York University
EECS 2011Z Winter 2018 – Problem Set 3
Instructor: James Elder

Solutions

1. Prove that $n \log n - n$ is $\Omega(n)$.

   - Answer:
     $\log n \geq 2 \ \forall n \geq 4$. Thus $n \log n - n \geq n \ \forall n \geq 4 \rightarrow n \log n - n \in \Omega(n)$.

2. Show that n^2 is $\Omega\left(n \log n\right)$.

   - Answer: We seek a $c > 0, n_0 > 0 : \forall n \geq n_0,\ n^2 \geq cn \log n \leftrightarrow n \geq c \log n$. Let $c = 1$. Then we require that $n \geq \log n$. This is satisfied $\forall n \geq 1$. Thus n^2 is $\Omega\left(n \log n\right)$.

3. Algorithm A is $\Omega\left(n^2\right)$ and Algorithm B is $\mathcal{O}(n \log n)$. If (worst-case) asymptotic run time is your only concern, which algorithm should you choose? Is this guaranteed to be the right choice?

   - Answer: One should choose Algorithm B, since its run time is guaranteed to be $n \log n$ at worst, whereas Algorithm A cannot be better than n^2, which is slower. This is guaranteed to be the right choice.

4. State the definition of $\mathcal{O}()$ and use it to show that $f(n) = 3n^3 + 5n^2 \log n + 9 \cos(2\pi n)$ is $\mathcal{O}\left(n^3\right)$.

   - Answer:

     $$f(n) \in \mathcal{O}\left(g(n)\right) \iff \exists n_0, c : f(n) \leq cg(n) \forall n \geq n_0.$$

     Let $n_0 = 1$. Note that $\forall n \geq n_0$, $\log n \leq n$ and $\cos(2\pi n) \leq n^3$. Thus we have that

     $$f(n) \leq 3n^3 + 5n^3 + 9n^3 = 17n^3, \forall n \geq n_0.$$

     Choosing $c = 17$, we have that $f(n) \leq cn^3 \ \ \forall n \geq n_0$ and therefore that $f(n) \in \mathcal{O}(g(n))$.

5. Please show the hash table that results from adding the keys below to an initially empty map using hash function $h(k) = k \mod 11$ and linear probing.

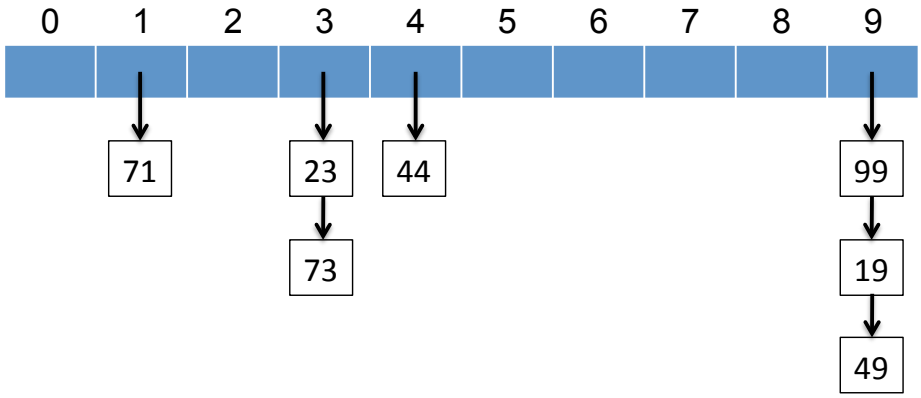
   Keys to add (in the order given): 98, 31, 39, 21, 33, 91, 90, 34

21	33	90	91	34		39			31	98
0	1	2	3	4	5	6	7	8	9	10

6. **Hashing**

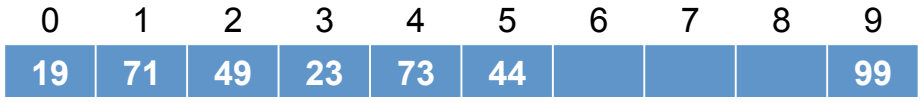
   Given input keys $\{71, 23, 73, 99, 44, 19, 49\}$ and hash function $h(k) = k \mod 10$, show the resulting hash table based upon:

   (a) Separate chaining



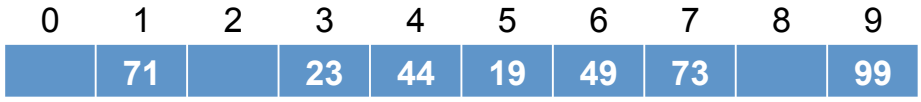
- Answer:

(b) Linear probing



- Answer:

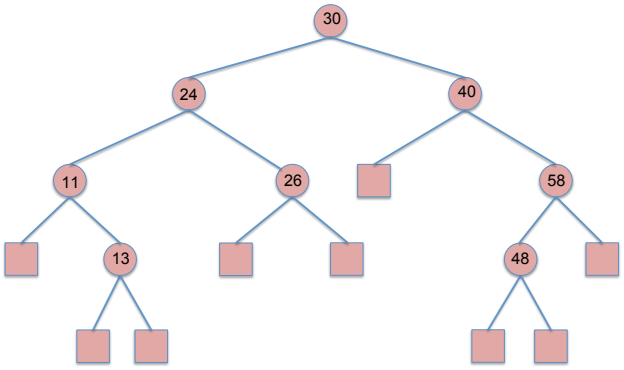
(c) Double hashing with secondary hash function $h'(k) = 7 - k \bmod 7$



- Answer:

7. Binary Search Trees
Insert, into an empty binary search tree, entries with keys 30, 40, 24, 58, 48, 26, 11, 13 (in this order). Draw the tree after each insertion.

- Answer: The final tree should appear as follows:



8. Nearest node in BST
Given a binary search tree T and a number k, the algorithm $Nearest(k, v)$ should return the key in the subtree of T rooted at vertex v that is closest in value to k. For example, if $k = 7$ and the subtree contains keys 17, 5, 13, and 11, then the algorithm should return 5. In the event of a tie, either key can be returned.

(a) Design a recursive algorithm for $Nearest(k, v)$. Describe your algorithm using concise pseudocode. You can assume access to an object T that refers to the tree and supports the following operations

- T.isExternal(v): Return true if v is an external node, false otherwise.
- T.left(v): return the left child vertex of v

- T.right(v): return the right child vertex of v
- key(v): return the key of vertex v

You can assume that v is a valid vertex of T. You may also assume access to a constant k_∞ that is maximally distant from all keys.

Answer:

```
1  Nearest(k,v)
2  if T.isExternal(v) return k_∞
3  if k ≤ key(v) then
4     r = Nearest(T.left(v), k)
5  else
6     r = Nearest(T.right(v), k)
7  end
8  if (r == k_∞ or |r − k| ≥ |key(v) − k|) then
9     return key(v)
10 else
11    return r
```

(b) What is the asymptotic running time of your algorithm for a tree with n nodes?

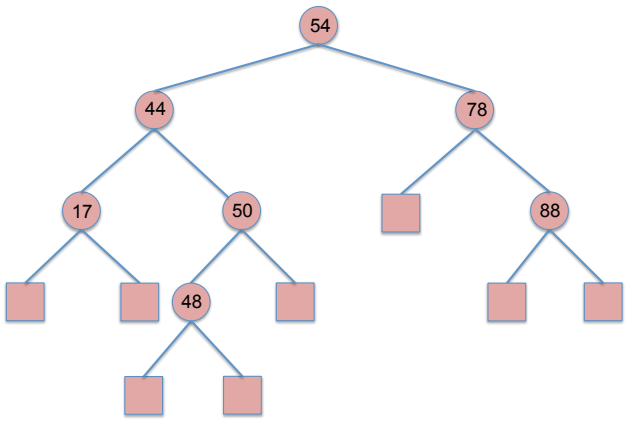
- Answer: $O(n)$. This occurs when the tree is completely imbalanced and the search key is nearest to the bottom node's key.

(c) What is the asymptotic running time of your algorithm for a tree with n nodes, if the tree is an AVL tree?

- Answer: $O(\log n)$, since an AVL tree is balanced, and has $O(\log n)$ height.

9. AVL Trees
Insert, into an empty binary search tree, entries with keys 62, 44, 78, 17, 50, 88, 48, 54 (in this order). Now draw the AVL tree resulting from the removal of the entry with key 62.

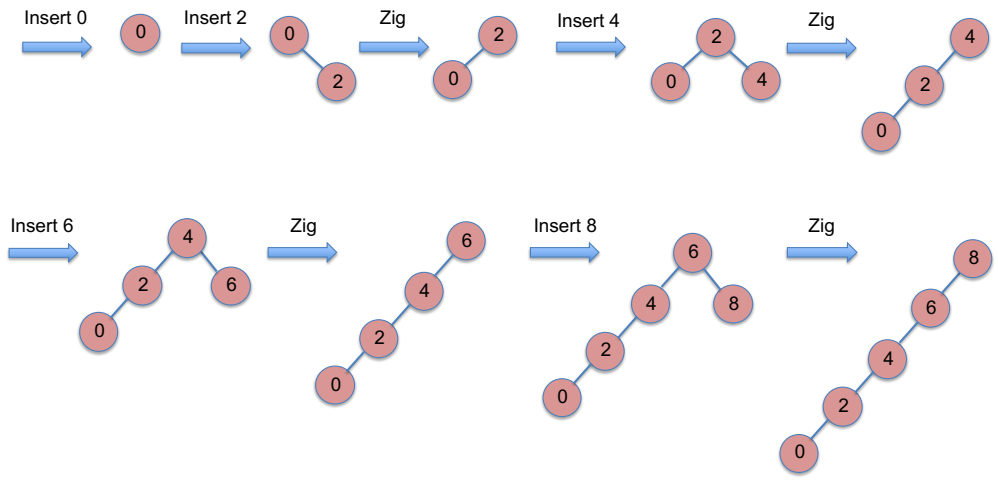
- Answer: The final tree should appear as follows:



10. Splay Trees
Perform the following sequence of operations in an initially empty splay tree and draw the tree after each set of operations.

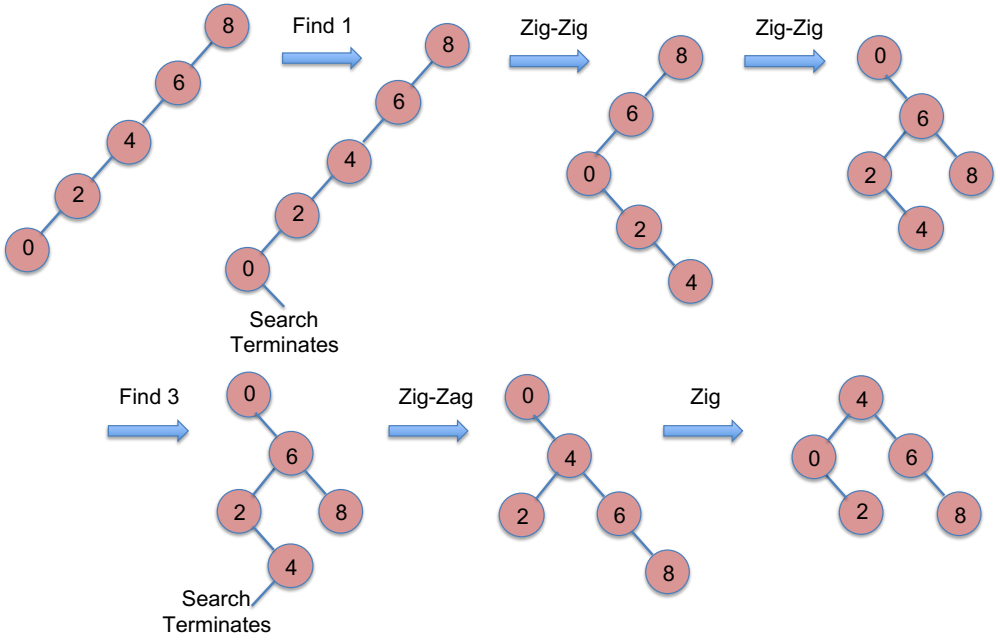
(a) Insert keys 0, 2, 4, 6, 8, in this order.

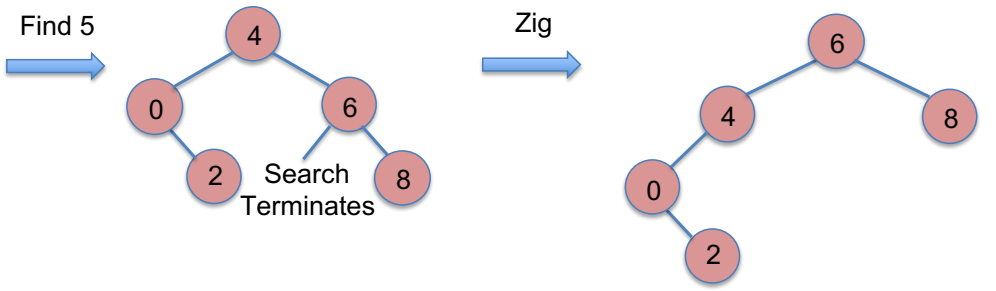
- Answer:

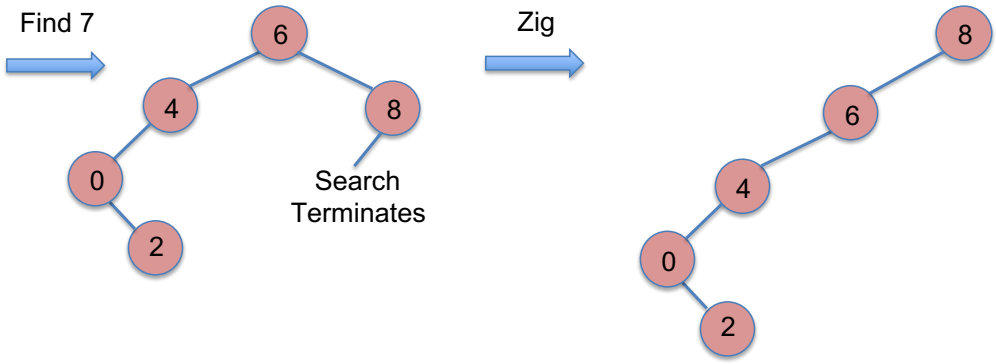


(b) Search for keys 1, 3, 5, 7, in this order.

- Answer:

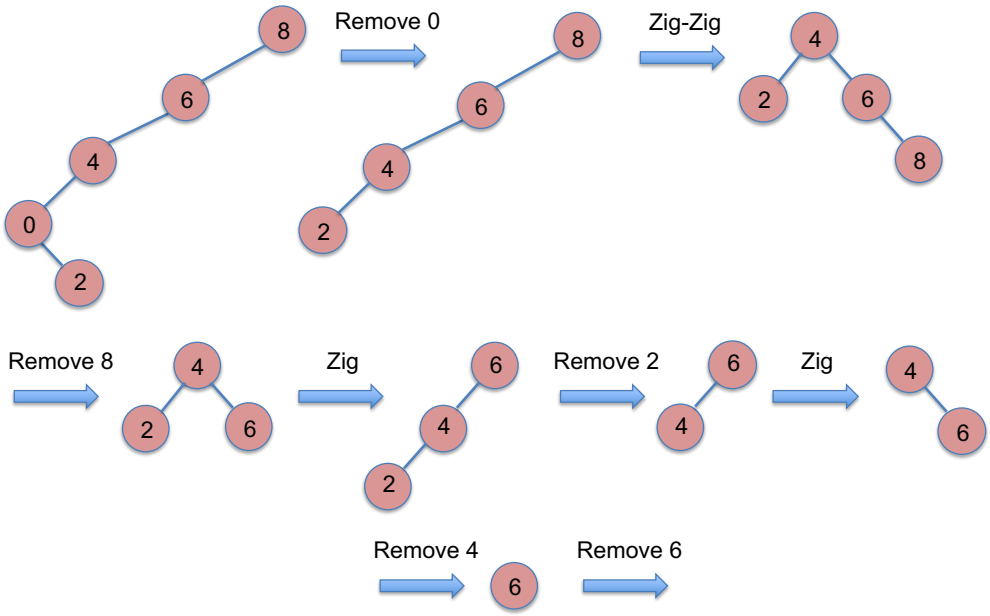






(c) Delete keys 0, 8, 2, 4, 6 in this order.

- Answer:



11. Comparison Sorts
Of the $n!$ possible inputs to a given comparison-based sorting algorithm, what is the absolute maximum number of inputs that could be sorted with just n comparisons?

- Answer: A decision tree of height n can store 2^n external nodes. Thus a maximum of 2^n inputs can be sorted with n comparisons.

12. Comparison Sorts
Give an example input list for which merge-sort and heap-sort take $\mathcal{O}(n \log n)$ time, but for which insertion sort takes $\mathcal{O}(n)$ time. What if the list is reversed?

- Answer: Merge-sort and heap-sort both take $\mathcal{O}(n \log n)$ time on a sorted list, while insertion sort takes $\mathcal{O}(n)$ time. If the list is reversed, merge-sort and heap-sort still take $\mathcal{O}(n \log n)$ time, while insertion sort takes $\mathcal{O}\left(n^2\right)$ time.

13. Stack-Based Quicksort
Describe in pseudocode a non-recursive version of the quick-sort algorithm that explicitly uses a stack.

Answer:

```
1 QuickSort(A,n)
2   Stack S
3   S.push(<0,n-1>)
4   while ¬S.isempty()
5     <p,q> = S.pop()
6     if q > p
7       r = partition(A,p,q)
8       S.push(<r+1,q>)
9       S.push(<p,r-1>)
```

14. Highest Points
You are implementing an algorithm that draws part of the landscape of a 3D terrain, and you are faced with the following problem: You are given the heights of n points of the terrain's grid, and you need to find the $\lfloor \sqrt{n} \rfloor$ highest of them, returning them in descending order. Note that these heights are real numbers, not integers.

(a) Design an algorithm that does this in $O(n)$ time. Assume that the points are given as an array $A[1 \ldots n]$.
Briefly describe your algorithm using concise pseudocode. Any of the algorithms we have covered in class may be used as subroutines. Refer to them by name (do not re-describe them).

Answer:

```
1 H ← makeMaxHeap(A) /* Linear bottom-up construction */
2 for i = 0 to ⌊√n⌋ − 1
3   B[i] ← H.removeMax()
4 return B
```

(b) Briefly state the asymptotic running time taken by each step of your algorithm, and thus show that it is has a total running time of $O(n)$.

- Answer: Building the heap takes $O(n)$ time. Each removeMax call takes $O(\log n)$ time. Thus the total time is $O(n + \sqrt{n} \log n) = O(n)$.

(c) Provide a Θ bound on the time complexity of this problem. Briefly justify your answer.

- Answer: Any solution entails finding the highest point, which takes $\Omega(n)$ time, since each point must be checked. Since we have an $\mathcal{O}(n)$ solution, the problem is $\Theta(n)$.

15. kth-Largest Element
You are to design an efficient recursive algorithm for finding and returning the k^{th} largest element in an array of n elements. Here k can be any number from 1 to n. For simplicity, you can assume that the elements are all unique.

Your recursive algorithm will have the interface kthLargestElement(A, p, q, k), where A is the array and p and q are the upper and lower index bounds of the current search region within A. Your algorithm will make use of the in-place partition method used in QuickSort. In particular, calling r = partition(A, p, q) will reorganize the contents of A between indices p and q so that $A[i] \leq A[r]$ if $p \leq i < r$ and $A[i] > A[r]$ if $r < i \leq q$, where $p \leq r \leq q$. Note that the partition function returns the pivot index r. Your algorithm would be invoked as kthLargestElement(A, 0, n-1, k), where n is the size of the array.

Important: 1) Do not sort the entire array. **2)** You do not have to implement the partition function.

(a) (12 marks) Describe your algorithm in concise pseudo-code or in English.

Answer:

```
1  kthLargestElement(A,p,q,k)
2   r = partition(A,p,q)
3   kr = q - r + 1 // rank of the pivot
4   if k = kr // found it
5     return A[r];
6   else if k < kr // in big set
7     return kthLargestElement(A, r+1,q,k);
8   else // in small set
9     return kthLargestElement(A,p,r-1,k-kr);
```

(b) (3 marks) What is the worst-case asymptotic running time of your algorithm? Briefly justify your answer.

- Answer: The worst-case running time is $\mathcal{O}\left(n^2\right)$. This occurs when each partition is maximally imbalanced, leading to a recursion tree of depth n, with $\mathcal{O}(n)$ work at each level.

(c) (1 mark) What is the expected asymptotic running time of your algorithm, assuming random input? No justification required.

- Answer: The expected running time is $\mathcal{O}(n)$.

(d) (1 mark) For comparison, suppose you implemented an alternative algorithm that simply sorts the input using QuickSort and then finds the k^{th} largest element in the sorted array. What is the expected asymptotic running time of this algorithm?

- Answer: The expected running time would be $\mathcal{O}(n \log n)$, as for QuickSort.

16. Linear Sorts
Given an array of n integers, each in the range $\left[0, n^2 - 1\right]$, describe a simple method for sorting the array in $\mathcal{O}(n)$ time.

- Answer:
  Solution 1:
  Notice that each integer can be coded by $\lceil \log n^2 \rceil = \lceil 2 \log n \rceil \leq 2 \lceil \log n \rceil$ bits. We choose to represent each integer by two $\lceil \log n \rceil$-bit words, each of which can assume $2^{\lceil \log n \rceil}$ values. We then use radix sort to sort the integers, which takes $O\left(2\left(n + 2^{\lceil \log n \rceil}\right)\right) \in O\left(2\left(n + 2n\right)\right) \in O\left(n\right)$ time.
  Solution 2:
  We use bucket sort with table B of length n, inserting A[i] into $B\left[\lfloor A[i]/n \rfloor\right]$.

17. Deleting Edges in Graphs
Let $G = (V, E)$ be an *undirected* graph with vertex set $\{0, 1, \ldots, |V| - 1\}$ and with $|E|$ edges. We denote with (i, j) the edge connecting vertex i and vertex j, and with $d(i)$ the degree of vertex i.

We define the following two operations:

deleteEdge(i, j)**:** delete from G a given edge (i, j).

deleteIncidentEdges(i)**:** delete from G all the $d(i)$ edges incident on a given vertex i.

Provide a precise analysis of the time complexity of operations

deleteEdge(i,j) and deleteIncidentEdges(i) for the following two representations of graph G:

(a) Graph G is represented by a $|V| \times |V|$ boolean matrix A such that $A[i, j]$ is true if and only if G contains edge (i, j).

- Answer: deleteEdge(i,j) will take $O(1)$ time. deleteIncidentEdges(i) will take $O(|V|)$ time.

(b) Graph G is represented by $|V|$ sequences $S_1, \ldots, S_{|V|}$, where sequence $S_i = \{S_i(1), \ldots, S_i(d(i))\}$ contains the $d(i)$ vertices adjacent to vertex i and is realized by means of a doubly-linked list.

- Answer: deleteEdge(i,j) will take $O(d(i)+d(j))$ time. deleteIncidentEdges(i) will take $O(d(i)+\sum_{j=1}^{d(i)} d(S_i(j)))$.

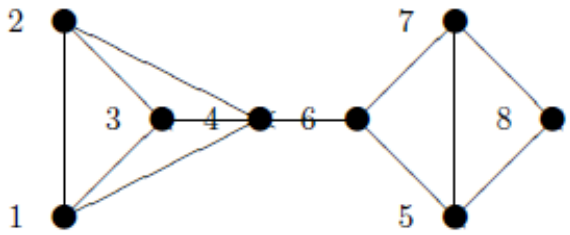
18. DFS and BFS
Let G be an undirected graph whose vertices are labelled by the integers 1 through 8, and having the following edges:

Vertex	Edges
1	2, 3, 4
2	1, 3, 4
3	1, 2, 4
4	1, 2, 3, 6
5	6, 7, 8
6	4, 5, 7
7	5, 6, 8
8	5, 7

Assume that, in a traversal of G, the adjacent vertices of a given vertex are returned in the order above.

(a) Draw G.

- Answer:



(b) Give the sequence of vertices of G visited using a DFS traversal starting at vertex 1.

- Answer: 1, 2, 3, 4, 6, 5, 7, 8.

(c) Give the sequence of vertices visited using a BFS traversal starting at vertex 1.

- Answer: The order is the same in this case: 1, 2, 3, 4, 6, 5, 7, 8