CS304 practice final

The final exam is *closed book.* Pen and/or pencil are the only materials allowed in the examination room. Answer all questions in your examination booklet in the same order as given below.

Q1: give the formal definition for 'big-O' notation. It starts like 'We define by O(g(n)) the set of all functions: {…}.' Draw a pictorial representation with n on the x-axis and showing functions f(n) and g(n).

Q2: the insertion sort algorithm is given below, with the cost and number of operations for each line of code. The selection sort algorithm is also given (python), but without cost or number of operations. Fill in the cost and number of operations for each line of selection sort, and use them to show that selection sort is an $O(n^2)$ algorithm.

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

Selection sort:

```
for i in range(0,len(arr)):
    smallest = 9999999
    smallest_index = 0
    for j in range(i,len(arr)):
        if arr[j] < smallest:
            smallest = arr[j]
            smallest_index = j
    temp = arr[i]
    arr[i] = smallest
    arr[smallest_index] = temp
```

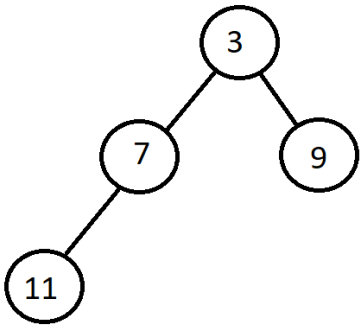
Q3. Give the time complexity for the following operations on a *queue* which is implemented using a python List as the underlying data structure:

Operation:	Time complexity:
q.insert(elem)	O(?)
a = q.remove_front()	O(?)
a = q.front()	O(?)

Q4. Give the time complexity for the following operations on a *stack* which is implemented using a linked list as the underlying data structure:

Operation	Time complexity
s.push(elem)	O(?)
a = s.pop(elem)	O(?)
A = s.top(elem)	O(?)

Q5. Consider the binary heap *h* below:



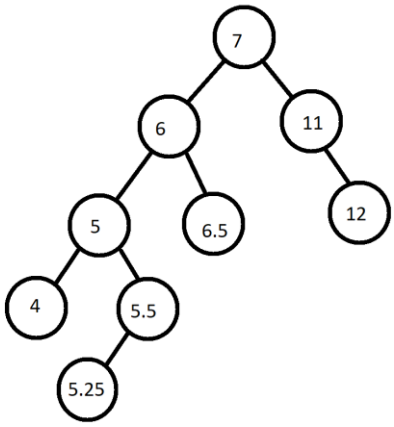
Perform the following operations on the heap, showing the state of the heap after each operation:

a. h.delete_min()
b. h.min()
c. h.insert(3)
d. h.insert(1)
e. h.delete_min()

Q6. consider the following sorted sequence: [1,2,3,4,5,6,7]

a. insert the sequence into an empty binary search tree (BST). Show only the final state of the BST.
b. Insert the sequence into an empty AVL tree. Show the state of the tree after each insertion.

Q7. consider the BST *t* below:

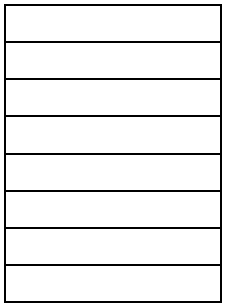


Perform the following operations:

a. t.delete(5),
b. t.delete(6),
c. t.insert(4.5).

Show the BST after each operation

Q8. Consider the empty hash table *t* and hash function below:



```
def hash(str, tableSize){
   int hashCode = 0
   for char c in str
        hashCode += c
   return hashCode % tableSize
}
```

Assume that the += operation when applied to a character uses the integer value of each character according to its position in the alphabet (a=1, b=2, c=3, d=4, etc.). example: the string 'aaa' would have a value of 3 (1+1+1).

Perform the following insertions on the hash table:

a. t.insert('cat', 8)
b. t.insert('rat',8)
c. t.insert('mat',8)
d. t.insert('fat',8)

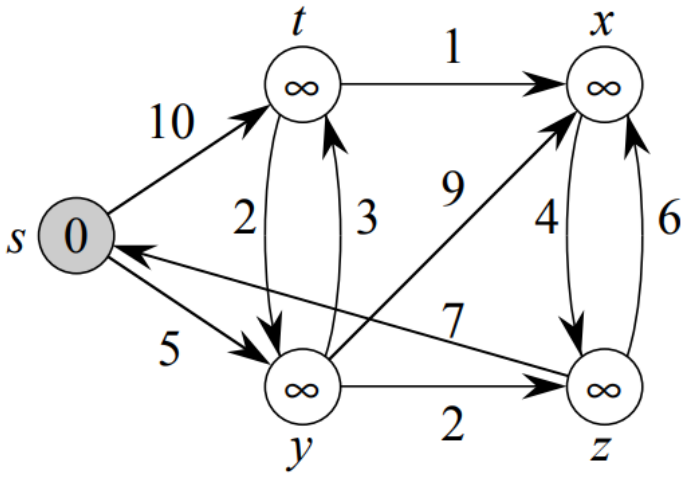
show the hash table after each insertion. If a collision occurs, use linear probing to resolve the collision.

Q9. Consider the graph below and the pseudocode for Dijkstra's algorithm:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0

RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u

DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```



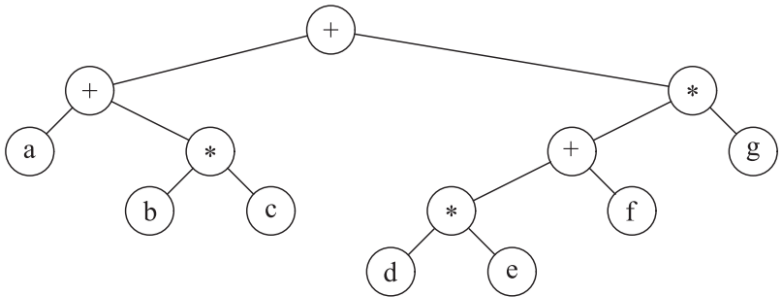
Run Dijkstra's algorithm on this graph to complete the table below:

Vertex	Distance to s	Known?	predecessor
s			
t			
x			
y			
Z			

Q10. Your boss wants you to implement an ordered map abstract data type (ADT). Your boss does not care what data structure you use, but the ordered map must have fast find(elem), fast findMin, and fast insertion/deletion time. Your boss will not accept anything slower than $O(logn)$ for any of these operations. Furthermore, your boss would like to be able to produce a sorted sequence from the map's elements in $O(n)$ time. Name two data structures that will satisfy the requirements and discuss their relative merits (compare them).

Q11. Name an application where a linked list would outperform an array for storing the underlying data of the application.

Q12. Consider the following expression tree:



Expression tree for `(a + b * c) + ((d * e + f ) * g)`

a. produce the postfix version of this expression
b. produce the prefix version of this expression

*hint – you can use a stack to convert the infix to post/pre but its much easier to just traverse the tree.

Q13. is it possible to implement a data structure that allows for O(1) insertion and O(n) ordered traversal? If so, describe such a structure, otherwise, explain why not?