CS6033 Design and Analysis of Algorithms I
Review Assignment
December 10, 2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2182 (Venu Vardhan Reddy Tekula)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)

1. (5 points) Show that an n-element heap has height $\lfloor logn \rfloor$.

   Ans:

   Max nodes in a complete tree of height h is $2^{h+1} - 1$

   Max nodes in a complete tree of height h-1 is $2^h - 1$

   Min number of nodes in a tree of height h is $2^h - 1 + 1$

   This implies that

   $2^h \leq n < 2^{h+1}$
   $h \leq lg(n) < h + 1$

   Since h is an integer, it follows that $h = floor(lg(n))$

2. (10 points) You have been given an array A on which you have already performed the BUILD-MAX- HEAP and converted array A into max-heap. From the resulted array A find the K'th Smallest element, Write a Pseudocode for the same and running time.

   Ans:

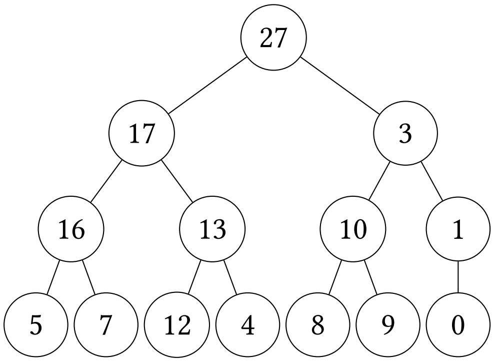
   To find the Kth smallest element in a max-heap, we can use the following algorithm:

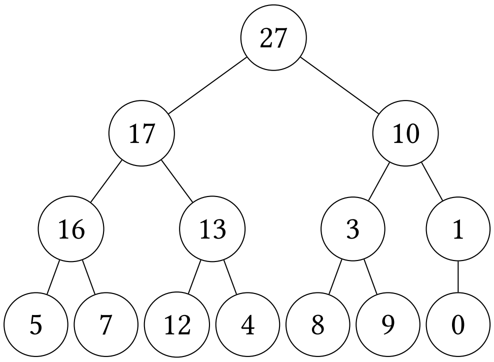
```
def kthSmallest(arr, N, K):

    # Build max heap
    mh = MaxHeap(arr, N)

    # Do extract max (n-k) times
    for i in range(N-K):
        mh.extractMax()

    # Return root
    return mh.getMax()
```

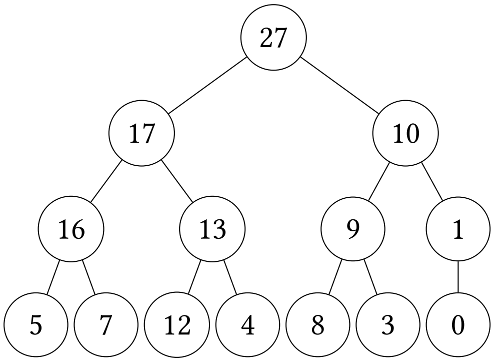
   Time complexity : O((N-K)LogN) excluding the time taken for build max heap

3. (5 points) Using Figure 6.2 as a model, illustrate the operation of MAX-HEAPIFY (A, 3) on the array A=[27,17,3,16,13,10,1,5,7,12,4,8,9,0].

Ans:







4. (10 points) You are eager to invest your money in stock market. You decide to ask your friends for stock tips. Its weekend today and you keep getting continuous suggestions on which stocks to buy.

   Due to not wanting to lose most of your money, you decide to buy the $B < N$ stocks that did the best in the past 6 months time. When the stock market opens back on the coming Monday, you want to quickly be able to determine the best stocks to buy in $O(B)$ time and insert a new stock suggestion in $O(logN)$ time. Partial credit for determining the top B stocks in $O(BlogN)$ time.

   Assume all stock suggestions are unique and Let B be the stocks you invest in and N be the total distinct suggestions.

   Write Pseudocode for the above question and justify your runtime.

   Ans:

We maintain a min heap of size B. As we keep getting suggestions, we insert them into the heap. If the size of the heap reaches B, and if the new suggestion is greater than min of heap(root), then we remove the min and replace with new suggestion and heapify there.

```
def find_top_stocks(suggestions, B):
    heap = Minheap(B)
    for suggestion in suggestions:
        if size(heap) < B:
            heap.insert(suggestion)
        else if (size(heap) == B and suggestion > minimum(heap):
            heap.remove_minimum()
            heap.insert(suggestion)
    return heap
```

In the weekend, as we keep getting suggestions, we insert them into the min heap. So, by monday morning our heap of size B will be ready and it contains the best suggestions. So, Time Complexity to determine top B stocks: O(B) which is the size of the heap.

Another approach is to build the total max heap and on monday keep extracting the maximums of size B. Now, insertion here takes O(logN) (log of size of heap).

5. (5 points) Consider the Quick sort algorithm which sorts elements in ascending order using the first element as pivot. Then which of the following input sequence will require a maximum number of comparisons when this algorithm is applied on it?

   A. 22 25 56 67 89
   B. 52 25 76 67 89
   C. 22 25 76 67 50
   D. 52 25 89 67 76

   Please, justify your answer.

   Ans: Option A will take maximum number of comparisons since the given sequence in option A is sorted array then worst case behaviour occurs for the Quick sort algorithm which use the first element as pivot. Therefore, the input sequence given in 22, 25, 56, 67, 89 will require a maximum number of comparisons.

6. (5 points) What is the running time of QUICKSORT when all elements of array A have the same value? Justify your answer.

   Ans:

   The running time of the QUICKSORT algorithm when all elements of array A have the same value is $\mathcal{O}(n^2)$, where n is the number of elements in A.

   When using the QUICKSORT algorithm, the pivot is chosen as the first element of the input sequence. The algorithm then compares the other elements in the sequence to the pivot and divides them into two groups: elements that are smaller than the pivot, and elements that are larger than the pivot. The algorithm then repeats this process on each of the two groups, until all of the elements are sorted.

In the case where all of the elements in array A have the same value, the pivot will always be the same element, and all of the other elements will be divided into the same two groups (either all elements will be smaller than the pivot, or all elements will be larger than the pivot). This means that the algorithm will have to perform the same number of comparisons for each element in the array, resulting in a running time of $\mathcal{O}(n^2)$.

7. (10 points) Show how to sort n integers in the range 0 to n^2–1 in $\mathcal{O}(n)$ time. Write a Puesdocode for the same and justify running time.

   Ans:

   To sort n integers in the range 0 to $n^2 - 1$ in O(n) time, we can use the following approach:

```
def countSort(arr, n, exp):
  ret = [0] * n
  count = [0]*n
  for i in range(n):
      count [arr[i]// exp) % n] += 1

  for i in range(1, n):
      Count[i] += count[i-1]
  for i in range(n-1, -1, -1):
      ret[count[(arr[i]//exp)%n] -1] = arr[i]
      count[(arr[i]//exp) % n] -= 1

  for i in range(n):
      Arr[i] = ret[i]

 def radixSort(arr, n):
  countSort(arr, n, 1)
  countSort(arr, n,n)
```

   Radix sort takes $O(d*(n+b))$ when there are d digits in input integers. Since n^2-1 is max possible value $d = O(log_b(n)$ → Overall complexity becomes $O((n+b)*O(log_b(n))$. We may use log properties to change the base b to n → runtime becomes $O((n+b)*O(1)) = O(n+b)$ $O(n)$

8. (5 points) As your midterm ends Prof. given you the List of midterm marks of all the students is stored in an array A. Prof. wants you to implement an algorithm to know the minimum and maximum marks using the least number of comparisons. Write a Puesdocode for the same and justify running time.

   Ans:

   Divide and Conquer approach

   Divide the array into two parts and compare the maximums and minimums of the two parts to get the maximum and the minimum of the whole array.

```
def getMinMax(low, high, arr):
    arr_max = arr[low]
    arr_min = arr[low]
    # If there is only one element
    if low == high:
        arr_max = arr[low]
        arr_min = arr[low]
        return (arr_max, arr_min)
    # If there is only two element
    elif high == low + 1:
        if arr[low] > arr[high]:
            arr_max = arr[low]
            arr_min = arr[high]
        else:
            arr_max = arr[high]
            arr_min = arr[low]
        return (arr_max, arr_min)
    else:
        # If there are more than 2 elements
        mid = int((low + high) / 2)
        arr_max1, arr_min1 = getMinMax(low, mid, arr)
        arr_max2, arr_min2 = getMinMax(mid + 1, high, arr)
    return (max(arr_max1, arr_max2), min(arr_min1, arr_min2))
arr_max, arr_min = getMinMax(0, len(arr)-1, arr)
```

T(n) = T($\lfloor n/2 \rfloor$) + T($\lceil n/2 \rceil$) + 2

T(2) = 1

T(1) = 0

If n is a power of 2, then we can write T(n) as:

T(n) = 2T(n/2) + 2

After solving the above recursion, we get

T(n) = 3n/2 -2

9. (5 points) In the $\mathcal{O}(n)$ worst case deterministic select algorithm, the pivot was found by dividing the input elements into groups of 5 and then finding the median of their medians.

   Would the algorithm still run in $\mathcal{O}(n)$ time if we divided the elements into groups of 3 instead of 5? Justify your answer.

   Ans:

   Yes, the O(n) worst case deterministic select algorithm would still run in O(n) time if we divided the elements into groups of 3 instead of 5. The specific size of the groups used to find the pivot does not affect the asymptotic running time of the algorithm. The key factor is that the pivot is chosen in such a way that it splits the input array into two subarrays of roughly equal size, which allows us to recursively apply the algorithm to each subarray.

As long as the pivot is chosen in this way, the algorithm will run in O(n) time in the worst case.

Therefore, we have that it grows more quickly than linear.

10. (10 points) Show how to determine the median of a 5-element set using only 6 comparisons.

    Ans:

    To determine the median of a 5-element set using only 6 comparisons, we can use the following algorithm:

    (a) Sort the first two pairs. [2 comparisons]

    (b) Order the pairs w.r.t. their respective larger element. [1 comparison]

    (c) Call the result [a,b,c,d,e]; we know a¡b¡d and c<d. Now there are 3 elements less than d, hence d can't be the median(i.e. 3rd element of the sorted array)

    (d) Without loss of generality, say $c < e$ and $a < b$ (known already). Order the pairs w.r.t. their respective larger element. Without loss of generality, $a < b < e$ and $c < e$. Compare c & b, greater one is the median. [3 comparisons - 1 for c & 6, 1 for ordering the pairs and 1 for b & c] Hence, total number of comparisons is equal to 6.

11. (10 points) You have given the linked list L. Design an algoritham : if you get head of linked list and the one number, you have to remove that number from linked list. If that number is not there in the list, return "No such number exists in this linked list". Write a Puesdocode for the same and justify running time.

    Ans:

    One possible algorithm to remove a number from a linked list is as follows:

```
procedure REMOVE_NUMBER($head, number$)
    $current \leftarrow head$
    $previous \leftarrow null$
    while $current \neq null$ do
        if $current.data = number$ then
            if $previous = null$ then
                $head \leftarrow current.next$
            else
                $previous.next \leftarrow current.next$
            end if
            return $head$
        else
            $previous \leftarrow current$
            $current \leftarrow current.next$
        end if
    end while
    return "No such number exists in this linked list"
end procedure
```

The running time of this algorithm is $\mathcal{O}(n)$ where n is the number of nodes in the linked list, because the algorithm iterates through the entire list in the worst case. This is the optimal running time for this problem, because the number to be removed must be found in the list before it can be removed.

12. (5 points) Show how to implement a queue using two stacks. Analyze the running time of the queue operations.

    Ans:

    To implement a queue using two stacks, we can use the following pseudo-code:

```
function ENQUEUE(x)
    $stack1.push(x)$
end function
function DEQUEUE
    if $stack2$ is empty then
        while $stack1$ is not empty do
            $y \leftarrow stack1.pop()$
            $stack2.push(y)$
        end while
    end if
    $z \leftarrow stack2.pop()$
    return $z$
end function
```

    The running time of the enqueue operation is $\mathcal{O}(1)$, since it performs a constant number of operations regardless of the size of the queue. The running time of the dequeue operation is $\mathcal{O}(n)$, where n is the number of elements in the queue, since it may need to move all of the elements from *stack1* to *stack2*. Therefore, the overall running time of the queue operations is $\mathcal{O}(n)$.

13. ((10 points) Given a singly linked list, write a pseudo-code that returns true if the given list is a palindrome, else false. (Note - You have to use stack in this question). Also provide the running time for the same.

    Ans:

    To check if a singly linked list is a palindrome, we can use the following pseudo-code:

    The running time of this algorithm is O(n), where n is the number of elements in the linked list. The algorithm performs a constant number of operations on each element in the linked list, so the time taken by the algorithm grows linearly with the size of the input.

14. (5 points) For the following array, $[10, 22, 37, 40, 52, 60, 70, 72, 75]$. Compute $h_{a,b}(k) = ((ak + b) \mod p) \mod m$, where $a = 3, b = 42, p = 101, m = 9$. In-case of collisions, mention the keys that collide along with the index they collide at.

    Ans:

```
function isPalindrome(head)
    $slow \leftarrow head$
    $stack \leftarrow []$
    $ispalin \leftarrow True$
    while $slow \neq null$ do
        $stack.append(slow.data)$
        $slow \leftarrow slow.ptr$
    end while
    while $head \neq null$ do
        $i \leftarrow stack.pop()$
        if $head.data = i$ then
            $ispalin \leftarrow True$
        else
            $ispalin \leftarrow False$
            break
        end if
        $head \leftarrow head.ptr$
    end while
    return $ispalin$
end function
```

To compute ha,b(k) for each element in the array, we need to use the formula $h_{a,b}(k) = ((ak+b) \mod p) \mod m$, where a, b, p, and m are given. For each element k in the array, we can compute $h_{a,b}(k)$ as follows:

$$h_{a,b}(10) = ((3*10+42) \mod 101) \mod 9 = (72 \mod 101) \mod 9 = 72 \mod 9 = 0$$
$$h_{a,b}(22) = ((3*22+42) \mod 101) \mod 9 = (108 \mod 101) \mod 9 = 7 \mod 9 = 7$$
$$h_{a,b}(37) = ((3*37+42) \mod 101) \mod 9 = (155 \mod 101) \mod 9 = 54 \mod 9 = 0$$
$$h_{a,b}(40) = ((3*40+42) \mod 101) \mod 9 = (162 \mod 101) \mod 9 = 61 \mod 9 = 7$$
$$h_{a,b}(52) = ((3*52+42) \mod 101) \mod 9 = (198 \mod 101) \mod 9 = 97 \mod 9 = 7$$
$$h_{a,b}(60) = ((3*60+42) \mod 101) \mod 9 = (222 \mod 101) \mod 9 = 20 \mod 9 = 2$$
$$h_{a,b}(70) = ((3*70+42) \mod 101) \mod 9 = (252 \mod 101) \mod 9 = 50 \mod 9 = 5$$
$$h_{a,b}(72) = ((3*72+42) \mod 101) \mod 9 = (258 \mod 101) \mod 9 = 56 \mod 9 = 2$$
$$h_{a,b}(75) = ((3*75+42) \mod 101) \mod 9 = (267 \mod 101) \mod 9 = 65 \mod 9 = 1$$

Keys 10, 37 collide at index 0

Keys 22, 40, 52 collide at index 7

Keys 60, 72 collide at index 2

15. ((15 points) In the following, say our hash function for n is n % 20 and our second hash function, where applicable, is (n % 7) + 1.

    (a) (5 points) Say we use separate chaining for collision resolution. Beginning with an empty hash table, we insert the following [8, 38, 3, 5, 28, 18, 65, 83] elements. How will the hash table look after the final insertion? If we then search for 48, how many of the inserted keys will we look at?

(b) (5 points) Say we use linear probing for collision resolution instead. What is the answer to the above questions?

(c) (5 points) Say we use double hashing for collision resolution. Now what is the answer to the above questions?

Ans:

(a) In separate chaining, when a collision occurs, the keys that map to the same slot are stored in a separate data structure such as a linked list or a binary tree. In this case, we are using the first hash function, which maps keys to integers between 0 and 19, inclusive, by taking the remainder when the key is divided by 20.

When we insert the elements [8, 38, 3, 5, 28, 18, 65, 83] into an empty hash table, the hash table will look as follows:

0	null
1	null
2	null
3	3 → 83
4	null
5	5 → 65
6	null
7	null
8	8 → 28
9	null
10	null
11	null
12	null
13	null
14	null
15	null
16	null
17	null
18	38 → 18
19	null

After the final insertion, the hash table will have 4 non-null slots, where each slot contains one or more keys that have been hashed to the same value.

When we search for 48, we first apply the hash function to the key to get its index in the hash table. In this case, the hash function will return 8 because 48 % 20 = 8. We then look in the slot at index 8 and find that it contains the keys 8 and 28. Since 48 is not equal to any of these keys, we conclude that 48 is not in the hash table.

(b) In linear probing, when a collision occurs, the next available slot in the hash table is used to store the key. In this case, we are using the first hash function, which maps keys to integers between 0 and 19, inclusive, by taking the remainder when the key is divided by 20.

When we insert the elements [8, 38, 3, 5, 28, 18, 65, 83] into an empty hash table using linear probing, the hash table will look as follows:

0	null
1	null
2	null
3	3
4	83
5	5
6	65
7	null
8	8
9	28
10	null
11	null
12	null
13	null
14	null
15	null
16	null
17	null
18	38
19	18

After the final insertion, the hash table will have 8 non-null slots, where each slot contains a key that has been hashed to the corresponding index.

When we search for 48, we first apply the hash function to the key to get its index in the hash table. In this case, the hash function will return 8 because 48 % 20 = 8. We then look in the slot at index 8 and find that it contains the key 8. Since 48 is not equal to 8, we continue to the next slot, which is at index 9. This process continues until we find a null slot or the key 48. Since 48 is not in the hash table, the search will not be successful.

(c) In double hashing, when a collision occurs, the key is hashed to a second value using a different hash function, and the key is stored in the slot at the index that is the result of this second hash function. In this case, we are using the first hash function, which maps keys to integers between 0 and 19, inclusive, by taking the remainder when the key is divided by 20. We are also using the second hash function, which maps keys to integers between 1 and 7, inclusive, by taking the remainder when the key is divided by 7 and adding 1.

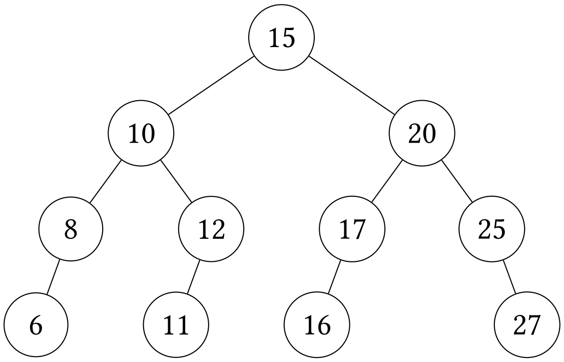
When we insert the elements [8, 38, 3, 5, 28, 18, 65, 83] into an empty hash table using double hashing, the hash table will look as follows:

0	null
1	null
2	null
3	3
4	null
5	5
6	null
7	null
8	8
9	28
10	83
11	65
12	null
13	18
14	null
15	null
16	null
17	null
18	38
19	null

After the final insertion, the hash table will have 8 non-null slots, where each slot contains one or more keys that have been hashed to the same value using the first hash function and the same value using the second hash function.

When we search for 48, the algorithm will look at 8's bucket and move up until it either reaches 48 or it reaches an empty slot. In this case it looks at the key in slot 8 before reaching slot 15 and halting on this empty slot; so it sees one key only.

16. ((10 points) Given the below binary search tree. Write inorder, preorder and postorder Traversal for the same. Justify the behavior of the inorder traversal output.



Ans: The inorder traversal (left, root, right) of the given binary search tree is
6, 8, 10, 11, 12, 15, 16, 17, 20, 25, 27.

The preorder traversal (root, left, right of the given binary search tree is
15, 10, 8, 6, 12, 11, 20, 17, 16, 25, 27.

The postorder traversal (left, right, root) of the given binary search tree is
6, 8, 11, 12, 10, 16, 17, 27, 25, 20, 15.

In an inorder traversal of a binary search tree, the nodes are visited in the order of their values. In the given binary search tree, the left subtree of a node contains all nodes with values that are less than the value of the node, and the right subtree of a node contains all nodes with values that are greater than the value of the node. This means that in an inorder traversal of the given binary search tree, the nodes in the left subtree of a node are visited before the node, and the nodes in the right subtree of a node are visited after the node. This explains the behavior of the inorder traversal output, which is sorted in ascending order.

17. ((5 points) Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

    Ans: Yes, we can prove this using proof of contradiction.

    Suppose that a node in a binary search tree has two children and its successor has a left child. Let the value of this node be x, the value of its left child be y, the value of its right child be z, and the value of its successor's left child be w. Since the successor of x has a left child, we know that the value of w is greater than y and less than z.

    Since the value of w is greater than y, it follows that w is not in the left subtree of x. This means that w must be in the right subtree of x, which means that w is also in the right subtree of z. However, since the value of w is less than z, this contradicts the fact that z is the smallest value in the right subtree of x. Therefore, our assumption that the successor of x has a left child leads to a contradiction, and we can conclude that the successor of x cannot have a left child.

    We can prove the same thing for the predecessor of x using a similar argument. Suppose that the predecessor of x has a right child. Let the value of this right child be v. Since the value of v is less than z, it follows that v is not in the right subtree of x. This means that v must be in the left subtree of x, which means that v is also in the left subtree of y. However, since the value of v is greater than y, this contradicts the fact that y is the largest value in the left subtree of x. Therefore, our assumption that the predecessor of x has a right child leads to a contradiction, and we can conclude that the predecessor of x cannot have a right child.

    In summary, we have shown that if a node in a binary search tree has two children, then its successor cannot have a left child and its predecessor cannot have a right child. This completes the proof.

18. ((5 points) Suppose that instead of always selecting the first activity to finish, you instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

    Ans:

    This approach is a greedy algorithm because it makes a locally optimal choice at each step, without considering the global effect of these choices. Specifically, at each step, the algorithm selects the last activity to start that is compatible with all previously selected activities, without considering whether this choice will lead to an optimal solution overall.

To prove that this approach yields an optimal solution, we can use the following argument:

(a) Suppose that there is an optimal solution to the activity selection problem that includes the activity A.

(b) Since A is included in the optimal solution, there must exist a subset of activities B that are compatible with A and are included in the optimal solution.

(c) The last activity in B to start must be compatible with all of the activities in B, including A.

(d) Since the last activity in B to start is compatible with A and is included in the optimal solution, it must also be included in any solution that includes A.

(e) Therefore, any solution that includes A must also include the last activity in B to start.

(f) Since the algorithm always selects the last activity to start that is compatible with all previously selected activities, it must also include the last activity in B to start if it includes A.

(g) Therefore, the algorithm must include the same activities as the optimal solution if it includes A.

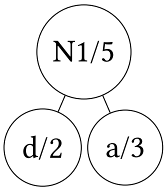
(h) By repeating this argument for all activities in the optimal solution, we can show that the algorithm must include the same activities as the optimal solution.

Therefore, the algorithm yields an optimal solution.

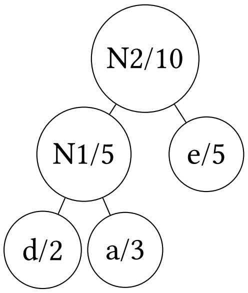
19. ((10 points) What is an optimal Huffman code for the following set of frequencies?

a:3,b:6,c:8,d:2,e:5,f :9,g:13,h:12

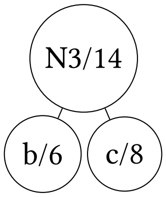
Ans:



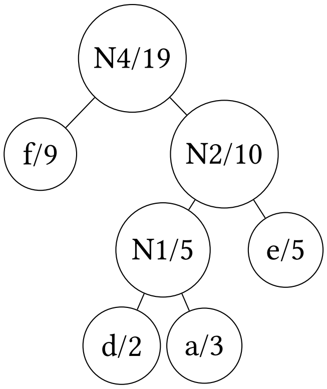
a:3 b:6 c:8 d:2 e:5 f:9 g:13 h:12



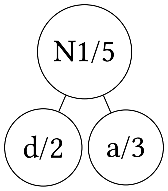
N1:5 b:6 c:8 f:9 g:13 h:12



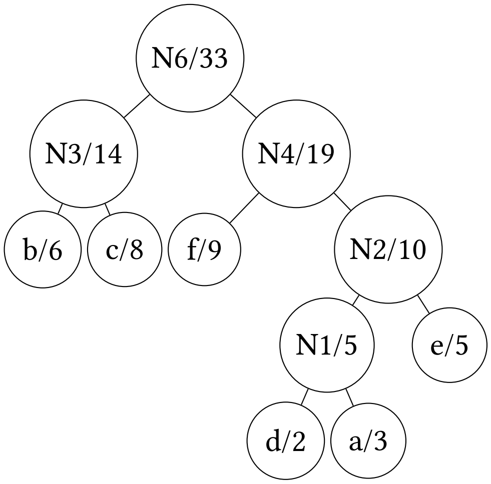
N2:10 b:6 c:8 f:9 g:13 h:12



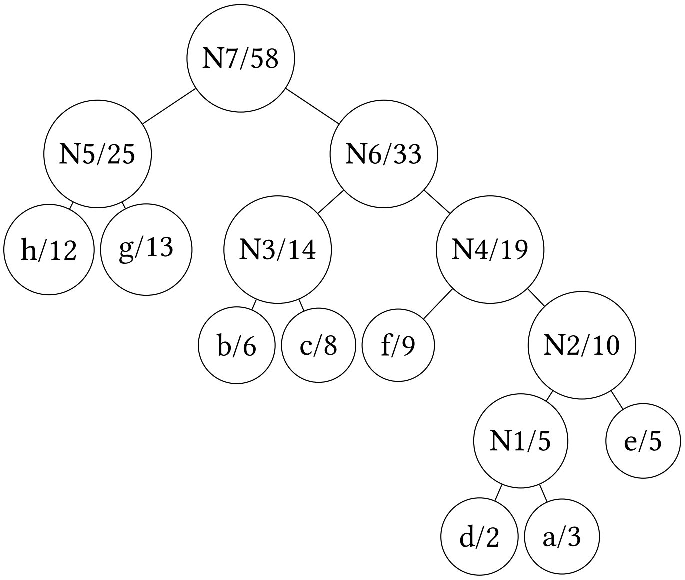
N2:10 N3:14 f:9 g:13 h:12



N4:19 N3:14 g:13 h:12



N4:19 N3:14 N5:25



N6:33 N5:25

```
a:11101
b:100
c:101
d:11100
e:1111
f:110
g:01
h:00
```

20. (10 points) After a horrifying road trip with your friend to California. You now need to return home. You vow to study hard in class (and thus get a job) so next time you visit California you will be driving your own car (you were unaware your friend refused to drink coffee). After the horrifying experience on the trip down (you nearly crashed 42 times), you have revised your return plan. On your trip home, you refuse to travel more than 300 miles per day. You have the list of hotel from last time at mile markers $m_1, m_2, ..., m_n$. You wish to get home in as few days as possible.

    Write Pseudocode for the above question and justify your runtime.

    Ans:

    One possible algorithm to find the minimum number of days needed to return home while traveling no more than 300 miles per day is as follows:

```
procedure $minDaysToReturn$(miles,hotels)
    days = 0
    position = 0
    while position $<$ miles do
        maxDistance = 300
        for i in range(position, len(hotels)) do
            if hotels[i] - position $<$ maxDistance then
                maxDistance = hotels[i] - position
            end if
        end for
        position += maxDistance
        days += 1
    end while
    return days
end procedure
```

This algorithm will find the minimum number of days required to return home by iterating over the hotels and we are finding the maximum distance (¡300 miles) that can be traveled on each day. It will then update the current position and number of days accordingly. The run-time of this algorithm is O(n)