/2 do
        a ← a + i
```

The outer loop executes n times.
The inner loop executes n/2 times
Assignment is constant.
We drop the constant from the runtime value of

$$O\left(\frac{n^2}{2}\right)$$

Therefore the time complexity is

$$O(n^2)$$

2) Complete problem C1.6 on page 45 of your textbook (*6 pts*)
Show ALL math conversion steps.

Starting with the base assumptions:

$$T(0) = 1$$

and

$$T(n) = T(n-1) + 2^n$$

We can then see that

$$T(n-1) = T(n-2) + 2^{n-1}$$

Continuing we get

$$T(n-2) = T(n-3) + 2^{n-2}$$

Looking at the following we can see the difference between any successive term is always a power of 2

$$T(n) - T(n-1) = 2^n$$
$$T(0) = 1$$

We can conclude from this fact and reminder from earlier that all terms in the sequence T(n) are powers of 2 – 1. That is:

$$T(n) = 2^{n+1} - 1$$

3) Given the following functions, determine the big-O notation for each (3 *pts each*)
   Hint: Use the rules given in the textbook.

   a) f(n) = 33n^2 + 5n^3 + 1000n^2
      a. $O(n^3)$

$$5n^3 \geq 33n^2 + 1000n^2$$
$$5n^3 \geq 1033n^2$$

Divide by n^2

$$5n \geq 1033$$

Divide by 2, then by 5 to obtain

$$n \geq 206.6$$

For all $n \geq 206.6, f(n) \leq 5n^3$

Therefore:

   b) f(n) = 6n^4 + 5n^4 + 20n^4
      a. O(n^4)

$$\text{c} = 20$$
$$20n^4 \geq 6n^4 + 5n\text{^}4$$
$$(20n^4)/n\text{^}4 \geq (11n^4)/n\text{^}4$$
$$20 \geq 11$$

Because this is not strictly true we cannot say that there is an n for which:

$$f(n) \leq cn^4, for\ all\ n \geq k$$

The term 20n^4 still grows the fastest and thus we say:

$$o(n^4)$$

   c) f(n) = 9n^3 + 2^n + 8n^2
      a. $O(n^3)$

Assuming we can prove O(n^3) we take the following:

$$9n^3 \geq 2n^n + 8n^2$$
$$9n^{\mathbf{3}} \geq nlog(2) + 8n^2$$

Dividing by n^2 we get the following:

$$\mathbf{9 \geq log(2) + 8}$$

Subtracting 8 from both sides we obtain:

$$1 \geq \log (2)$$
$$n \geq 1, f(n) \leq 9n^3$$
$$Therefore\ O(n^3)$$

   d) f(n) = 50n (log n) + 22n

a. $O(nlog(n))$

C = 50 because 50 is the greatest term.

$$50nlog(n) \geq 22n$$
$$50\log(n) \geq 22$$

Subtracting 22 from both sides get us the following which can be then divided by 28

$$28\log(n) \geq 0$$

Dividing by 28 yields:

$$\log(n) \geq 0$$

We know this is always true for all:

$$n \geq 1, f(n) \leq cn(logn)$$

4) Consider the following algorithm (shown with line numbers) that sorts the elements in an array of positive integers into ascending order.

```
1:    void selectionSort(int nums[], int N) {
2:       int temp,
3:           currentIdx = 0,
4:           lastIdx = N - 1;
5:       while (currentIdx < lastIdx) {
6:          minIdx = currentIdx;
7:          for (int i = currentIdx + 1; i <= lastIdx; i++) {
8:             if (nums[i] < nums[minIdx])
9:                minIdx = i;
10:         }
11:         if (minIdx != currentIdx) {
12                temp = nums[minIdx];
13:               nums[minIdx] = nums[currentIdx];
14:               nums[currentIdx] = temp;
15:         }
16:         currentIdx++;
17:      }
18:   }
```

a) Best case
   i. Describe the best case **data** for this algorithm. *(3 pts)*
      - The best case scenario is when the items are sorted.
        1. The outer loop executes n times.
        2. The inner loop runs the sum of n…0 times
           a. The result is a runtime of (n)*sum((n))
        3. Ultimately this algorithm runs in:
           a. $O(lastIdx * (lastIdx - 1)/2)$
           b. O((n*(n-1)/2)
        4.
   ii. Then, for this best case, state how often is line 9 executed. *(2 pts)*
      Line nine is executed zero times because num[i], the item after nums[minIdx], will always be sorted and thus always be less than the previous item.

b) Worst case

i. Describe the worst case **data** for this algorithm. *(3 pts)*

- The worst case is when the items are in reverse order.
  1. The outer loop runs n times
  2. The inner the sum of n-1 times.
  3. The runtime is:
  4. $O(\frac{n(n-1)}{2})$

ii. For this worst case, state how often is line 9 executed. *(2 pts)*

Worst case scenario line 9 is executed

$$\frac{n(n-1)}{2}$$

times

5) Suppose that for the worst case, given input size **n**:

Algorithm 1 does **f(n) = n^3 + 1** steps
Algorithm 2 does **f(n) = n^2 + 200** steps

For what input sizes is algorithm 1 faster than algorithm 2, in the worst case? (*5 pts)*
This occurs when

$$n^3 + \mathbf{1} < \boldsymbol{n^2} + \mathbf{200}$$
$$n^3 - n^2 - 200 < 0$$
$$(n - 10)(n^2 + 10n + 20) < 0$$

The inequality only holds if

$$n - 10 < 0$$

Therefore algorithm 1 is faster than algorithm 2 for input <u>sizes less than 10.</u>

6) Given the following ordered array: (*5 pts each*)

[0]	3
[1]	6
[2]	8
[3]	12
[4]	19
[5]	23
[6]	32
[7]	44
[8]	48
[9]	51
[10]	62
[11]	70
[12]	73
[13]	74

[14]	78
[15]	80
[16]	85
[17]	88

a) Using the binary search algorithm to search for the target value **32**
For each pass, list the high and low index, and the calculated middle index.
Finally, state the result of the search.

Using the binary search algorithm to fin 32:

PASS 1
HIGH: 17 LOW: 0 MID: 8
PASS 2
HIGH: 7 LOW: 0 MID: 3
PASS 3
HIGH: 7 LOW: 4 MID: 5
PASS z
HIGH: 7 LOW: 6 MID: 6
RESULT: 6

b) Using the binary search algorithm to search for the target value **79**
For each pass, list the high and low index, and the calculated middle index.
Finally, state the result of the search.

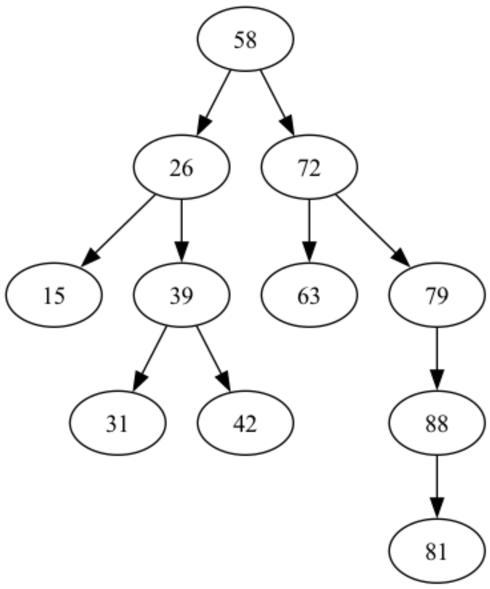
Using the binary search algorithm to fin 79:

PASS 1
HIGH: 17 LOW: 0 MID: 8Target: 79
PASS 2
HIGH: 17 LOW: 9 MID: 13 Target: 79
PASS 3
HIGH: 17 LOW: 14 MID: 15 Target: 79
PASS 4
HIGH: 14 LOW: 14 MID: 14 Target: 79
PASS 5
s
RESULT: NOT FOUND -1

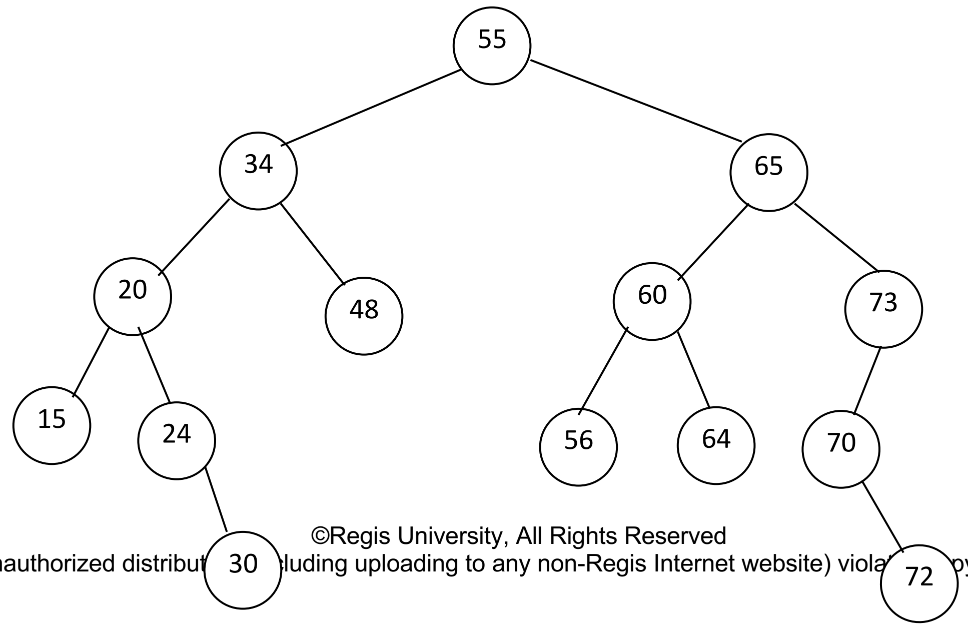
7) Assume you have a 15-element hash table which uses the hash function
H(key) = key mod 15
and quadratic probing to resolve collisions.
If the following keys were inserted into the hash table, in the order shown: **29, 31, 22, 44, 45**
where will each be placed (i.e. at what index)? *(8 pts)*

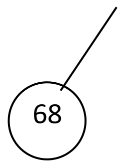
Number	*Index*
29	*14*
31	*1*
22	*7*
44	*0*
45	*4*

8) Begin with an empty binary search tree. (5 *pts*)
Insert the following keys, in the given order:
58, 72, 26, 39, 31, 79, 88, 63, 42, 15, 81
Draw the resulting tree.
The resulting tree was drawn given the above values.



9) Starting with the binary search tree shown below. (*5 pts each*)





a) List the order the nodes will be visited, using pre-order traversal.

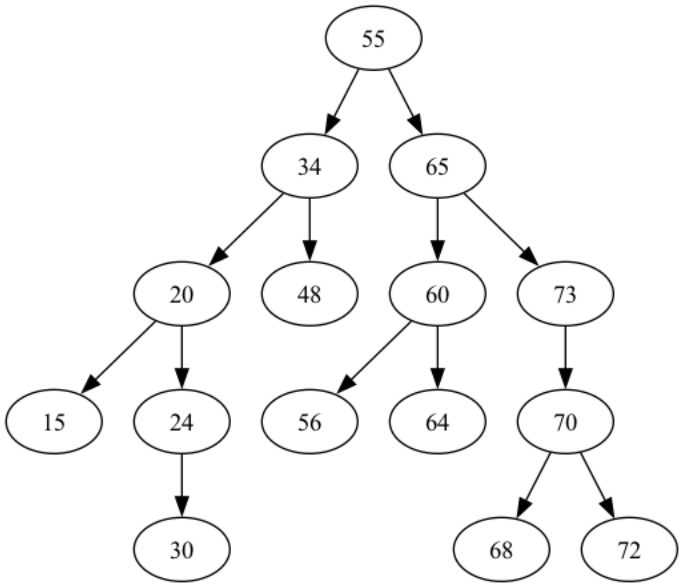
The nodes will be visited in the following order, array indices serve as priority/visit order.
[55, 34, 20, 15, 24, 30, 48, 65, 60, 56, 64, 73, 70, 68, 72]

b) List the order the nodes will be visited, using post-order traversal.

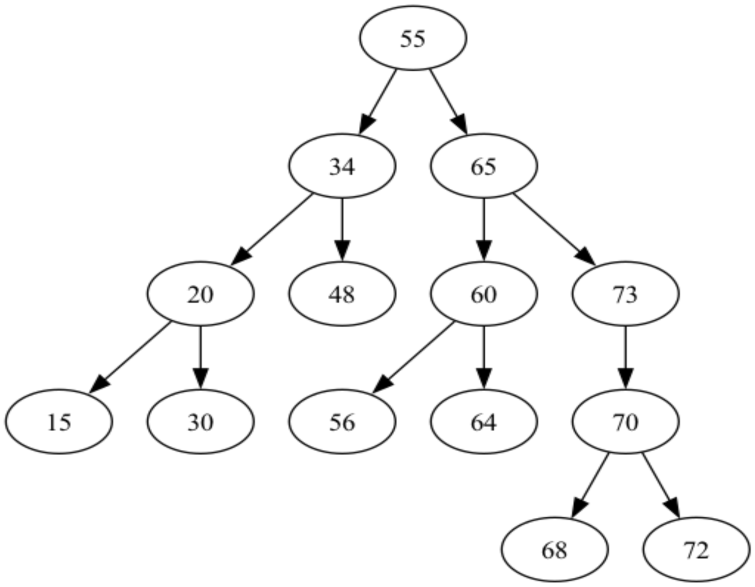
The nodes will be visited in the following order, array indices serve as priority/visit order.
[15, 30, 24, 20, 48, 34, 56, 64, 60, 68, 72, 70, 73, 65, 55]

c) Deletion algorithms either use the **left** subtree or **right** subtree to find the node to replace the deleted node (must use ***same choice for all deletions***).
State which subtree you will use.
Then show the resulting tree, after delete nodes 24, 73, 34, 65, 55, in that order.

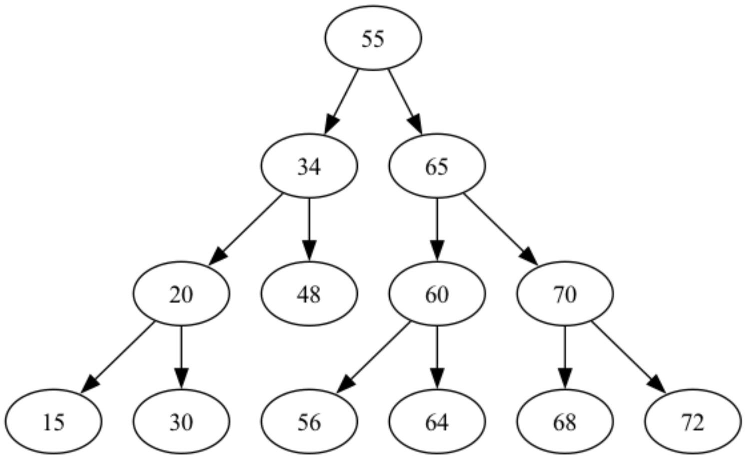
The original tree looks like:



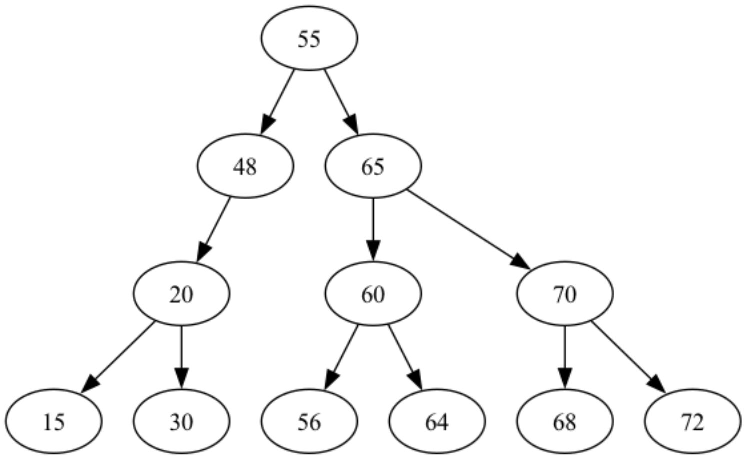
Deleting 24 results in



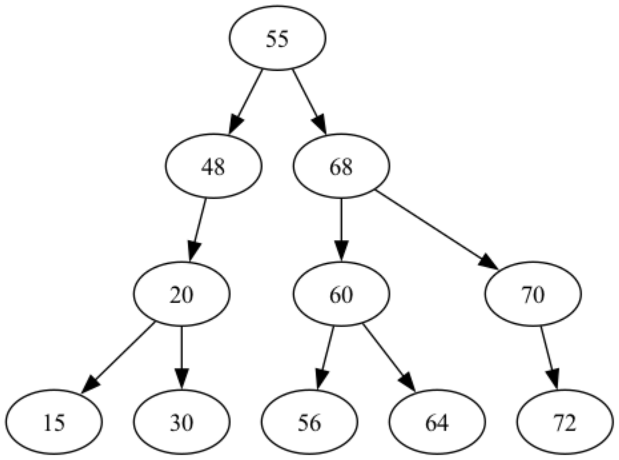
Deleting 73 results in



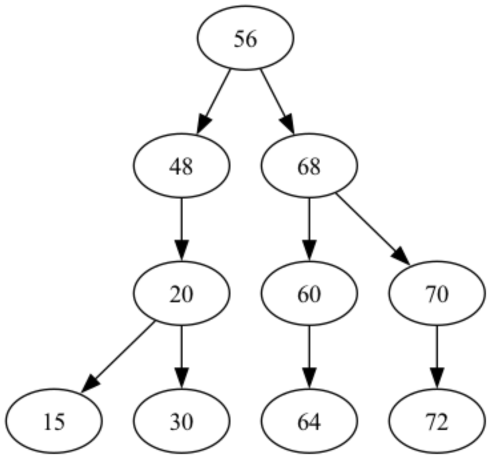
After deleting 34



After deleting 65



After deleting 55



10) Complete problem **A1.9** on page 49 of your textbook (*8 pts*)
Present the algorithm in pseudocode.
Do not use any built-in code library methods in your solution.

Proposal:

- use the definition of the number 0-9 as single bytes to split the string into single digits.
- Subtract 0 from each numeric character to obtain the int value
- Multiply the current sum by ten and then add the next digit int value.
- Do this using a cast or inherent safe type conversion given the languages properties
-

PseudoCode

```
 def convert_string_to_numbers(number_list:str) ->int:
        numbers = numberlist.split(b'"")
        converted_numbers = []
        for num in numbers:
                number = number * 10 + numeric_represent(num-
[ASCII_VALUE_FOR_0])
```

Done mostly in python this relies on the idea that you can split a string by bytes. Use Unicode. And obtain the Unicode point for a single character string.

11) Compare and contrast some of the previously studied data structures:
Linked lists
Binary trees
Discuss advantages and disadvantages of each, in terms of the time it takes to **add** and **delete** elements within the data structure, and the **memory required** for the data structure. (*5 pts*)

Linear lists with nodes linked to each other. Inherently introduces long geography to the traversal given only node functions rather than indexing functions. They function as a one dimensional data structure. Linked lists can be fast and easy to construct custom data structures for but operate slower with bigger N. This data structure touches a lot on immutability since this can be a complex issue for certain languages such as Javascript. Linked lists insert much faster than BSTs and their **add** in O(1) time. Deletion also occurs in O(1) time in a singly or doubly linked list (https://www.bigocheatsheet.com/).

In comparison Binary tress are two dimensional. It operates in O(log(n)) in almost every function including **deletion.** Further Binary trees operate consistently well compared to the O(n) operation of search and access complexities for linked lists. In all they are both valuable choices for very different scenarios. A linked list takes more memory than an array, however remains still slightly less than aa binary tree which typically uses aa big more storage for the increase in efficiency.

Utility of application such as the inherent ability of aa problem to take advantage of a data structures structure is one reason why you could choose a binary tree over a linked list. Aside from the dimensional aspects the structures operate very closely at small size Ns. At higher volumes Binary trees certain beat out linked lists.

A final note about these structures is the sorting ability of each. A binary tree is often constructed to automatically sort the values inserted into it, whereas a linked list is not. This can produce a very valuable increase in efficiency when using the structure for organized/formal data access.

Submission

This homework assignment is due by midnight of the date listed on the **Course Assignments by Week** page of the Content.

- Before you submit your homework, append your last name to the front of the Word doc filename. For examples: `Smith-CS324-AltHwk1v1.docx`
- Submit your **.docx** file to the **Hwk Assn 1** Submission Folder (located under **Assignments** tab in online course

WARNING:
*Homework submitted more than **3 days** past the due date will **not** be accepted, and will receive a grade of 0.*